SAKARYA UNIVERSITY
INSTITUTE OF SCIENCE AND TECHNOLOGY

A REAL TIME DEMONSTRATIVE ANALYSIS OF LIGHTWEIGHT PAYLOAD ENCRYPTION IN RESOURCE CONSTRAINED DEVICES BASED ON MQTT

M.Sc. THESIS

Nanabayin MENYAH

Department	**:**	**COMPUTER ENGINEERING**
Field of Science	**:**	**ENGINEERING**
Supervisor	**:**	**Assoc. Prof. Ahmet ÖZMEN**

November 2017

SAKARYA UNIVERSITY
INSTITUTE OF SCIENCE AND TECHNOLOGY

A REAL TIME DEMONSTRATIVE ANALYSIS OF LIGHTWEIGHT PAYLOAD ENCRYPTION IN RESOURCE CONSTRAINED DEVICES BASED ON MQTT

M.Sc. THESIS

Nanabayin MENYAH

Department	**:**	**COMPUTER ENGINEERING**
Field of Science	**:**	**ENGINEERING**
Supervisor	**:**	**Assoc. Prof. Ahmet ÖZMEN**

This thesis has been accepted unanimously by the examination committee on 10.11.2017.

Prof. Dr.	**Prof. Dr.**	**Doç. Dr.**
Cemil ÖZ	**Ahmet ALTUNCU**	**Ahmet ÖZMEN**
Head of Jury	**Jury Member**	**Jury Member**

DECLARATION

I declare that all the data in this thesis was obtained by myself in academic rules, all visual and written information and results were presented in accordance with academic and ethical rules, there is no distortion in the presented data, in case of utilizing other people's works they were referred properly to scientific norms, the data presented in this thesis has not been used in any other thesis in this university or in any other university.

Nanabayin MENYAH
10.11.2017

PREFACE

This is a Master of Science Thesis in 2017 for the Computer Engineering department of Sakarya University (SAÜ), Turkey. I would like to express my sincere appreciation to Almighty God for His protection and to my able supervisor Assoc. Prof. Ahmet Özmen from SAÜ for providing the necessary guidance for the thesis work. I am also thankful for all the lecturers who imparted knowledge unto me at the department of Computer Engineer and helping me write this thesis. Special thanks belong to my family for supporting and encouraging me through my studies in SAÜ. Last but not least, I would also express sincere thanks to all of my colleagues in and out of Sakarya.

TABLE OF CONTENTS

LIST OF SYMBOLS AND ABBREVIATIONS

µs	: Microseconds
ABE	: Attribute base encryption
AES	: Advanced encryption standard
AP	: Access point
Bit	: Binary digit
BSD	: Berkeley software distribution
CBC	: Cipher block chaining
CCM	: Counter with cipher block chaining-message authentication code
CoAP	: Constrained application protocol
CONNACK	: Connection acknowledgement
CORE	: Constrained restful environment
CPU	: Central processing unit
CSV	: Comma separated values
D2D	: Device to device
DES	: Data Encryption Standard
DoS	: Denial of service
DPWS	: Device profile for web service
DTLS	: Datagram transport layer security
DUP	: Duplicate
E2E	: End to end
H2M	: Human to machine
HTTP	: Hypertext transfer protocol
HTTPS	: Hypertext transfer protocol secure
IBM	: International business machine
IDE	: Integrated development environment
IETF	: Internet engineering task force

IoT	: Internet of Things
IP	: Internet protocol
IPv6	: Internet protocol version 6
JSON	: JavaScript object notation
Kb	: Kilobyte
LAN	: Local area network
LLSec	: Link layer security
LSB	: Least significant byte
M2M	: Machine to machine
MAC	: Message authentication code / Media access control
MHz	: Megahertz
MQTT	: Message queue telemetry transport
MSB	: Most significant byte
OASIS	: Organization for the advancement of structured information standards
OCB	: Offset Codebook Mode
OS	: Operating system
PINGREQ	: Ping request
PINGRESP	: Ping response
PUBACK	: Publish acknowledgement
PUBCOMP	: Publish complete
PUBREC	: Publish received
PUBREL	: Publish released
QoS	: Quality of service
RAM	: Random access memory
REST	: Representational state transfer
RFID	: Radio frequency identification
ROM	: Read only memory
RSA	: Rivest-Shamir-Adleman
SOA	: Service oriented approach
SRAM	: Static random-access memory
SSL	: Secured socket layer

SUBACK	: Subscribe acknowledgement
TCP	: Transmission control protocol
TLS	: Transport layer security
UDP	: User datagram protocol
UNSUBACK	: Unsubscribe acknowledgement
UTF	: Unicode transformation format
WSN	: Wireless sensor networks
XML	: Extensible markup language

LIST OF FIGURES

LIST OF TABLES

SUMMARY

Keywords: IoT Security, Networking, MQTT, Publish, Subscribe, Quality of Service, Arduino, Advanced Encryption Standard

Constrained devices are limited in resources namely, memory (ROM and RAM), CPU and battery life (if available). They are often used as sensors that collects data, machine to machine (M2M) or smart devices that control services and electrical appliances. When such devices are connected to a network they form what is called “things” and in a whole, they form part of the “Internet of Things” (IoT).

Message Queue Telemetry Transport (MQTT) is a common light weight, open, simple, client-server publish/subscribe messaging transport protocol useful and efficient for most resource constrained IoT devices that supports three Quality of Service (QoS) levels for reliable communication. It is an essential protocol for communication in constrained environments such as Device to Device (D2D) and Internet of Things (IoT) contexts. MQTT protocol is devoid of concrete security mechanisms apart from Transport Layer Security (TLS) based on Secure Socket Layer (SSL) certificates. However, this is not the lightest of security protocols and increases network overheads especially for constrained devices. About 70 % of most ordinary IoT devices also lack data encryption especially at the client-end which could have been a perfect alternative for TLS.

In this thesis, an experimental setup is designed to demonstrate the effect on network performance of MQTT protocol on a constrained device for different Quality of Service (QoS) and variable size of payloads. The novel part of this study covers client-side encryption of payloads and its effect over network performance. In the experiments, a lightweight encryption of 128-bits Advanced Encryption Standard (AES) is applied on the data. The messages are transferred using the three different QoS levels in MQTT over real wired low-end publish client and low-end subscriber client via a broker server based on different payload sizes. The packets are captured to analyze end-to-end latency, throughput and message loss along with the measurement of encryption and decryption processing time.

According to the results of the experiment, it was concluded that, non-encrypted (plaintext) payload have a lower network load effect and hence produces a relatively better network performance using MQTT in terms of percentage loss and message delivery than the encrypted payload.

MQTT'YE DAYANAN KAYNAK KISITLI CİHAZLARDA HAFİF YÜK ŞİFRELEMESİNİN GERÇEK ZAMANLI BİR DEMONSTRASYON ANALİZİ

ÖZET

Anahtar Kelimeler: IoT Güvenliği, Ağ oluşturma, MQTT, Yayınlama (Publish), Abone Olma (Subscribe), Hizmet Kalitesi, Arduino, İleri Şifreleme Standardı

Kısıtlı cihazların kaynakları, yani bellek (ROM ve RAM), CPU ve pil ömrü (varsa) sınırlıdır. Genellikle, veri toplayan sensörler, makinadan makineye (M2M) veya servisleri ve elektrikli ev aletlerini kontrol eden akıllı cihazlar için puanlar. Bu tür aygıtlar bir ağa bağlandığında "nesnelerin Internet'i" nin (IoT) bir parçasını oluştururlar.

Message Queue Telemetry Transport (yani MQTT), hafif, açık, basit, istemci-sunucu yayın/abone mesajlaşma taşıma protokolüdür. Güvenilir iletişim için üç Hizmet Kalitesi (QoS) seviyesini destekleyen çoğu kaynak kısıtlamalı IoT cihazı için kullanışlıdır ve verimlidir. Cihazdan Cihaza (D2D) ve nesnelerin Internet'i (IoT) bağlamları gibi kısıtlı ortamlarda iletişim için gerekli olan bir protokoldür. MQTT protokolü, güvenli soket katmanı (SSL) sertifikalarına dayalı taşıma katmanı güvenliği (TLS) dışında somut güvenlik mekanizmalarından yoksundur. Bununla birlikte, bu güvenlik protokollerinin en hafif değildir ve özellikle kısıtlı cihazlar için ağ yüklerini artırır. IoT cihazlarının yaklaşık %70'inde özellikle de istemci tarafında veri şifrelemesi yoktur ve TLS için mükemmel bir alternatif olabilir.

Bu tezde, farklı Hizmet Kalitesi (QoS) ve veri yüklerin değişken boyutu için kısıtlı bir cihaz üzerinde MQTT protokolünün ağ performansı üzerindeki etkisini göstermek için bir deney düzeneği tasarlanmıştır. Bu çalışmanın yeni kısmı, yüklerin istemci tarafında şifrelenmesini ve ağ performansı üzerindeki etkisini kapsıyor. Denemelerde, verilere 128-bits ileileri şifreleme standardı (AES) hafif bir şifreleme uygulanmıştır. Mesajlar, farklı yük boyutlarına dayanan bir komisyoncu sunucusu aracılığıyla gerçek kablolu alt uçtakı yayıncılık istemcisi ve düşük uçtakı abone istemcisi üzerinden MQTT'deki üç farklı QoS seviyesini kullanarak aktarılır. Paketler, şifreleme ve şifre çözme işlem süresinin ölçülmesiyle birlikte uçtan uca gecikme, verimlilik ve mesaj kaybı analiz etmek için yakalanır.

Deney sonuçlarına göre, şifrelenmemiş (şifresiz metin) yükün daha düşük bir ağ yük etkisine sahip olduğu ve bu nedenle, yüzde kaybı ve mesaj tesliminde, şifreli yüke göre MQTT'yi kullanarak nispeten daha iyi bir ağ performansı ürettiği sonucuna varılmıştır.

CHAPTER 1. INTRODUCTION

In this chapter, a background relating to this thesis is presented on IoT definition and architecture, the common protocols and standards that are used and how they relate. The section continues with a description and an introduction to the system of IoT device classifications. Furthermore, some background information about IoT privacy and security is further presented. Also, the research questions, purpose of research, research motivation, research limitations and thesis outline are presented in this chapter.

In the next chapter, we will have a look at an overview of the MQTT protocol in relation to this thesis.

1.1. Background

This section presents a theoretical foundation for this research and covers most of the essential concepts that are required for the following chapters. It is divided into three sections. The first section explains definition, concept and idea behind Internet of Things and provides necessary overview of the architecture reference model of IoT. Second part covers theoretical background concerning common protocols and standards available. Lastly, we have a look at privacy and security in IoT relevant to this thesis.

1.1.1. Internet of things definition and architecture

In 1999, the term IoT was first coined by Kevin Ashton during his RFID (radio frequency identification) presentation [1]. It has become a very important research field since that time. IoT in information system plays a major role of bridging the gap between the things we see (i.e. physical world) and its various representations. A more detailed definition of IoT is given as

"A world where physical objects are seamlessly integrated into the information network, and where the physical objects can become active participants in business processes. Services are available to interact with these 'smart objects' over the internet, query their state and any information associated with them taking into account security and privacy issues." [2].

IoT has a fundamental aspect that focuses on collection and utilization of large amounts of data that comes from the various types of sensors placed in various kinds of physical objects. So, in decision making and remote monitoring, it is essential to produce a form of processed, refined or meaningful data from the accumulated raw data for the purpose of great productivity. Hence it is equally important to process (thus to analyze and refine) the collected data as just receiving raw data from such devices.

Constrained devices are limited in resources namely, memory (ROM and RAM), CPU and battery life (if available) [3]. They are often used as sensors that collects data, machine to machine (M2M) or smart devices that control services and electrical appliances. When such devices are connected to a network they form what is called "things" and in a whole, they form part of the "Internet of Things" (IoT).

IoT is a network of objects such as constrained devices, embedded computers, controllable, intelligent and automated devices (smart devices) [4]., and sensors with the capability to connect and exchange data with other devices and services. Each domain has quite a number of different specifications, purpose, challenges and security requirements. IoT solutions are networks of devices and sensors that gather and exchange data transferred over networks and the cloud. It has a number of applications namely home automation [5]., manufacturing, medical and health care systems [6]., environmental monitoring, and transportation.

The internet enables devices to provide information and services as they interact with the physical world to anyone, at any time, to anywhere. Hence, IoT users can be able to have direct access to their device information that are stored on web servers so that they can interact and control their devices through a web, mobile, and other application interfaces.

The fast emergence and innovations of digital things and Information Communication Technology are enabling rapid development and deployment of IoT around the globe. Innovations include Information Communication Technology and IPv6 (Internet protocol version 6). There are estimates that predicts that trillions of IoT devices will be deployed in next five years [7]. IoT applications are growing in number and are utilized to enhance solutions for multitude of diversified problems. Furthermore, a study [8]. points to estimates that places the number of IoT devices to exceed 30 billion with more than 200 billion intermittent connections that can bring forth a revenue of over 700 billion Euros by 2020. In addition, a study by world bank according to [9]. makes a prediction that IoT opportunity can reach staggering 32 trillion dollars or could be 46% of the size of the global economy today. They further predict that IoT chip opportunities could enable the industry to surpass a 400-million-dollar mark by 2020.

According to the research by Professor Howard [10]., the number of IoT devices surpassed the number of humans in the year 2014 (see Figure 1.1.). There are two milestones namely the year the “Internet of Things” was coined as a term. The second milestone is shown to be around 2014, and this is the point at which the number of device to device communication became more than the number of people to people communication. The author’s calculations for prediction was based on data accumulated from ABI Research (2013), Business Insider (2013), Cisco (2013, 2015), EMC (2014), Ericsson (2011), Forbes (2013), Gartner (2013), Hammersmith Group (2010), Intel (2014), Internet Census (2012), Internet World Stats (multiple), Machina Research (2013) and Navigant Research (2013) [10].

The connectivity of devices, systems and services which is IP-based now goes beyond the normal human-to-machine (H2M) and machine-to-machine (M2M) communication which is now termed as the Internet of Things (IoT). Sensors and actuators, are strategically deployed in various areas namely residential (home automation), military, e-textiles, healthcare, industrial systems and automobiles [11].

We are digitally surrounding smart systems namely smart watches, smart homes and smart cars [12]. We are seemingly heading to an era where objects are smart like we have ever thought or imagined and they will be interacting with each other. IoT is a promising area, which means that things are connected to the network through the internet and transmitting data including live events in matters of seconds.

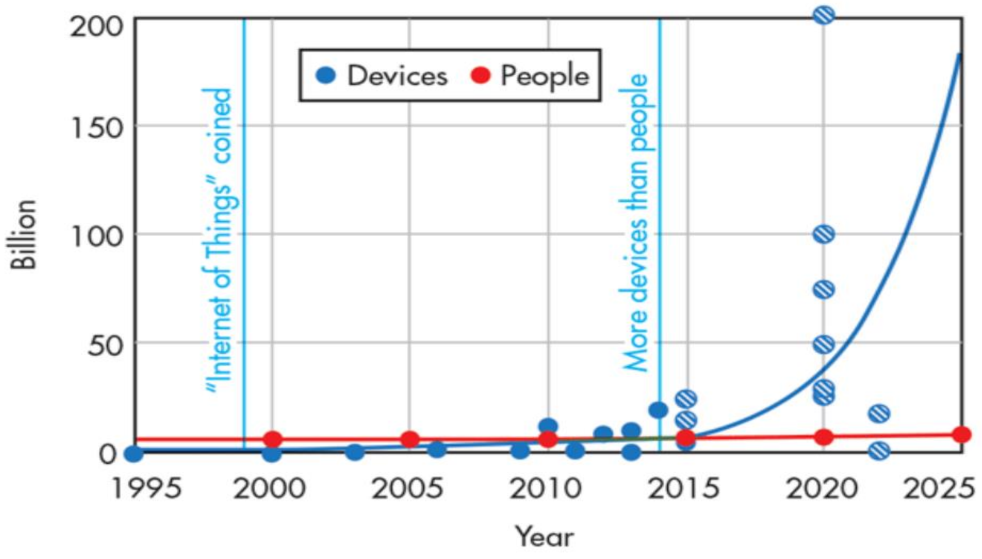


Figure 1.1. Timeline estimates of IoT and world population [10].

IoT are promoted by manufacturers or businesses to better engage the clients or consumers by providing better products or services that is geared towards improving efficiency [12]. One potential area among others, where IoT can be widely applied is health-care facilities. As pointed out by R.A Rahman et al. [12]., a number of researchers have brought up means to attach or embed smart devices on the human body. Examples include wearable devices that are basically used to monitor and maintain human health and wellness, higher productivity, disease management, increased fitness, etc., [12]. The expectation of this is that, IoT might help in predicting and discovering the start of an ailment, disease or health issue in their early stage. Such systems are essential to help hospitals to be more responsive to avoid unfortunate scenarias in the form of casualties. Nevertheless, in spite of the technological advances even in the hospitals of which we are witnessing, security has become a very essential, crucial and criticial area and these are of concern and they have to be tackled [13].

The future of IoT has an idea to grant physical objects a common goal of autonomously working together as a group in problem solving with little human interaction [14]. Also, it is envisioned that most objects will have the ability to learn as well from the refined information collected over a certain period of time from various objects in a network.

The Internet of Things (IoT) as a global industry movement brings people, processes, data, and things together to form networked connections that are more relevant and valuable. Opportunities for countries, industries and individuals will be on the rise in the near future, as the growth and convergence of information, people, and things on the Internet increases. This group known as the IoT World Forum Architecture committee [15] sets out an IoT Reference Model with the purpose to provide a clear definition and description to be made applicable to various elements of IoT and its applications. They simplify by breaking down systems that are complex for better understanding, clarify by providing more information to accurately know the levels of IoT and establishes a common terminology. Furthermore, they also identify specific types of optimized processing in different areas of the IoT system. They standardize to provide a baseline to enable vendors to manufacture IoT products that can interoperate easily and they organize to make IoT realistic, feasible and approachable than just a mere concept [16]. According to studies and researches [15,16,17]., IoT has a general architecture and set of interfaces. These includes physical objects (sensors and actuators), network interface infrastructure (routers, switches and gateways), cloud computing entities and interface for end users. A model explored by IoT World Forum Architecture committee that consists of Cisco, General Electric, IBM, Intel and Oracle [15,16] is shown in Figure 1.2. It is also referred to as the IoT reference model and its levels.

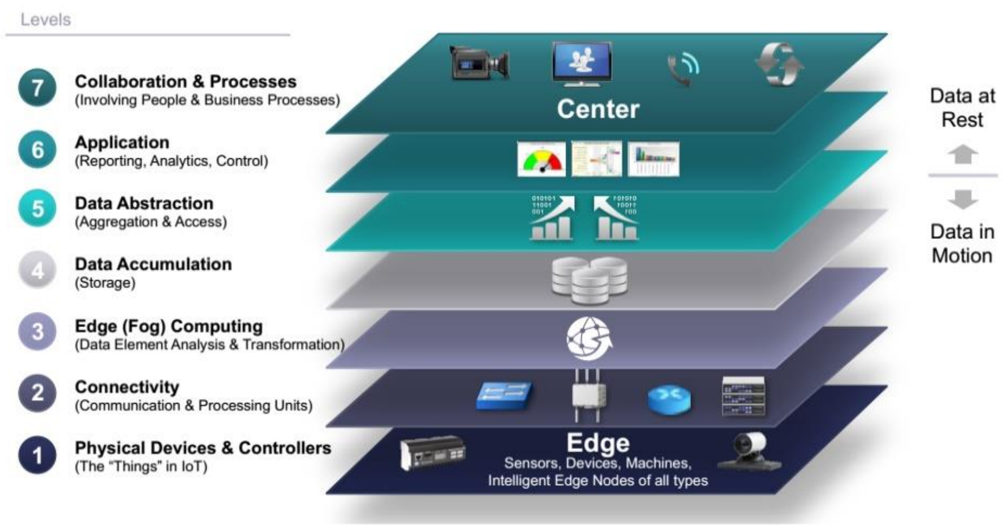

Levels
7 Collaboration & Processes
(Involving People & Business Processes)
6 Application
(Reporting, Analytics, Control)
5 Data Abstraction
(Aggregation & Access)
4 Data Accumulation
(Storage)
3 Edge (Fog) Computing
(Data Element Analysis & Transformation)
2 Connectivity
(Communication & Processing Units)
1 Physical Devices & Controllers
(The "Things" in IoT)
Center
Edge
Sensors, Devices, Machines, Intelligent Edge Nodes of all types
Data at Rest
Data in Motion

Figure 1.2. The IoT reference model according to [16].

Another model [9] is shown in Figure 1.3. It is basically seen to be similar in design and illustrates the same idea.

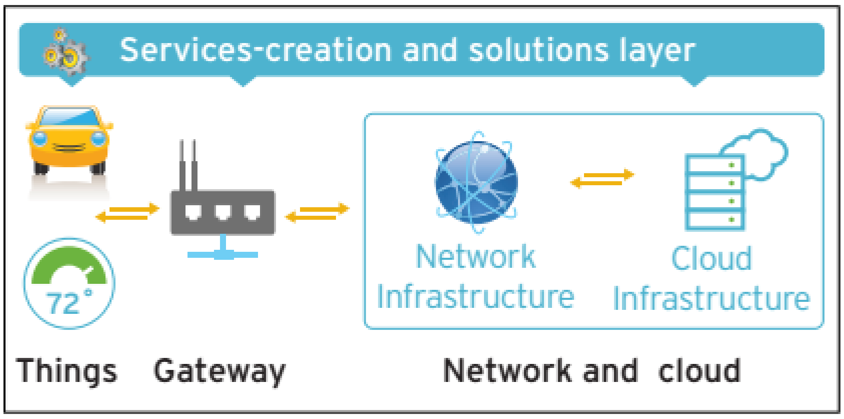


Figure 1.3. IoT Topology according to [9].

The gateway basically provides internet or links to an internet infrastructure and consequently a cloud infrastructure that are mostly comprised of large pools of virtualized servers or storages that are basically networked together to make it easily accessible to end users, applications and services. According to these (Figure 1.2. and 1.3.), we can generally say that an IoT architecture is composed of physical, communication infrastructure, cloud computing infrastructure and applications and services. These form a model where communication infrastructures connect to physical and cloud infrastructures that allows data to flow in both ways (i.e. from top-to-bottom and vice versa).

1.1.2. Internet of things protocols

According to the definition of IoT, we can see that it is a connection of devices via the internet that were not connected previously. Since the internet is used to qualify the connected devices as IoT they must follow the Internet Engineering Task Force (IETF) internet protocol suite (Table 1.1.). The internet is seen to have connections of more powerful devices with high power usage, memory and processing power. Hence, the protocol used is considered too high for most emerging IoT devices [18,19]. Other requirements of IoT drove IETF to implement the suite for IoT. These include, losses at end nodes, long life span, low power, and constrained resources. Hence the new suite needs new, lighter-weight protocols that requires a much lower amount of resources. MQTT is one of the most protocol that addresses these needs. It employs message management, and lightweight message overhead and above all small message sizes.

Table 1.1. IETF Internet suite (taken from http://www.electronicdesign.com/iot/mqtt-and-coap-underlying protocols-iot).

Layer	Full internet	Description
Application	HTTP	Defines TCP/IP application protocol and the interface to transport layer services.
Transport	TCP/UDP	Provides communication session management. Defines the level of service and status of the connection.
Internet	IP	Performs IP routing with source and destination address information.

A classification scheme (see Table 1.2.) by Bormann et al. (2014) [20] was designed to differentiate IoT devices base on resources that are available. This was done to avoid confusion during discussions.

Table 1.2. IoT device classification according to [20].

Name	Ram	Rom/flash
Class 0, (C0)	<<10Kb	<<100Kb
Class 1, (C1)	~ 10 Kb	~ 100 Kb
Class 2, (C2)	~ 50 Kb	~ 250 Kb

According to this classification scheme, our constrained device falls in the classification of Class-0 IoT Device. More details about our device will be in the subsequent chapter. Class-0 devices are seen to be very much resource constrained, hence it cannot support secure communication channels over the internet. On the other hand, Class-1 devices have relatively enough resources to support constrained communication protocols such as CoAP [12,21] and in some cases, it supports transport layer security protocols such as DTLS (Datagram Transport Layer Security) [22]. Class-2 devices have enough resources available to give support to heavier web protocol stacks such as HTTP [13] over TLS [20]. However, there is a need to conserve resources for especially highly constrained devices since that will help to determine the kind of security scheme that can be supported.

HTTP, MQTT and CoAP are some of the most used protocols for communicating to the web. Due to the standardization in data transfer, CoAP and HTTP mostly uses standardized REST (Representational State Transfer) methods. These methods include (GET, POST, PUT, DELETE) [21,23] and media types namely, JSON, ATOM, XML. Hence to minimize resource overheads in most resource constrained devices (IoT), MQTT, Devices Profile for Web (DPWS) [11,24] and CoAP are utilized.

IoT protocols focuses to tackle the issues of security with standardization initiatives to enhance interoperability, efficiency, scalability, and secure communication stacks [1,11,24]. Efforts are being made to standardize protocols that aims to unify IoT devices and applications. Some may be proprietary and openly available as well [11].

K. Fysarakis et. al [11] detail out three main approaches to the protocols used in IoT. They are Service Oriented Approach (SOA) architecture, Resource-Constrained Approach, and the Message-oriented Approach.

Service-oriented approach architecture service is also known as Device Profile for Web Services (DPWS). It was introduced in 2004 by Microsoft and now OASIS (Organization for the Advancement of Structured Information Standards) open standard. It is integrated with various windows OS versions. It further provides secure web service messaging, discovery, description, synchronous and asynchronous interactions on resource-constrained devices. It also enables embedded and sensor devices with constrained resources to leverage the SOA concept and benefits across heterogenous systems in smart environments. Industrial automation [25]., smart homes [26]., smart cities [27]., and e-health [18] are other applicable areas of SOA.

Secondly, another protocol approach is the resource-constrained Approach [11]. This protocol follows the representational state transfer (REST) architecture which is by far popular across the globe currently. They rely on HTTP. REST architecture is inappropriate for IoT due to high usage of resources, bandwidth, and power. Hence the Internet Engineering Task Force (IETF) formed Constrained Restful Environment (CORE) working group that designed the CoAP [12,21] which is now a standard of IETF. CoAP uses simple proxies. It is termed often as the “HTTP for IoT” [11,22]. It is based on the request/respond model using HTTP methods like PUT, GET, POST, DELETE on servers’ resources.

Lastly, the other IoT protocol approach is known as the Message-oriented approach. It uses the asynchronous data transfers between devices. For reliable messaging, QoS are mostly the focus with a centralized controller for message delivery. MQTT is one of such message oriented protocol. It was introduced in 1999 by IBM and standardized by OASIS. It was designed as a publish/subscribe lightweight messaging transport protocol that optimizes high latency or unreliable networks for small sensors and mobile devices. MQTT is used in a variety of domains and researches have been performed in the area of e-health, WSNs, smart grid [22]., and in the area of mobile IoT [28].

CoAP messages are transported over UDP (User Datagram Protocol), MQTT relies on Transmission Control Protocol (TCP), and DPWS can use both (TCP for most of the device interactions, and UDP for device discovery and other auxiliary functions). MQTT stands out when it comes to publish/subscribe interactions. CoAP can support such functionality partially; it possesses synchronous interactions, instead of the event-based ones [11].

DPWS on the other hand, is more flexible. It has a web service (WS)-Eventing specification that enables a functionality that is a feature-rich publish/subscribe. Moreover, QoS remains an important aspect in MQTT, with this protocol supporting three different levels of message delivery (“Fire and forget”, “Delivered at least once” and “Delivered exactly once”). CoAP on the other hand, only brings to board a choice between “Confirmable” and “Non-confirmable” messages. The former has to be acknowledge by the receiver with an ACK packet, normally in applications where it becomes quite necessary to handle UDP’s unreliable transport. DPWS relies solely on the delivery mechanism of TCP. Various extensions enhance the reliability and QoS features of Web Services, but are yet to be integrated into DPWS. More detailed survey on IoT protocols can be found in [21,29,30].

According to studies [31]., MQTT has got much usage and abilities as compared to CoAP. According to another study [5]., MQTT protocol consumes less power than CoAP, hence a much-preferred choice for class-0 IoT devices.

Thangavel et al. [31] illustrate a performance analysis between MQTT and CoAP via a common middleware which shows that the performance of either protocol depends on the network conditions. They also showed that at lower packet loss rate, MQTT tends to have lower delay than CoAP.

According to research [11]., MQTT by far was rated with the best client-side response time (the time taken for a user when trying to reach to a sensitive resource, like sensors for temperature and humidity) as compared to DPWS and CoAP in a benchmark test. Another lab-based comparison of CoAP and MQTT, in the context of communications over cellular networks, can be found in reference [32].

1.1.3. Security in Internet of things devices

There have been great advances in the industry of IoT for different purposes and applications. Each domain has quite a number of different specifications, purpose, challenges and security requirements. For instance, a patient monitoring system is more likely to require a higher data privacy than a smart parking solution [33]. The exposure of more data to more applications makes security a major challenge for IoT developers. About 70% of most ordinary IoT devices lack data encryption as pointed out by J. King et al. [3]. TLS provides security for transferring data over the network. The data is encrypted to prevent anyone from listening and understanding the content. TLS is popularly used to enhance secured access to a wide range of webpages. It uses server certificates that clients must validate and in some cases the server also must validate client-targeted certificates. MQTT uses TCP and by default no encryption mechanism is implemented during communication. Although implementing TLS impacts the performance, communication and the load on the server, most MQTT brokers support the use of TLS. However, an additional security at the application level can be implemented [7].

In this thesis, security is defined as the protection of data from an unauthorized access or interference by ensuring confidentiality, integrity, and authenticity of data. Confidentiality of data is defined as the protection of data from being disclosed to unauthorised persons, parties or systems. Integrity on the other hand is defined as the prevention of modification of data by unauthorised persons. And authenticity refers to the proper verification of a device or system by following a special identification process [34]. This thesis focuses on the network analysis during the basic form of confidentiality (data privacy) technique i.e., payload encryption.

The Hypertext Transfer Protocol (HTTP) is an application layer protocol for distributed and hypermedia information systems. Since 1990, it has been used for most of data communication when internet came to being [35,36]. This protocol is considered insecure as it sends data in plaintext without applying security mechanisms for data protection. With the rise in data that are sensitive and being transmitted over the internet, there was a need for more security. This led to the development of HTTP over Secure Socket Layer (SSL) and then it was succeeded by the Transport Layer Security (TLS). This combination became known as HTTPS, a protocol for secure communication designed to prevent most of the security vulnerabilities such as eavesdropping, tampering, or message forgery [12,17,26]. HTTP and HTTPS uses the transport layer protocol Transmission Control Protocol (TCP) that ensures reliability, error protection and flow control during data transmission. Additional system resources are used during the data checking to ensure reliable communication [37].

However, HTTP and HTTPS were not designed for IoT devices with resource limitation. In an effort to standardise constrained device communication, IETF developed an efficient web standard for constrained devices namely CoAP. Security standards adapted existing TLS security protocol to create a secure IoT. This led to the design of the DTLS protocol which provides a mechanism for securing data communication in some IoT devices. It enhances data confidentiality, integrity, and authenticity of data communication just like the protection provided by TLS on HTTP [35]. However, DTLS is still a heavy weight protocol, hence devices must have sufficient resources to run it while still being able to perform the devices intended functionalities e.g. temperature and humidity sensors collecting data. Some researches [12,38] have studied and shown mechanisms to improve upon DTLS and also to use other means of security that may not demand higher resources.

On the other hand, studies [24,33] have been conducted with MQTT in terms of providing other security mechanism other than TLS/SSL which is a disadvantage for Class-0 IoT devices. MQTT was designed as an extremely lightweight publish/subscribe messaging transport, for small sensors and mobile devices, optimized for high-latency or unreliable networks.

In an MQTT environment, SSL and TLS protocols are the only security available. Hence, it is the job of users to provide other means of security. Furthermore, the TLS/SSL protocol is not sufficient for optimal security at MQTT. This security does not cover the broker's level. Thus, a user in the broker's access is authorized to access all information. After his connection to the broker, the user is in listening to a Topic and receives all the information. The use of symmetric algorithms such as AES or DES (Data Encryption Standard) [39]., and asymmetric algorithms such as RSA may solve this issue by means of the encryption/decryption of messages. The best distribution of secret keys to all users is thus essentially required.

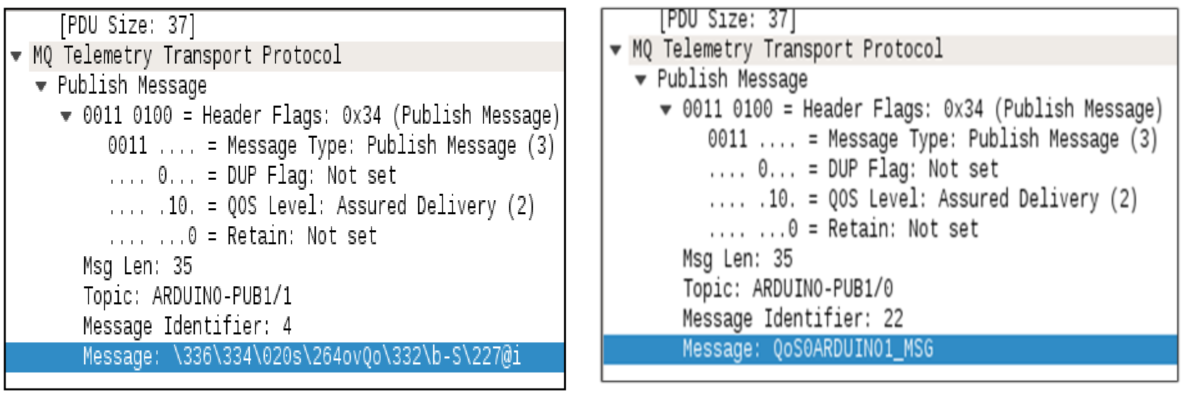
The MQTT publish scenario can undergo data encryption and the subscription scenario can undergo data decryption at the application level. This implementation is particularly important for untrusted environments or any insecure network connections among devices and MQTT broker. Some advantages and disadvantages of payload encryption are provided in the Table 1.3.

Table 1.3. Advantages and disadvantages of payload encryption.

Advantages	Disadvantages
Provides a complete end-to-end message security.	Encryption and decryption may use much resources.
Adds a layer of security for applications that are transmitting highly sensitive and confidential data.	Man-in-the-middle and replay attacks are not solely prevented.
It thrives in situations where TLS cannot be implemented.	There may be the need to implement secure keys exchange for clients.

The message fields of MQTT-publish metadata are not altered after payload encryption except the payload information which is binary-based. This is encrypted and also there is no need for special encoding such as the base 64-encoding in most HTTP request-based messaging [3] while it transmits the message (see Figure 1.4.). This is important to save additional bandwidth since text encodings are typically greater in size than raw byte representations. Likewise, the broker requirements are also met as well even after encryption. The application that will interpret the message needs to be decrypted and this will occur at the subscriber client. MQTT over TLS hinders performance by increasing CPU usage (see Figure 1.5.). While this performance cost is not significant for most brokers, it poses a challenge for devices with limited resources.

Encryption can basically be termed as a process in which plaintext message is transformed into a cipher text using an encryption algorithm and a secret key. Decryption on the other hand is a process in which a cipher text is transformed into a plaintext using a decryption algorithm and a secret key. Plaintext refers to the original message and cipher text refers to the coded message. So, in symmetric cryptography, the system of encryption and decryption algorithms use the same key. Asymmetric cryptography refers to cryptographic system where encryption and decryption algorithms use different keys.



```
    [PDU Size: 37]
▼ MQ Telemetry Transport Protocol
  ▼ Publish Message
    ▼ 0011 0100 = Header Flags: 0x34 (Publish Message)
        0011 .... = Message Type: Publish Message (3)
        .... 0... = DUP Flag: Not set
        .... .10. = QOS Level: Assured Delivery (2)
        .... ...0 = Retain: Not set
      Msg Len: 35
      Topic: ARDUINO-PUB1/1
      Message Identifier: 4
      Message: \336\334\020s\264ovQo\332\b-S\227@i
```

```
    [PDU Size: 37]
▼ MQ Telemetry Transport Protocol
  ▼ Publish Message
    ▼ 0011 0100 = Header Flags: 0x34 (Publish Message)
        0011 .... = Message Type: Publish Message (3)
        .... 0... = DUP Flag: Not set
        .... .10. = QOS Level: Assured Delivery (2)
        .... ...0 = Retain: Not set
      Msg Len: 35
      Topic: ARDUINO-PUB1/0
      Message Identifier: 22
      Message: QoS0ARDUINO1_MSG
```

a) b)

Figure 1.4. MQTT header metadata as seen from Wireshark application console: a) Encrypted Message, b) Non-encrypted message.

Symmetric encryption is therefore referred to as a cryptographic approach that employs the possibility of encryption and decryption of a message with the same key. This works very well for a trusted network. Symmetric encryption is much easier in implementation than asymmetric. The U.S. National Bureau of Standards created an encryption standard that is complicated. It is called DES (Data Encryption Standard) and it is used to encrypt data by offering unlimited ways to do that. This was later replaced by Rijndael encryption [39]. Rijndael also known as AES alogorithm uses a key for encryption that has a size of 128, 192 or 256 bits. It provides high protection against brute force attacks and it is three times faster in software than the Data Encryption Standard (DES).

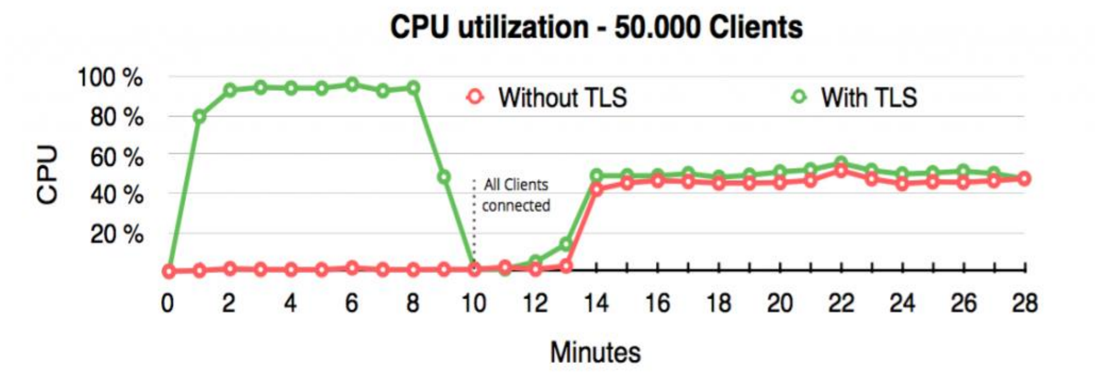


Figure 1.5. Comparison of CPU usage for plain TCP versus TLS (taken from http://www.hivemq.com/tls-benchmarks).

This method can be used for securely exchanging keys as well as transferring data with a size of 128, 192, or 256 bits. More notably, AES-256 bits encryption algorithm is certified in the USA for government documents that are marked as top secret [39]. In our experiment, we employ the use of AES-128 bits Cipher Block Chaining (CBC) with a 128 bits initialization vector (iv) and a 128 bits key since the payload sizes will be in levels of 16 bytes (128 bits).

Symmetric encryption, can also be referred to as a conventional encryption or single-key encryption. It was the only type of encryption before the invention of asymmetric cryptography in 1970 [40]. Symmetric encryption can be classified into two operational categories: block ciphers and stream ciphers. Block ciphers take a specific block of plaintext as input and produce same length of block of encrypted data as their output. While stream ciphers use stream of bits or bytes as input and produces corresponding stream of encrypted bits or bytes as output [40]. Implementations can be found from these two categories: AES and DES (Data Encryption Standard) [40].

On the other hand, asymmetric encryption, also known as public-key cryptography, uses a key pair instead of single shared key in symmetric encryption. In theory, it is different from symmetric cryptography since it is solely based on mathematical functions, specifically one-way functions, rather than on various substitution and permutation schemes. It uses one key for encryption and another related key for decryption. These keys form a pair: public key and private key. The private key is kept secret (private) and it is not distributed to others, unlike the public key. All plaintext messages are encrypted with the public key and can be decrypted using only the corresponding private key. Various implementations rely on asymmetric cryptography. These includes RSA (Rivest-Shamir-Adleman) and Elliptic Curve [41].

The Transport Layer Security (TLS) works by using cryptography to ensure a secure and a reliable connection for data communication channel. Basically, the public key of the receiver is used to encrypt the data by the sender. Thus, data is then sent across the internet to the rightful recipient. Decryption is only performed by only the recipient's private key and this key is held private and secured by the recipient. In terms of operation, asymmetric cryptography methods clearly require more resources than symmetric cryptography since security handshake and also key exchange must take place. Class-0 devices is very limited in resources, hence asymmetric cryptography and DTLS protocols are too 'resource-heavy' for them.

Asymmetric cryptography is too resource intensive to secure communication in Class-0 devices. Hence symmetric encryption offers an alternative solution with minimal resource demand. Symmetric cryptography uses a single encryption key which is mostly shared between a number of devices. These devices which possess the key can decrypt data sent from other devices with same key. To prevent the key from falling into the wrong hands the key must always be kept safe. One of these symmetric encryption is the AES and it functions at fast speeds and requires less resources than DTLS hence very appropriate for Class-0 constrained devices [38]. AES inputs are 16-byte (128-bit) blocks which are then encrypted using a key of 128-bit, 192-bit, and 256-bit in size [37,39]. The larger the key size the greater the security and resource requirement. In the communication layers symmetric encryption can be applied at different layers such as the data link layer and to specific data objects of message such as sensor readings.

A study [24] presents a comprehensive evaluation of different security mechanisms that are based on MQTT using different AES encryption mechanisms on different payload sizes. Furthermore, another study [44] proposes an interesting approach for high end devices that uses a hybrid encryption (AES and RSA), i.e., symmetric and asymmetric encryption.

1.2. Research Statement And Question

Constrained IoT devices especially Class-0 IoT devices possess very limited amount of resources hence are limited to some protocols and standards that they can support. According to a study's introduction by J. King et al. [3]., about 70% of ordinary IoT devices lack data encryption. As there is a rise in information, M2M and D2D communications, security vulnerabilities also tend to increase. MQTT is a lightweight communication protocol that operates on different levels of QoS for reliable communication. However, it lacks lightweight security mechanisms. Lightweight mechanisms such as symmetric cryptography is being employed at the application layer. Theoretically, this encryption adds additional resource load on the device but it is worth the privacy service it provides. Solutions to secure data exist, however most rely on TLS mechanism in which Class-0 IoT devices lack the necessary resource support. We therefore ask further questions which eventually forms the basis of this thesis research.

Q1: Can encrypted payload based on AES-128 bits affect network performance or characteristic?

Q2: Will it have the similar effect as compared to an unencrypted payload/plaintext for different MQTT QoS levels and different payload sizes?

1.3. Related Research

MQTT Publish/subscribe is steadily increasing and becoming a very essential communication protocol for sensor devices and Internet of Things due to its communication of messages with reliability and efficiency and further more consumes less power for devices that are resource constrained. There have not been much efforts in analyzing payloads (encrypted and non-encrypted) in MQTT and their effect on network. Several works [7,24,33,42,44] have made use of MQTT in studies and research especially in improving privacy (securing the payload). Application of MQTT have been studied [7,11,23,24,29,32,33,42,44] and it has showed to be a major protocol to be reckoned with for next generation or emerging IoT devices. Furthermore, data object encryption at the client side has been a prominent study of interest to quite a number of researches [7,24,33,42,44] and they applied the AES mechanism in most cases.

A study also on the other hand [42]., performs encryption on the payload using AES and ABE (Attribute base encryption) on the secret AES key to ensure that the ciphertext is same as the original message. This is resource expensive and a detailed analysis will definitely show that it is going to be a computational overhead for resource constrained devices since they generate a few amounts of data for encryption as well. In as much as encryption is required in such devices, they are expected to give a well optimized operational result. Our work seeks to present the real-time effect of one of the main encryption algorithm (AES) on some network parameters as compared to when it is not encrypted (plaintext).

Quite a number of researches and studies have been presented on MQTT that have applicable features for the IoT industry namely automotive, railway, health, smart home and cities as discussed earlier in the previous chapter. Also, a comprehensive research [24]., illustrates the use of MQTT in evaluating a series of security mechanisms that can be used for this protocol. The study analyzes the network characteristics of various security mechanisms including link layer security (LLSec) using AES-CCM, application layer or payload encryption using AES-128 BITS, AES-CBC and AES-OCB) on an actual wind park as an illustration of an industrial network. They use an ultra-low IoT device (Zolertia Z1) which is a Class-0 IoT device. They run network traces to compute the evaluation performance amongst these security mechanisms. This is a very good research that provides a motivation for this thesis to seek similar research into the effect of AES-128 BITS payload encryption on the network in relatively the basic of network connections. They used MQTT and as such have the pleasure of the QoS levels.

A study [43]., related to this thesis analyzed different payload sizes using high-end devices and MQTT communication protocol based on different QoS levels to establish a correlation analysis between parameters: message loss and end-to-end delay over wireless and real-wired network via the internet. It deduced that they are correlated. However, a comprehensive analysis is also needed in situations when the payload is encrypted and the devices are low-level/resource constrained or falls in the range of a Class-0 IoT device. This thesis seeks to show also the effects on network alongside each QoS levels.

1.4. Purpose Of Research

The main objective of this thesis is to present a method of analysis of lightweight encrypted and non-encrypted payload based on MQTT communication protocol and its QoS levels. The analysis is focused on getting results that seeks to establish how the network communication is affected when communication is performed using encrypted payload at the client side and also how they relate with the non-encrypted payloads. Also through the literature, results and discussions, the reader will come to an understanding of some technologies, current issues and may find future research areas related to this thesis.

1.5. Research Motivation

The IoT industry is expected to surge in coming years to bring lots of revenue. Also, security is a main issue and the need to present or contribute to researches related to IoT security is essential to a positive growth to IoT in the near future. Much work and studies have been conducted into securing data transmissions from constrained devices. However, there is a minimum resource requirement needed to support most of these security mechanisms especially for Class-0 devices. Hence, they are limited to just some lightweight yet powerful security mechanisms such as AES. We look at its encryption effect on network performance by analyzing encrypted and non-encrypted payload sizes based on MQTT and its QoS levels.

1.6. Research Limitations And Thesis Outline

The scope of this research will be focused around the effect of the end-to-end client encryption of payload on network performance/characteristics as compared to non-encrypted payload based on MQTT. We analyse in the range of a local network set-up with no other secured communication channels implementations. This research does not focus of the use and storage of data but the sending of encrypted message to the broker server and the decryption of the message by the subscriber client's end. We focus on knowing what happens to some network parameters namely, latency, message loss and throughput when data is encrypted and sent from the publisher client and received for decryption at the subscriber client's end.

The thesis is organized as follows. Chapter 2 presents an overview of the MQTT protocol for this work. It covers the main features and relevant control packets of MQTT. Chapter 3 covers the method of research, what and how it was performed. It presents the experimental set-up, design and the analytical method applied. Chapter 4 presents the experimental results and discussion. It covers both encrypted and non-encrypted payload analysis results, and a correlation analysis. Chapter 5 briefly covers the conclusion and future research.

CHAPTER 2. MQTT PROTOCOL OVERVIEW

The objective of this overview is to have a fundamental background knowledge of the MQTT protocol used for this research. The overview begins with an introduction to the protocol and follows with the main features and it ends with some aspects of the technicalities involve in the packets sent using this protocol.

The intended purpose and functionality of IoT device depends on the amount of available resources. A Class-0 device resource must be below a certain resource threshold as stated by Bormann et al. [20] with less than 100Kb ROM and/or less than 10Kb RAM. An example of a Class-0 device is the Arduino Uno [45]., an 8-bit microcontroller with 16MHz CPU, 32Kb RAM, 2Kb ROM. MQTT as a lightweight protocol and its ability to be implemented on such device makes it a suitable protocol for research purposes.

MQTT (Message Queuing Telemetry Transport) is an application level protocol which functions and relies on top of the TCP/IP stack. It is simple, lightweight and easy to implement protocol which is based on a client-server publish-subscribe messaging pattern. It is suitable for M2M (Machine to Machine) or IoT where a low resource requirement is expected/or network bandwidth is at a very low.

Originally developed at IBM in 1999, MQTT was designed to be lightweight, bandwidth efficient, simple to implement, agnostic about delivered data, aware of the session and able to provide QoS (Quality of Service) for delivered data. MQTT was used initially used at IBM for proprietary embedded systems. The turn-around came in 2010 when IBM decided to release the protocol free for everyone to use [46]. The protocol was placed under OASIS and in 2014 it was released as a standard under open OASIS standard with a version 3.1.1 from the previous version of 3.1. As at the time of writing this thesis, version 3.1.1 is the latest version of the protocol [47]. MQTT system is based on publish-subscribe pattern that relies on a central node, called a message broker (server). All communicating end points (clients) are connected to the broker. Thus, a client’s messages are sent to and received from a broker. It is the task of the broker to receive messages from clients and to send them to the recipients rightfully in need of them. In MQTT, when a client sends a message, the message is assigned to some topic. Each client gives an indication of interested topics to the broker. It could be one or more topics. According to this, the broker can apportion the right message according to the topics that is received and deliver it to the rightful recipient. The term publish is used when a message that is assigned to a specific topic is sent by the client to the broker. While the term subscribe is used to describe the moment a client registers an interest in a topic and its subsequently the corresponding messages to the broker [48].

2.1. MQTT Main Features

The protocol is may be considered simple however there are some features which needs a proper understanding and if possible can lead unto further research studies. The strength and some main features of MQTT related to this thesis are covered in subsequent sub-topics.

2.1.1. Connection

During a client’s establishment of connection to the broker, a CONNECT packet to the broker is sent. With the CONNECT packet, the client configures set of parameters that are used for the connection with the broker. These parameters control e.g. what happens if client disconnects from broker, or whether some messages should be stored if it goes offline. Below are some of the parameters.

a. Client identifier

Client identifier uniquely identifies the client for the broker. The first UTF-encoded string. The Client Identifier (Client ID) is between 1 and 23 UTF-8 encoded bytes in length (characters long) [47]. It must be unique across all clients that are connecting to a broker server and is the key in handling Message IDs messages with QoS levels 1 and 2. If the Client ID is more than 23 characters, the server responds to the CONNECT message with a CONNACK return code 2: Identifier Rejected [47].

b. User name and password

User name and password are specifically used to control or check authentication and authorization to broker. They are transmitted in plaintext. A connecting client can specify a user name and a password, and setting the flag bits signifies that a User Name, and optionally a password, are included in the payload of a CONNECT message. If the flag for the User Name is set, that field is now mandatory, otherwise its value becomes disregarded. If the flag for Password is set, that field is now mandatory, otherwise its value becomes disregarded. It is invalid to provide a password without provide its corresponding username [46].

c. Clean session

This is a flag client that indicates whether an establishment of a clean or a persistent session with the broker is needed. Flag is set to true if a clean session is requested/ This means that, the broker will neither restore nor start storing any state for the client and it will purge all information from previous persistent session. However, persistent session (flag = false), previous session (if any) for the client will be restored. This means that any topic subscriptions made by client in previous session are restored and the messages which the client had subscribed with QoS 1 or 2 and also which were received when the client was offline are transmitted to it. If persistent session is requested, broker starts storing state for the current session [46].

d. Will message

This is termed as Last will. It is a part of a feature known as Last will and testament of the MQTT protocol. It is used to notify other clients if a client disconnects from the broker. In case of such event, the broker, on behalf of disconnected client, sends predefined message to predefined topic. Both the message and the topic are defined by the disconnected client during the connection establishment. [49]

e. Keep alive

Keep alive is used as a maximum time interval which is allowed to elapse between consecutive messages sent from client to broker. In the situation when a client does not receive a PINGRESP (ping response) message from the broker within a Keep Alive time frame after sending a PINGREQ (ping request), the TCP/IP socket connection will be closed. The Keep Alive timer is a 16-bit value which represents the time period in as number of seconds. The actual value is specific according to application, but normally a typical value is a few minutes. However, the maximum value is about 18 hours. A value of zero (0) means the client is not disconnected [46,49].

f. Topic name

The topic name is present in the variable header of an MQTT PUBLISH message. The topic name is the key that identifies the information channel to which payload data is published. Subscribers use the key to identify the information channels on which they want to receive published information. The topic name is a UTF-encoded string. Topic name has an upper length limit of 32,767 characters [46].

2.1.2. Topics and messages

The clients in MQTT do not literally communicate directly with each other. All the messages are filtered or pass through the broker server. Every MQTT message has a topic and every client can subscribe to a variety of topics available. Topics are notably organized in a hierarchical form (called topic levels) [50]. It follows after the form of a file path like a computer's file system; e.g. “home/sittingroom/light/status”. The broker receives published messages from a client and is then it is its responsibility to send or push them to any client that is rightfully subscribed to this topic.

A PUBLISH packet is sent to the broker when an MQTT client publishes application data. This packet is made up of an actual application data and topic but also other important information such as retained flag, duplicate flag, message type, and QoS level. [46]

The broker checks the topic and delivers the message to clients that are subscribed to that topic. Messages that dispatched from broker to a subscribed client are also sent as PUBLISH packets. However, the packets are not exact copies of the received ones from the published client, but they have same content in their payload portion (the portion that holds the actual application data). For instance, the QoS level can change during message delivery which affects one of the fields in the PUBLISH packet.

During topic subscription, the client sends a SUBSCRIBE packet to the broker. Such packet basically contains a list of topics with QoS levels that client wants to subscribe corresponding topic. As mentioned earlier, topics have hierarchical levels where each level is separated from each other using a forward slash (/) as shown earlier.

Furthermore, wildcards can be used to subscribe to topics one at a time that may represent multiple topics as well. Wildcards in MQTT are single and multilevel with (#). Example of single level wildcard on a topic description is “stage1/+/stage3”. This means that a subscription with the topic “stage1/A/stage3” or “stage1/B/stage3” is accurate. On the other hand, a multilevel wildcard defined as “stage1/stage2/#” can have an accurate subscription in the form stage1/stage2/C or stage1/stage2/C/D or stage1/stage2/D/C. Thus, stage1 and stage2 should be left intact [50].

The application data embedded in payload portion of PUBLISH packet can be binary representation, XML (Extensible Markup Language), JSON (JavaScript Object Notation) or CSV (Comma-Separated Values). This makes MQTT a data agnostic [46,47] protocol. Handling the payload is sole responsibility of clients. The broker just delivers messages as it is between clients. This implies that the payload portion can be encrypted so that the broker or any unwanted client cannot view its contents without knowing any provided secret key.

2.1.3. Quality of service

MQTT is designed with three message delivery semantics to ensure communication reliability and this is known as QoS levels. QoS is basically creates an agreement between sender client and a receiver client in which they settle on the assurance of message delivery from client to broker and broker to server. QoS level is independent for each message during client publishing and independent for each topic during client subscription processes. Hence, QoS level can therefore get changed (go up or down in level). For instance, a client can publish with a higher QoS than another client that will subscribe same topic with a lower QoS level.

QoS levels in MQTT are: 0 (at most once), 1 (at least once) and 2 (exactly once) [46,47,50]. With QoS level 0; the simplest of the levels, the sender sends PUBLISH packet without any form of waiting in terms of confirmation or acknowledgement from the receiver. In this situation, PUBLISH packet is received at most once. It is represented with a flow below as:

Client to Server: PUBLISH
Server Action: Publish message to subscribers

QoS level 0 does not check message arrival to its destination. This level is mostly used for sensor data where message loss can be considered. QoS 1 (at least once) is where messages are readily assured to arrive but duplicates can occur. Every PUBLISH message is required to be acknowledged by the receiver with PUBACK packet. If the acknowledgement is not received, PUBLISH packet is sent again which might cause data duplication. The flow [8, 29] is shown below:

Client to Server: PUBLISH
Client Action: Store Message
Server Actions: Store Message,
  Publish message to subscribers,
  Delete Message
Server to Client: PUBACK
Client Action: Discard Message

In QoS 2 (exactly once), messages are assured to arrive exactly once. Four main packets are utilized in QoS level 2. The first packet is the PUBLISH packet. To avoid duplication the packet ID is stored by the receiver. It is then acknowledged with a PUREC (publish received) packet. The sender client can discard the initial data after PUREC. When PUBREC is received, sender stores reference to this packet and responds with PUBREL packet (contains also original packet identifier). The packet receiver can now discard every state of packet identifier after receiving PUBREL and responds with PUBCOMP. PUBCOMP is the final packet which ends QoS level 2 packet delivery. If the packet that is assumed to be received is not received in a suitable time window in any of the above delivery stages, previous packet is directly sent again always. Now, in case the subscriber client subscribes with QoS level 1 or 2 and persistent session is true, the broker will store the packets for the client until it confirms them received. This is similar to the case when the client is offline. Generally, the broker will store and queue packets until client reconnects with persistent session and confirms packets received. If client reconnects with clean session all previously stored packets are discarded. Packets that are subscribed with QoS level 0 are not stored at all.

Client to Server: PUBLISH
Client Action: Store Message
Server Actions: Store Message OR Store Message ID,
  Publish message to subscribers
Server to Client: PUBREC
Client to Server: PUBREL
Server Actions: Publish message to subscribers,
Delete Message OR Delete Message ID
Server to Client: PUBCOMP
Client Action: Discard Message

Figure 2.1. illustrates the QoS levels and the packet flow using a diagram.

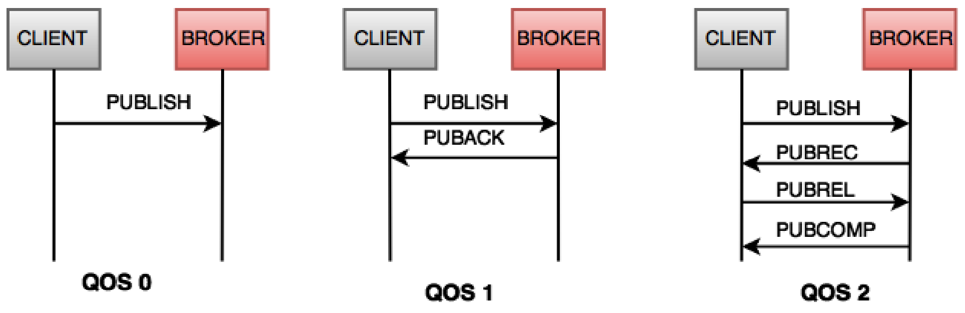


Figure 2.1. MQTT Quality of Service packet transmission.

2.1.4. Security

MQTT is based on an unencrypted TCP which is totally not secure. However, because it sits on top of TCP, it can use TLS/SSL Internet security [51]. TLS is a very secure method for encrypting the communication channel [51]., however it is also resource-intensive for lightweight or constrained clients' devices. This is due to the required handshake and the increased network packet overhead. Hence for networks where energy is so important and security is less, encrypting just the packet payload may be the best solution.

Using client identifier, username and password MQTT can provide mainly authentication and authorization. As such, protocol does not explicitly specify e.g. how application data should be encrypted or its integrity checked when carried within PUBLISH packets. Although MQTT protocol might not have diverse features regarding security, there are various ways to incorporate them on application level: applications can be specified to data format that can make it easy to implement an encryption mechanism and data integrity check in various formats on PUBLISH packet's payload.

CONNECT packet sends Client identifier, username and password broker during an established connection. The broker authenticates client and authorizes what topics it can have access. The broker undergoes a configuration before this is done. Another issue is that the CONNECT packet sends client identifier, username and password in plaintext [24] and is therefore visible to any intermediate network equipment if transferred on top of plaintext TCP connection.

When TLS is used as underlying protocol all MQTT packets can be encrypted and their integrity checked. As a contrast to plaintext username and password, certificates provide better method to authenticate clients. However, certificates must be generated and private keys which poses a major challenge to Class-0 device. TCP port 8883 [47] is used on broker side, if MQTT is used on top of TLS. It is standardized for secure

MQTT connections. While for plaintext, TCP port 1883 is used [33,47]. TLS provides set of good features but it is complex and utilizes a high level of resources.

In some situations, TLS cannot be used. However, with MQTT it is possible to implement security features also on application layer. Security features in the application layer are implemented on the PUBLISH packet’s payload (the actual data) [24]. Encrypted payload remains encrypted from the source to destination (end-to-end (E2E) encryption) [24,30]. Only designated clients with the right key can recover the actual contents. Privacy and confidentiality is witnessed as data transfer and also authentication can be implemented since only clients with the right key has access to the real data. However, a malicious or compromised broker has the capability to manipulate the payload’s integrity. A way to prevent this is to use the Message authentication code (MAC) by calculating it from the payload and added to it before encryption. This will ensure that to be able to modify any part of the payload, any compromised node would need the secret key before it can modify. This primitive mechanism of encryption is very useful at situations where TLS cannot be used for some reason.

2.1.5. Space decoupling

With this process the node will have the broker’s IP address and the broker can also identify the node. Nodes have the capability to publish information and also subscribe to other nodes’ published information. They do not have to have each other’s IP or any knowledge of each other at all since everything goes through the central broker. This tends to reduce network overhead that can accompany TCP sessions and ports. Hence it ensures that the end nodes do operate independently of one another [52].

2.1.6. Time decoupling

A node can publish its information independent of the state of other nodes. As other nodes remain active they can receive published data that they have subscribed unto from the broker. Nodes can remain in sleepy states even when other nodes are publishing messages directly relevant to them since everything passes through the central broker [52].

2.1.7. Synchronization decoupling

In a scenario where a node is in an operation, it cannot be interrupted by a message it needs to receive that it has obviously subscribed to. This message is queued by the broker and makes the broker makes sure that the node finishes its initial operation. This in turn saves operating current and reduces repeated operations by avoiding interruptions of on-going operations or sleepy states [52].

2.2. MQTT Control Packets

MQTT standard defines fourteen different control packet types (see Table 2.2.) [46]. The enumeration is the packet protocol level used to identify those control packets.

a. CONNECT, CONNACK and DISCONNECT are for the establishment of connections and termination of the connection with the broker.
b. PUBLISH, PUBACK, PUBREC, PUBREL and PUBCOMP are used during the publishing of application data to broker.
c. SUBSCRIBE, SUBACK, UNSUBSCRIBE and UNSUBACK are used when subscriptions are made or canceled.
d. PINGREQ and PINGRESP are used to verify that client and broker are alive and reachable.

The message header or message format for each MQTT command message may contain fixed header, variable header or payload. Variable header and payload depends on the packet but the fixed header is always available. Fixed header is composed of a set of fields that are fixed while the fields in the variable header and payload may vary between packets. Below is the table (Table 2.1.) illustrating the fixed header format which is related to our study.

Table 2.1. Fixed header format.

Bit	7	6	5	4	3	2	1	0
byte 1	Message Type				Dup flag	QoS level		RETAIN
byte 2	REMAINING LENGTH							

Text fields in MQTT packets have their encoding as UTF-8 (Universal Character Set Transformation Format) strings and integer values are represented using 16 bits and big-endian byte order. The fixed header has three fields namely message type, flags and remaining length. [46,47].

Byte 1 Contains the message type and flags (DUP, QoS level, and RETAIN) fields.
Byte 2 (At least one byte) contains the remaining length field. All the data values are in big-endian order i.e. higher order bytes precede lower order bytes. A 16-bit word is represented on the wire as Most Significant Byte (MSB), followed by the Least Significant Byte (LSB).

The Message Type Position: (byte 1, bits 7-4) is represented as a 4-bit unsigned value which takes the enumeration values from either of the fourteen control packet types. Enumeration 0 and 15 are reserved. Table 2.2. shows the enumeration for the protocol control packet types and their description. [46,47]

Table 2.2. Enumeration of the control packet types of MQTT

Mnemonic	Enumeration	Description
Reserved	0	Reserved
CONNECT	1	Client request to connect to Server
CONNACK	2	Connect Acknowledgment
PUBLISH	3	Publish message
PUBACK	4	Publish Acknowledgment
PUBREC	5	Publish Received (assured delivery part 1)
PUBREL	6	Publish Release (assured delivery part 2)
PUBCOMP	7	Publish Complete (assured delivery part 3)
SUBSCRIBE	8	Client Subscribe request
SUBACK	9	Subscribe Acknowledgment
UNSUBSCRIBE	10	Client Unsubscribe request
UNSUBACK	11	Unsubscribe Acknowledgment
PINGREQ	12	PING Request

PINGRESP	13	PING Response
DISCONNECT	14	Client is Disconnecting
Reserved	15	Reserved

The Flags are comprised of the DUP, QoS and RETAIN as shown in the table below

Table 2.3. MQTT fixed header flags.

BIT POSITION	NAME	DESCRIPTION
3	DUP	Duplicate delivery
2-1	QoS	Quality of Service
0	RETAIN	RETAIN flag

DUP Position (byte 1, bit 3) is set when there is an attempt to re-deliver a PUBLISH, PUBREL, SUBSCRIBE or UNSUBSCRIBE message by the client or server. This applies to messages whereby the QoS level value is greater than zero (0), and an acknowledgment is required. The recipient should treat this flag as a hint as to whether the message may have been previously received or has been duplicated. The variable header includes a Message ID when the DUP bit is set. NB: It should not be relied on to detect duplicates. [46,47]

QoS Position (byte 1, bits 2-1) indicates the level of assurance for delivery of a PUBLISH message (see Table 2.4.).

Table 2.4. QoS levels.

QoS value	bit 2	bit 1	Description		
0	0	0	At most once	Fire and Forget	< =1
1	0	1	At least once	Acknowledged delivery	> = 1
2	1	0	Exactly once	Assured delivery	= 1
3	1	1	Reserved		

RETAIN Position (byte 1, bit 0) is only used on PUBLISH messages sent by a client to a server and when the Retain flag is set to 1, the server holds on to the message after it has been delivered to the subscribers that are currently connected . Also, the last retained message on that topic has to be sent to the subscriber that has the Retain flag set in the situation when a new subscription is established on that same topic. Nothing is sent when there is no retained message.

Remaining Length Position (byte 2) represents the number of bytes that is remained within the current message. This includes data in the variable header and the payload. The variable length encoding scheme uses a single byte for messages (127 bytes long). 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" [46,47].

The variable header and payload parts vary between packets. For the sake of our study focus we will not cover this area but more of it can be studied from [46,47]. Relevant packets concerning this thesis work are: CONNECT, CONNACK, PUBLISH, PUBACK, PUBREC, PUBREL, PUBCOMP, SUBSCRIBE and SUBACK. These form the minimal set of packets which are required when client needs to establish and/or terminate connection with broker and also send an application data to the broker.

CHAPTER 3. LIGHT WEIGHT PAYLOAD ANALYSIS

In this section, we present the method of payload analysis using a real-wired local network based on packet loss, latency, throughput, different QoS levels and payload sizes for encrypted and non-encrypted payload. The process of capturing packets was performed in real time as two devices communicated.

Our main requirements for the proposed method is to capture network packets for both encrypted and non-encrypted payload at different payloads and QoS levels as they are communicated between a publisher client-to-server-to-subscriber client for analysis. Our expectations were that, there is similar or close effect on network performance for both encrypted and non-encrypted payloads and that they have similar or close correlation coefficients on throughput, and end-to-end latency via MQTT at different QoS levels.

Although the encryption process might add a bit of latency to the processing time, it is worth the confidentiality service it provides for the payload. Also, it will add additional layer of security for devices that can handle TLS/SSL because TLS/SSL is not sufficient for optimal security with MQTT [41]. However, our main objective is to observe the effect of encrypted payload based on MQTT at different QoS levels. The output of this research will serve as a guide or stepping stone onto more studies about MQTT and Class-0 IoT devices in the near future.

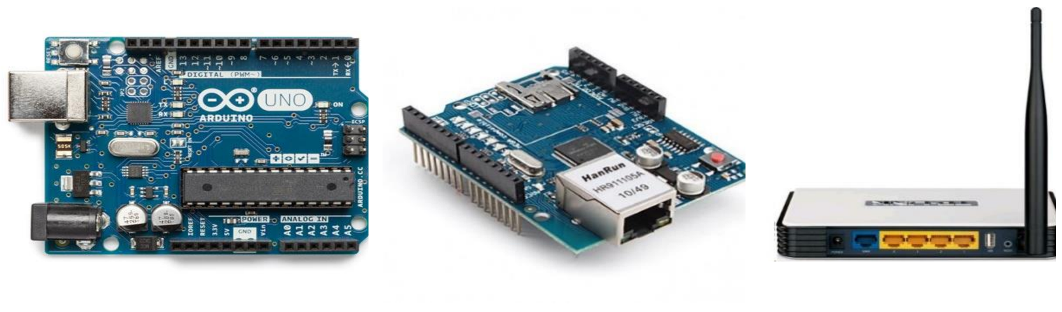
In this thesis we follow the approach of research methodolgy. We have put forward the objective and motivation for the research and our design and implementation of the research is in the subsequent sub sections.

3.1. Experimental Set-Up

For the experimental set-up, we used a 64-bit Linux Kali OS 2017.1 [53] as the server. Kali is an open source and a Linux distribution that has most of the network and security analysis tools already pre-installed. There are other linux distributions that are equally as good as Kali namely Ubuntu, Fedora, and Cent OS. We chose kali because it is renowed to posses good qualities for most security and network analysis support. Moreover, a number of studies and researches [3,5,7,11,33,43,54,55] also depended on linux distributions as the operating system for their server due to the robustness of this operating system.

We also used an open source MQTT broker known as Mosquitto [56]. It is open-sourced and supports the latest standard version of MQTT. It is simple, non-proprietary and easy to use for simple publish/subscribe implementation with C and C++ libraries [56]. Its primary goals are: to avoid polling of sensors, allowing data to be sent to interested parties the moment it is ready and lightweight, so that it can be used on very low bandwidth connections. MQTT is currently undergoing standardization at OASIS. A broker stores the topic of message sent from the publisher client and releases the messages to the subscriber that requests or subscribes to a specific topic. Upadhyay et al. [5] describe it as a form of filter which only filters or sends the messages that are requested by the subscriber and sends an alert to the publisher after a request so that publisher can release its topic or data. There are other brokers namely, ActiveMQ, Apollo, JoramMQ, RabbitMQ [57]., and VerneMQ (proprietary and open source) that may be equally good as well. However, for experimental and research purposes, Mosquitto is highly preferred and mostly used [5,7,24,42,50,54].

We employ the use of Arduino Uno Rev 3 (Figure 3.1. a) that uses the ATmega328 microcontroller (16 MHz CPU) with 32KB in system programmable flash and 2KB internal SRAM. The Arduino Uno is Class-0 IoT device as discussed earlier. A device like this can be tasked to control actuators, electrical appliances, internet services, or collect data from temperature and humidity sensors. Also, an Ethernet Shield Wiznet 5100 (Figure 3.1. b) is used to establish communication over the router. It has a RJ45 connector and it can transfer data up to a speed of 100Mbps. The clients and server are connected via a TP-Link 150Mbps Wireless AP/Client Router with wired-LAN support (Figure 3.1. c).



a) b) c)

Figure 3.1. Devices used in the experiment: a) Arduino Uno Rev 3, b) W5100 Ethernet Shield, c) TP-LINK router

Reading of results data and the ability to know if the devices were communicating was seen throught the end-user. The server had other sub-servers running on it as well namely apache web server, mysql server and the MQTT broker server. Figure 3.2. shows the block diagram of the experimental setup.

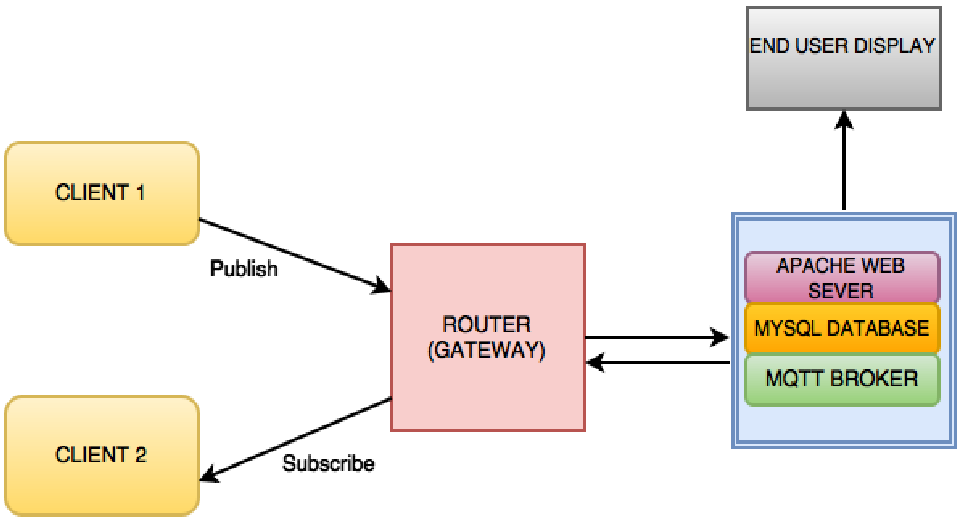


Figure 3.2. Block diagram of experimental setup.

3.2. Design And Implementation

The experimental design was based on a client end-to-end encryption/decryption. One of the device is setup to be the client-publisher and the other is se to be a client-subscriber. They are connected via the router that serves as a gateway for server - client communication. An encrytpion mechanism is performed at the publish-client and a decryption at the subscribe-client's end. Data for payload is hard-coded in the device. This mimics data readings from sensors. Figure 3.3. illustrates the flow of the client end-to-end encryption/decryption process. The encrypted data is published to the broker and the publish client is clearly independent of the state of the subscribe client. Communication can only be established when a published topic can be subscribed by the designated subsciber clients or clients that are subscribed to that particular topic. The publish/subscibe process can be both ways for to-and-fro publish/subscribe communication but it is made in one way in the case of this research.

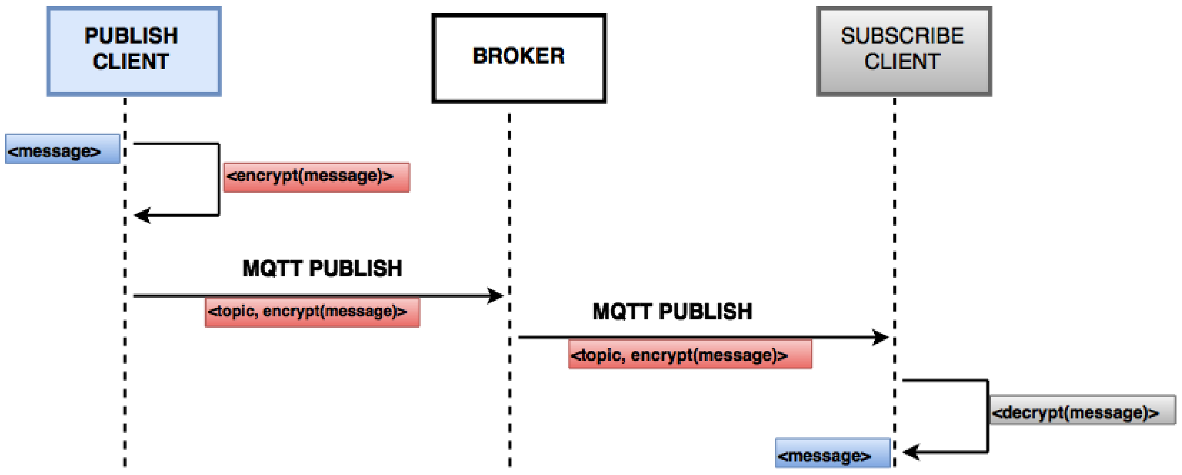
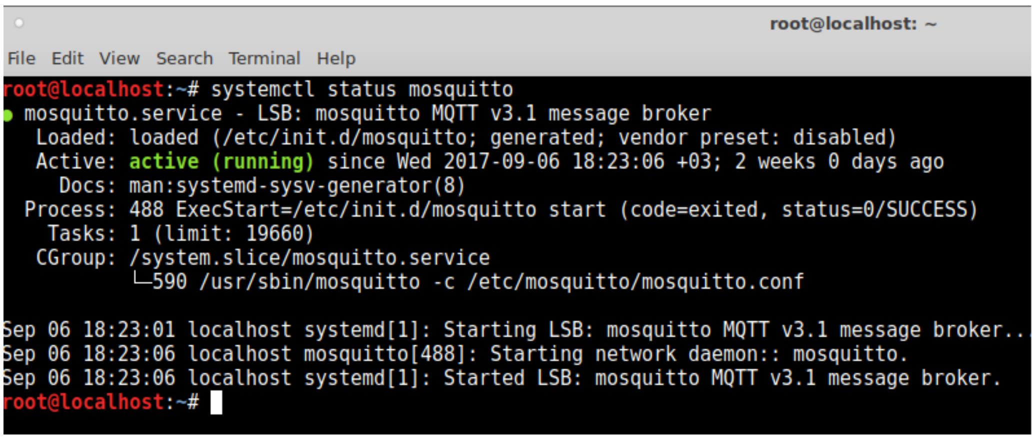


Figure 3.3. End-to-end client encryption mechanism based on MQTT.

This setup and end-to-end encryption and decryption mechanism is still prone to some attacks in a network environment that is most likely linked to the internet. Security vulnerabilities may include compromised devices, easy accessibility of data at rest in servers, timing attacks, denial of service (DoS) attacks, interception, replay attacks, main-in-the-middle attacks, alteration of data and disclosure of data. However the scope of this research does not cover solving vulnerabilities associated to such security mechnism. MQTT protocol is devoid of concrete security mechanisms apart from TLS/SSL certificates despite its load on network and impact on data transfer. For high end clients, it is fairly easier to implement TLS based on SSL communication. However, it is not the case for the resource constrained device due to its handshake and increased packet overhead [23,52].

For the implementation of the system, we use an MQTT broker known as Mosquitto and we also employ the use of free and open-source Eclipse Paho MQTT C/C++ client for embedded platforms and an Arduino-ready AES library. The installation of the broker was performed on the terminal of the the Kali Linux. Figure 3.4. illustrate the availability and active and running state of the Mosquitto broker.



```
root@localhost:~# systemctl status mosquitto
● mosquitto.service - LSB: mosquitto MQTT v3.1 message broker
   Loaded: loaded (/etc/init.d/mosquitto; generated; vendor preset: disabled)
   Active: active (running) since Wed 2017-09-06 18:23:06 +03; 2 weeks 0 days ago
     Docs: man:systemd-sysv-generator(8)
  Process: 488 ExecStart=/etc/init.d/mosquitto start (code=exited, status=0/SUCCESS)
    Tasks: 1 (limit: 19660)
   CGroup: /system.slice/mosquitto.service
           └─590 /usr/sbin/mosquitto -c /etc/mosquitto/mosquitto.conf

Sep 06 18:23:01 localhost systemd[1]: Starting LSB: mosquitto MQTT v3.1 message broker..
Sep 06 18:23:06 localhost mosquitto[488]: Starting network daemon:: mosquitto.
Sep 06 18:23:06 localhost systemd[1]: Started LSB: mosquitto MQTT v3.1 message broker.
root@localhost:~# 
```

Figure 3.4. Terminal active and running state of the Mosquitto broker.

We further implemented the code required for the Arduino Uno and Ethernet shield to communicate with the Mosquitto broker. Arduino provides an integrated development environment (IDE) for writing the Arduino commands or coding. It also has a serial monitor screen which receives the data logs directed to it for debugging purposes and to know what is transpiring in the Arduino device during its operation or communication in the case of this research. A serial cable connected to the computer from the Arduino device is able to send the logs and it is made available to the serial monitor. In the code, we set the broker server's internet protocol, QoS levels, the loop time for publish function, the message, topics, and also included the library for AES to implement encryption and decryption of the message. We run them on our Arduino device (class-0 IoT Device). The following are code snippets and their description of what they do.

```
byte mac [ ] = { 0x00, 0x11, 0x22, 0x33, 0x44, 0x77 };  //MAC ADDRESS
IPAddress dnServer(192, 168, 1, 1); //DNS SERVER IP
IPAddress gateway(192, 168, 1, 1);  //GATEWAY IP
IPAddress subnet(255, 255, 255, 0); //SUBNET IP
IPAddress ip(192, 168, 1, 105);     //IP FOR THE CLIENT DEVICE
```

Figure 3.5. Arduino code for MAC, internet protocol settings.

Figure 3.5. illustrates the Arduino code for setting up the MAC addressand IP for the Ethernet shield and Arduino Uno as a whole. These will help identify the device to connect to the router and enable the visibility of the device on the network.

```
const char* topicPub = "ARDUINO-PUB1/0"; //DECLARATION OF A CONSTANT TOPIC VARIABLE
MQTT::Message message; //MQTT CLIENT CLASS INSTANCE
void loop( )  {  //ARDUINO LOOP FUNCTION
  if (!client.isConnected( ))
            connect( ); // A CONNECT FUNCTION TO CONNECT DEVICE TO MQTT SERVER
   if (millis( ) - lastMillis > 2000) {  //RUN EVERY 2 SECONDS, HENCE PUBLISH EVERY 2 SECONDS
          lastMillis = millis( ); //A ASSIGNMENT OF THE CURRENT MILLISECOND
          char buf [33]; //DECLARATION OF A BUFFER CHARACTER VARIABLE
          strcpy(buf, "QoS0ARDUINO1_MSGQoS0ARDUINO1_MSG"); //32 BYTES MESSAGE IS
          COPIED INTO CHAR VARIABLE
          message.qos = MQTT::QOS2; //QUALITY OF SERVICE SETTING IS SET TO QoS LEVEL2
           message.retained = false; //NO MESSAGE RETAINED
           message.dup = false; //NO DUPLICATE
           message.payload = (void*) buf;  //ASSIGNING BUFFER CONTENT TO MQTT PAYLOAD
          VARIABLE
           message.payloadlen = strlen(buf)+1; //ASSIGNING LENGTH OF PAYLOAD TO MQTT PAYLOAD
          LENGTH //VARIABLE
           client.publish(topicPub, message); //PUBLISH CLIENT FUNCTION CALL
```

Figure 3.6. Arduino code snippet to publish data to the server.

After the TCP and MQTT connection has been succesful, the Arduino client is set to publish to the MQTT broker server. The basic parameters like QoS, retained and duplicate values are set. Also the topic and payload value are provided accordingly.

Figure 3.6. shows Arduino snippet used to publish 32 bytes of plaintext payload to the broker.

On the otherhand, Figure 3.7. illustrates the Arduino snippet for the encryption process. Clearly the code in Figure 2.6 was tweaked to include the AES 128 bits cipher block chaining encryption. This encrypts multiple blocks of 16 bytes data or payload of length 16. In other words, the data length must be in mod 16. Also this mechanism makes use of the initialization vector (IV) which is used along with the secret key. It increases the strength of encryption by preventing repetition in data encryption and thereby hindering or making attacks such as dictionary attacks more difficult since attacks tends to look at patterns from encrypted data. The length of the IV is generally the same as the length of the secret key.

```
const uint8_t iv[ ] = {0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15}; //INITIATION VECTOR

uint8_t key[ ] = {'a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'k', 'l', 'm', 'n', 'o', 'p'}; // DECLARE AES 128-BIT
KEY

const char* topicPub = "ARDUINO-PUB1/0"; //DECLARATION OF A CONSTANT TOPIC VARIABLE

void loop( ) //ARDUINO LOOP FUNCTION {

    if (!client.isConnected( ))

            connect( ); // A CONNECT FUNCTION TO CONNECT DEVICE TO MQTT SERVER IF IT FAILS

       if (millis( ) - lastMillis > 2000) {  //RUN EVERY 2 SECONDS, HENCE PUBLISH EVERY 2 SECONDS

          lastMillis = millis( ); //AN ASSIGNMENT OF THE CURRENT MILLISECOND

          char buf[33]; //DECLARATION OF A BUFFER CHARACTER VARIABLE

          strcpy(buf, "QoS0ARDUINO1_MSGQoS0ARDUINO1_MSG"); //MESSAGE IS COPIED INTO
          BUFFER //VARIABLE

          const uint16_t data_len = strlen(buf);  //DECLARATION OF SIZE OF MESSAGE FOR
          ENCRYPION

          aes128_cbc_enc(keyX, iv, buf, data_len); //AES 128-BITS CIPHER BLOCK CHAINING
          IMPLEMENTED

           message.qos = MQTT::QOS2; //QUALITY OF SERVICE SETTING IS SET TO QoS LEVEL2

           message.retained = false; //NO MESSAGE RETAINED

           message.dup = false; //NO DUPLICATE

           message.payload = (void*) buf;  //ASSIGNING BUFFER CONTENT TO MQTT PAYLOAD

           message.payloadlen = strlen(buf)+1; //ASSIGNING LENGTH OF PAYLOAD TO MQTT PAYLOAD
          LENGTH VARIABLE

           client.publish(topicPub, message); //PUBLISH CLIENT FUNCTION CALL
```

Figure 3.7. Arduino code snippet used to publish encrypted payload.

Decryption is performed at the subscribing client-end after the encrypted payload is is released successfully to it. The subscribe client has a function call to subscibe data associated to the particular topic. Furthermore the subscribe function has a parameter that has a message handler data type. This parameter takes a callback function that is able to release the data captured from the topic that was subscribed. Figure 3.8. illustrates the code snippet for subscription, callback function and the decryption.

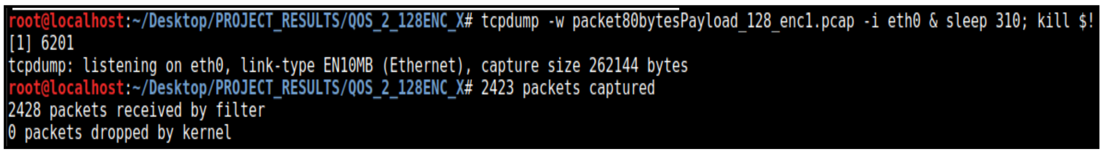
```
client.subscribe(topicSub, MQTT::QOS0, messageArrived); //SUBSCRIBE CLIENT FUNCTION CALL
//WITH A CALL TO CALLBACK FUNCTION

//CALLBACK FUNCTION TO SHOW MESSAGE FROM SUBSCRIBED TOPIC

void messageArrived(MQTT::MessageData& md){

  MQTT::Message &message = md.message;

  unsigned long ms = micros ();

  aes128_cbc_dec (keyX, iv, (void*) message.payload, data_len);  //AES-128BITS CBC DECRYPTION

  Serial.print ("Decryption took: ");

  Serial.println (micros() - mss);

  Serial.print ("Payload ");

  Serial.println((char*)message.payload);

}
```

Figure 3.8. Arduino code snippet for subscribe client and decryption.

3.3. Method Of Analysis

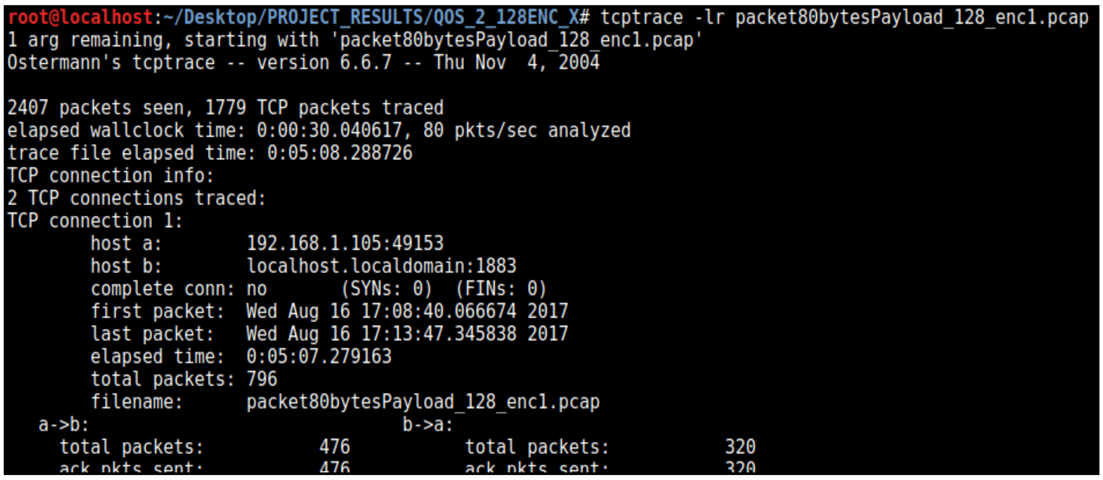
The experiment was treated in two folds; thus, the encrypted and the non-encrypted payload approach. We used a minimum data of 16 bytes and a maximum of 96 bytes with 16 bytes interval increment. The data was published in plaintext (non-encrypted message) with a topic every two seconds to the broker server. A subscriber client is set to listen to the broker for messages from the subscribed topic. Packets were captured using tcpdump [58] for a period of 310 seconds for each QoS and for an increment of 16 bytes of data till 96 bytes of data is reached. Tcpdump is an ideal tool which is free, runs on many Unix platforms, and has a Microsoft Windows version as well. The features of its syntax and its file format have been employed by a large number of programs and other capture software. Due to the fact that tcpdump is text based, it makes it easy to run remotely using even a Telnet connection. A lack of analysis is its biggest disadvantage, but it can easily capture network traffic and can be analyzed with other software. Similarly, the procedure was performed by applying the AES-128 bits CBC mechanism on the payload.

We computed the average end-to-end latency, percentage message loss and throughput as well as the encryption and decryption processing time. The latency was measured by using the difference in timestamp formed from the start of a published packet, the various acknowledgment packets flow to the broker server and its reception and acknowledgement packets flow form the broker by the subscriber client. With the aid of Wireshark [59]., the latency was recorded and the average throughput were recorded from the results generated by the statistics of tcptrace [58] on the captured packets. Wireshark is one of the most popular open-source packet analyzer along-side Capsa. Wireshark is cross-platform and it uses pcap (capture file format) to capture packets. It runs on Microsoft Windows Linux, Mac OS X, BSD, and Solaris [60]. According to N.A. Ben-Eid [60]., it is the most widely used, and it provides a larger number of supported protocols (more than 500) and possesses a user-driven support base that is unrivaled and it is more powerful. Tcptrace normally takes a tcpdump file that is specified on the command line or terminal and generate a summarization of the network communication and connections. Likewise, it can also take as input the generated files by other popular packet-capture programs, namely, snoop, etherpeek, and WinDump. Tcptrace can generate different types of output with information about each network connection available. These include elapsed time, round trip times, window advertisements, bytes and segments sent and received, throughput, retransmissions, and more. It can also produce a number of graphs for further analysis. Tcptrace chooses only valid samples found. A sample is recognized if an acknowledged packet is received from the destination for a previously transmitted packet from a source such that the acknowledgment value is 1 greater than the last sequence number of the packet. Also, it is a necessity that the packet that is being acknowledged is not retransmitted, and that no packets that came before it in the sequence space were retransmitted after the packet was transmitted. Figure 3.9. and Figure 3.10. shows the console execution of tcpdump and tcptrace respectively.



```
root@localhost:~/Desktop/PROJECT_RESULTS/QOS_2_128ENC_X# tcpdump -w packet80bytesPayload_128_enc1.pcap -i eth0 & sleep 310; kill $!
[1] 6201
tcpdump: listening on eth0, link-type EN10MB (Ethernet), capture size 262144 bytes
root@localhost:~/Desktop/PROJECT_RESULTS/QOS_2_128ENC_X# 2423 packets captured
2428 packets received by filter
0 packets dropped by kernel
```

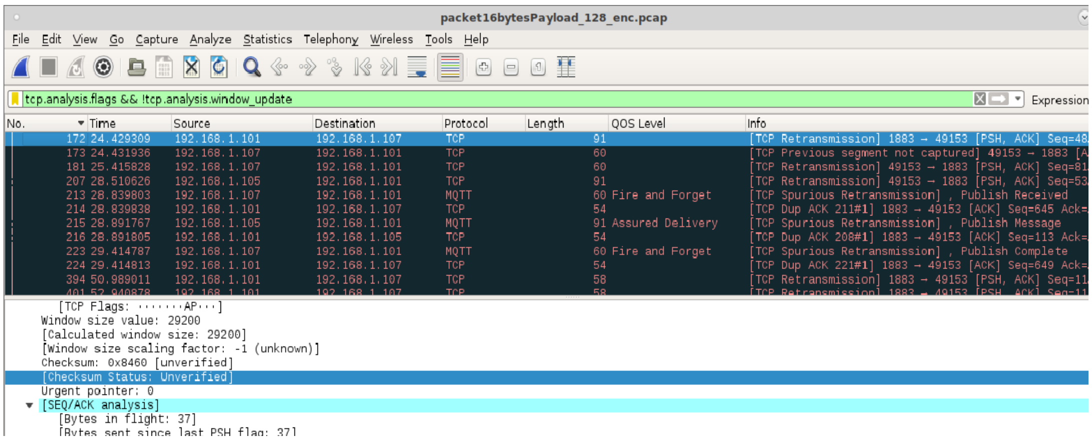
Figure 3.9. Tcpdump command to capture packet of an encypted payload of size 80 btyes and to stop the packet reading after 310 seconds.



```
root@localhost:~/Desktop/PROJECT_RESULTS/QOS_2_128ENC_X# tcptrace -lr packet80bytesPayload_128_enc1.pcap
1 arg remaining, starting with 'packet80bytesPayload_128_enc1.pcap'
Ostermann's tcptrace -- version 6.6.7 -- Thu Nov  4, 2004

2407 packets seen, 1779 TCP packets traced
elapsed wallclock time: 0:00:30.040617, 80 pkts/sec analyzed
trace file elapsed time: 0:05:08.288726
TCP connection info:
2 TCP connections traced:
TCP connection 1:
        host a:        192.168.1.105:49153
        host b:        localhost.localdomain:1883
        complete conn: no       (SYNs: 0)  (FINs: 0)
        first packet:  Wed Aug 16 17:08:40.066674 2017
        last packet:   Wed Aug 16 17:13:47.345838 2017
        elapsed time:  0:05:07.279163
        total packets: 796
        filename:      packet80bytesPayload_128_enc1.pcap
   a->b:                              b->a:
     total packets:          476           total packets:           320
     ack pkts sent:          476           ack pkts sent:           320
```

Figure 3.10. Tcptrace command to show traced packet analysis results of an encypted payload of size 80 bytes.

Furthermore, we employed the use of Wireshark application to count the number of TCP analysis flags that includes, TCP retransmissions, spurious retransmissions, duplication acknowledgements, and previous segments not captured, as message loss. We computed the percentage of these loss packets to total TCP packets accordingly. Figure 3.11. Shows the Wireshark sample of TCP flags noted as message loss.



packet16bytesPayload_128_enc.pcap
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
tcp.analysis.flags && !tcp.analysis.window_update
Expression

```
No.    Time         Source          Destination     Protocol  Length  QOS Level         Info
   172 24.429309    192.168.1.101   192.168.1.107   TCP          91                     [TCP Retransmission] 1883 → 49153 [PSH, ACK] Seq=48
   173 24.431936    192.168.1.107   192.168.1.101   TCP          60                     [TCP Previous segment not captured] 49153 → 1883 [A
   181 25.415828    192.168.1.107   192.168.1.101   TCP          60                     [TCP Retransmission] 49153 → 1883 [PSH, ACK] Seq=81
   207 28.510626    192.168.1.105   192.168.1.101   TCP          91                     [TCP Retransmission] 49153 → 1883 [PSH, ACK] Seq=53
   213 28.839803    192.168.1.107   192.168.1.101   MQTT         60 Fire and Forget     [TCP Spurious Retransmission] , Publish Received
   214 28.839838    192.168.1.101   192.168.1.107   TCP          54                     [TCP Dup ACK 211#1] 1883 → 49153 [ACK] Seq=645 Ack=
   215 28.891767    192.168.1.105   192.168.1.101   MQTT         91 Assured Delivery    [TCP Spurious Retransmission] , Publish Message
   216 28.891805    192.168.1.101   192.168.1.105   TCP          54                     [TCP Dup ACK 208#1] 1883 → 49153 [ACK] Seq=113 Ack=
   223 29.414787    192.168.1.107   192.168.1.101   MQTT         60 Fire and Forget     [TCP Spurious Retransmission] , Publish Complete
   224 29.414813    192.168.1.101   192.168.1.107   TCP          54                     [TCP Dup ACK 221#1] 1883 → 49153 [ACK] Seq=649 Ack=
   394 50.989011    192.168.1.101   192.168.1.107   TCP          58                     [TCP Retransmission] 1883 → 49153 [PSH, ACK] Seq=11
   401 52.940878    192.168.1.101   192.168.1.107   TCP          58                     [TCP Retransmission] 1883 → 49153 [PSH, ACK] Seq=11
```

```
      [TCP Flags: ·······AP···]
    Window size value: 29200
    [Calculated window size: 29200]
    [Window size scaling factor: -1 (unknown)]
    Checksum: 0x8460 [unverified]
    [Checksum Status: Unverified]
    Urgent pointer: 0
  ▼ [SEQ/ACK analysis]
      [Bytes in flight: 37]
      [Bytes sent since last PSH flag: 37]
```

Figure 3.11. Wireshark interface that shows sample of TCP flags to depict message loss.

CHAPTER 4. EXPERIMENT RESULTS AND DISCUSSION

In this chapter, we present our analysis results of the end-to-end latency, percentage message loss, throughput and the encryption and decryption processing time. We also present the correlation analysis between throughput and end-to-end latency for encrypted and non-encrypted payload as well as the deductions from various graphs according to QoS levels and payload size.

4.1. Non-encrypted Payload Analysis Results

We present the results from the use of plaintext as payload. The results are in graphs for simplicity and understanding. Other results are presented from other statistical computation and calculations. Figure 4.1. shows the average end-to-end latency analysis result in relation to payload sizes and QoS levels when the payload was plaintext. The QoS 2-line graph uses the secondary axis on the right for reading purposes.

It can be noted clearly from Figure 4.1. that there is a high latency recorded when QoS level 2 is implemented. This is as a result of the 4-way handshake it uses [43]. QoS level 0 is observed to be numerically lower in latency since it uses the publish and forget principle used (at most once) as compared to the assured delivery principle (exactly once) of QoS level 2 [8,29,43,46]. Other studies [43,61]., came to similar conclusion of this fact, and showed that QoS level 2 has a higher end-to-end delay (latency) than QoS level 0.

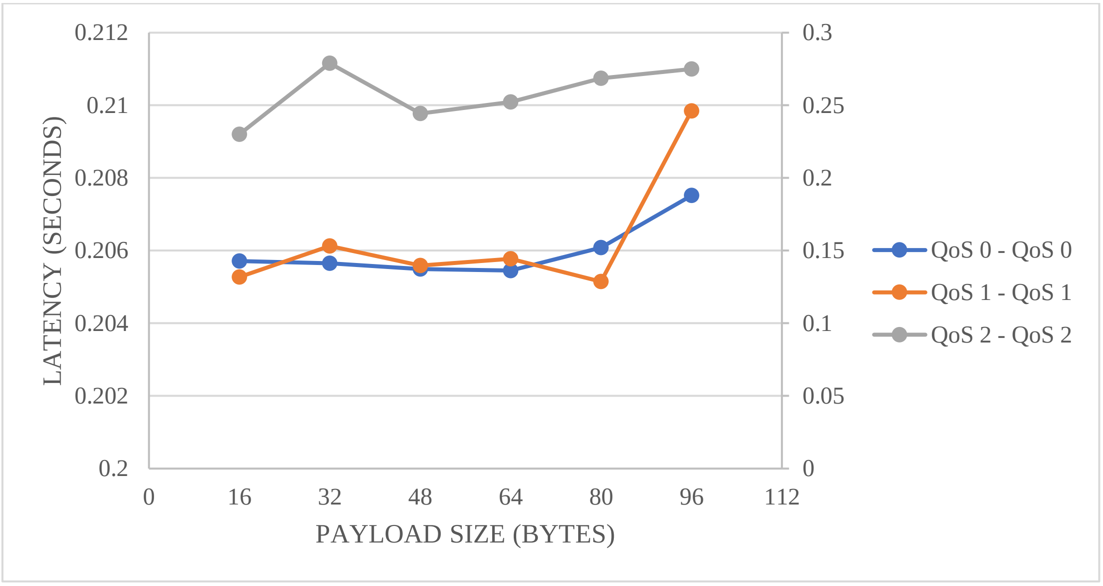


Figure 4.1. Non-encrypted payload average end-to-end latency analysis result.

Furthermore, Figure 4.2. shows the comparison of message loss, payload size and QoS for non-encrypted payload. When QoS level 2 was implemented, the percentage of the average percentage loss of all the payloads (16-96 bytes) was reduced by approximately 59.17% as compared to QoS level 0 when payload was not encrypted. Despite the high latency with QoS level 2, it is efficient at message delivery by 2.45 times than QoS 0 according to results from the average percentage loss of all the various payloads that was not encrypted.

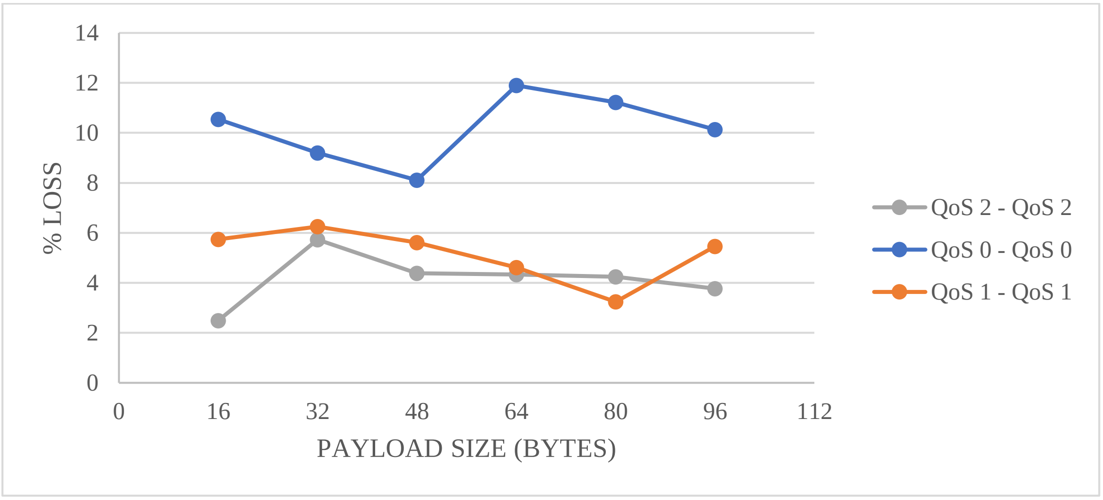


Figure 4.2. Non-encrypted payload loss percentage analysis result.

From Figure 4.3., we can deduce that the throughput is directly proportional to the payload size and QoS. They are marginally close according to QoS as the payload increases. QoS level 2 is higher in throughput due to the higher number of packets exchanged.

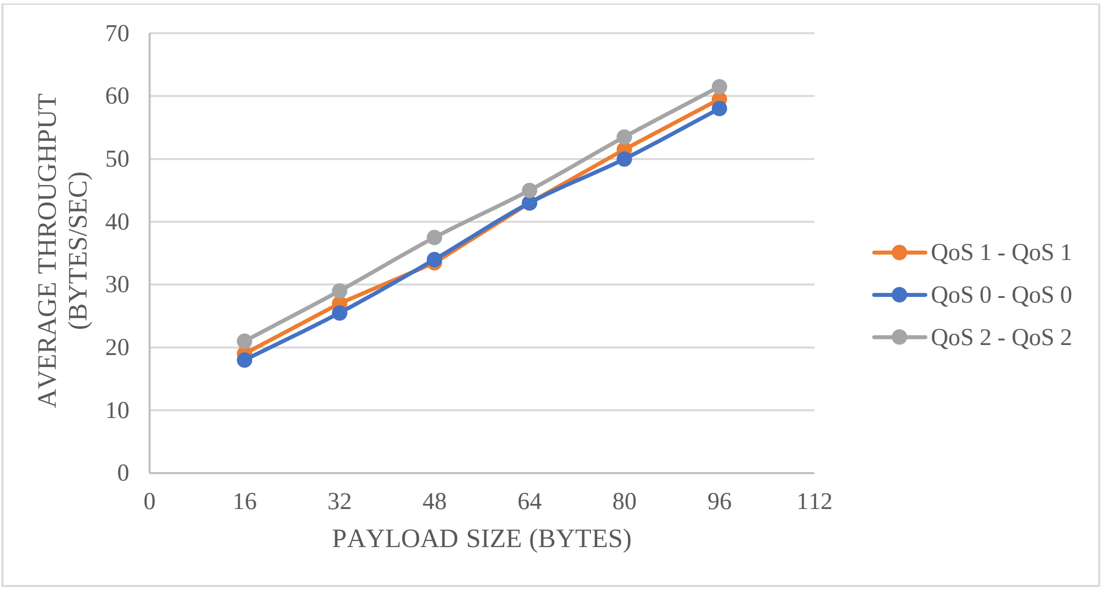


Figure 4.3. Non-encrypted payload throughput analysis result.

4.2. Encrypted Payload Analysis Results

We present the results from the use of AES 128 bits CBC encrypted plaintext as payload. The results are in graphs for simplicity and understanding. Other results are presented from other statistical computation and calculations. Figure 4.4. shows the average end-to-end latency analysis result in relation to different payload sizes and QoS levels for encrypted payload. The QoS level 2-line graph uses the secondary axis on the right for reading purposes. QoS level 1 is relatively higher in latency than QoS 0 and lies below QoS 2. QoS 0 and QoS 1 of both graphs (Figure 4.1. and 4.4.) are marginally below 0.21 seconds however, the graph of QoS level 2 lies between 0.23 and 0.28 seconds. Table 4.1. shows the averages of the end-to-end latencies for each QoS.

Table 4.1. Averages of the end-to-end latencies for each QoS.

	Encrypted payload (Seconds)	Non-encrypted payload (Seconds)
QoS 0	0.205927	0.205986
QoS 1	0.206358	0.206293
QoS 2	0.256868	0.258206

According to the results from Figure 4.5., the average percentage loss of all the payloads showed that QoS level 2 reduced its percentage loss by 54.64 % compared to QoS level 0 for encrypted payload. Similarly, 59.17% reduction was observed for non-encrypted payload.

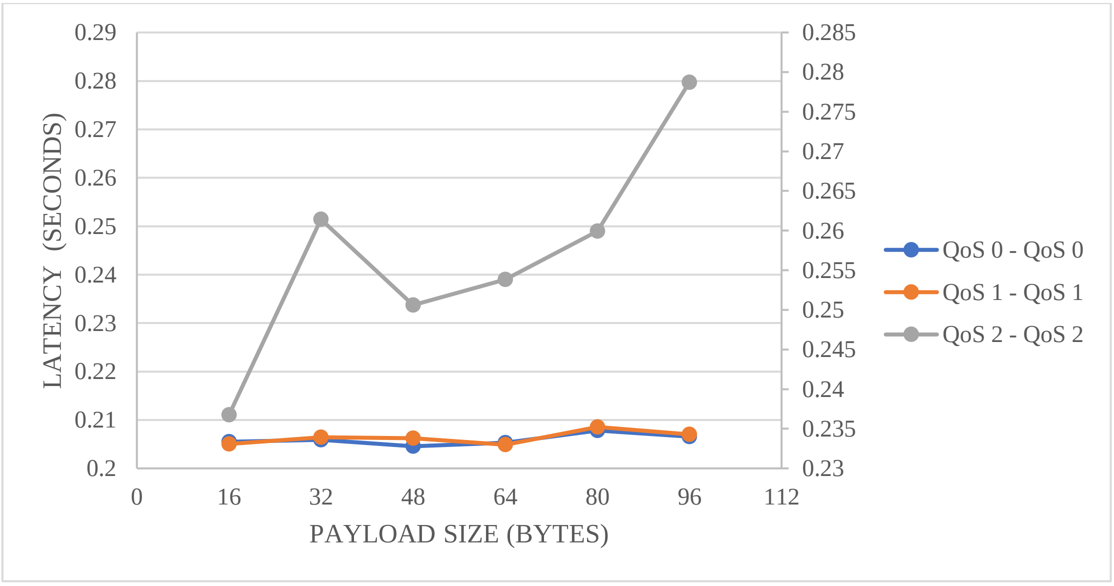


Figure 4.4. Encrypted payload average end-to-end latency analysis result.

Furthermore, QoS level 2 was efficient at message delivery than QoS level 0 by 2.20 times for encrypted payload as compared to the 2.45 times when payload was plaintext. Also, at 16 bytes for QoS level 2, the percentage loss for encrypted payload was 1.6 times (about twice) more than that of non-encrypted payload. Likewise, at 80 bytes the percentage loss for encrypted payload was approximately 1.53 times more than non-encrypted payload.

The average of the percentage loss of non-encrypted payload at QoS level 2 was 4.158% as compared to 4.693% for the encrypted payload. These figures show that, on the average, encrypted payload lost messages by 1.13 times more than non-encrypted payloads at QoS level 2. From Wireshark statistics, the average packet size of a packet loss for encrypted payload at 16 bytes for QoS level 2 was 1.5 bytes more than non-encrypted payload and likewise it was 2 bytes more for 80 bytes payload. Thus, for a 100 packets of message losses for encrypted payload of size 80 bytes, we can have about 200 bytes of message losses more than non-encrypted payload loss as network load.

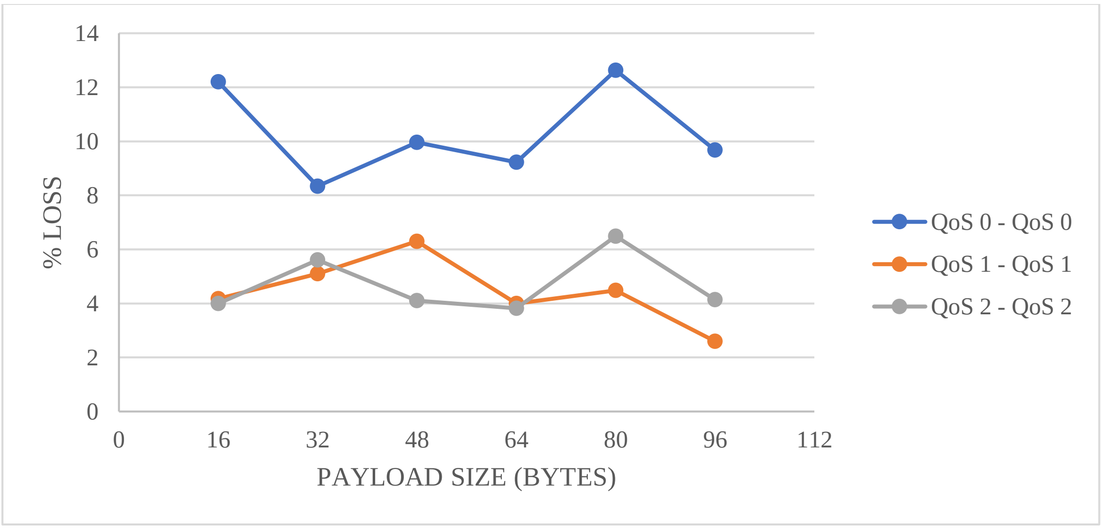


Figure 4.5. AES-128 BITS encrypted payload loss percentage analysis result.

We further observed from Figure 4.6. that, the throughput was marginally similar to that of the non-encrypted payload (see Figure 4.3.) and it increases with an increase in payload and QoS level. Table 4.2. shows the averages of the throughput for each QoS level.

Table 4.2. Averages of the throughput for each QoS level.

	Encrypted payload (bytes/second)	Non-encrypted payload (bytes/second)
QoS 0	37.916670	38.083330
QoS 1	39.083330	38.916670
QoS 2	41.250000	41.250000

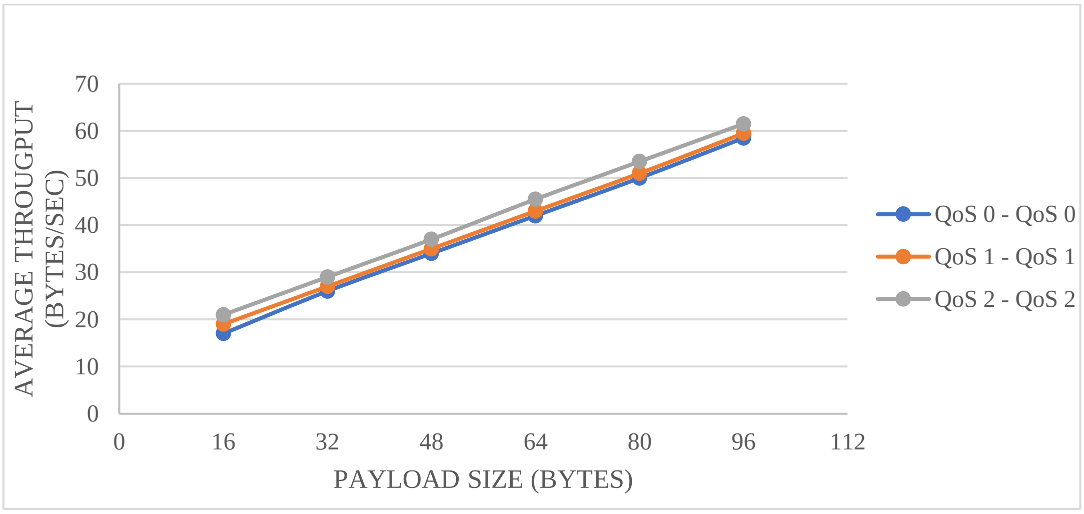


Figure 4.6. Encrypted payload throughput analysis result.

Also in addition to our analysis, results for Figure 4.7. shows the processing time for encryption and decryption is divergently increasing in microseconds. It shows that the higher the payload, the higher the encryption and decryption time. However, by calculating the point of interception of the line, the decryption time is clearly greater than the encryption time after the interception point (1.58 Bytes, 299.28µs) and it diverges after that point. This does not affect network load but rather the processing time at the client end.

With proper optimization in code to decrease encryption/decryption time, there is a greater chance to find a payload size that produces same time for encryption and decryption that could be negligible or small enough for resource constrained devices. This processing time will be nearly negligible and data privacy services can be improved especially in the area of client-side encryption techniques. This will be useful for data, such as temperature and humidity readings that are small, or sensitive enough and are needed to be kept private.

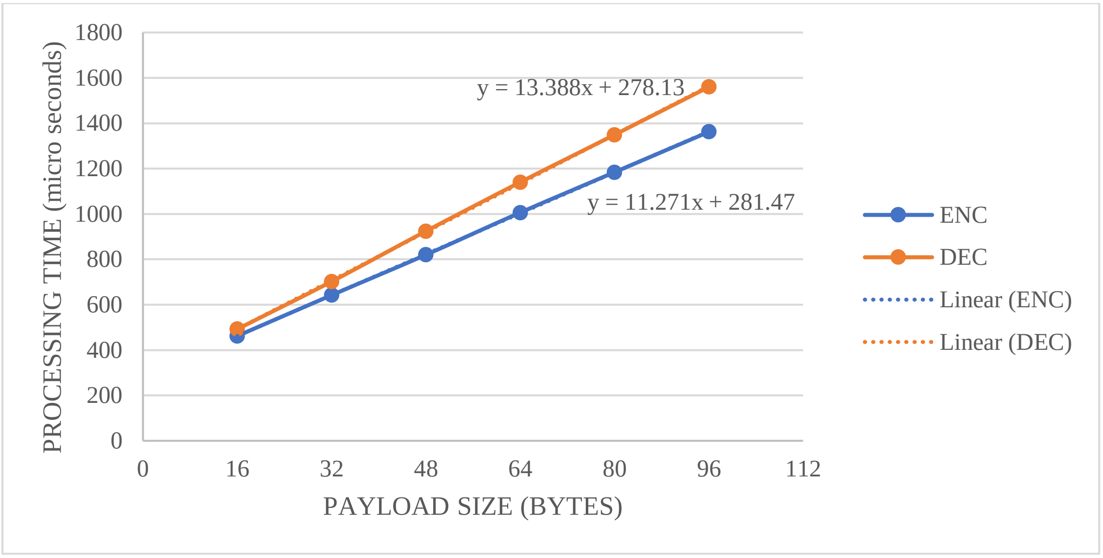


Figure 4.7. Average processing time for encryption and decryption analysis.

4.3. Correlation Analysis Based On Throughput and End-to-end latency

We also computed the correlation coefficients between the various end-to-end latency and the average throughputs. A correlation analysis is a statistical technique that shows the influence a set of variables have over another set of variables. It is one of the most common and useful statistical technique used to show the strength of the relationship between pairs of a set of variables. The coefficient of correlation (r) is in the form of a single decimal number defined as $-1 < r < 1$. Hence, the more the coefficient is closer to 1 it implies that it has a positive correlation and if it is closer to -1 then it implies a negative correlation. Table 4.3. summarizes the correlation coefficients from the results.

It showed that end-to-end latency is positively correlated to throughput for each QoS level and for both non-encrypted and encrypted payload. Thus, end-to-end latency and throughput are closely related. From the results, we can notice a pattern for the encrypted payload’s correlation coefficient. It tends to steadily becomes stronger when the QoS increases. It is strongest at QoS level 2. And this is totally not the case when payload is plain-text. Its strongest correlation occurs when the QoS is at level 0 and it steadily decreases as QoS levels increases. How this came about is not immediately known. However, this may call for further research to ascertain this pattern.

Table 4.3. Correlation analysis of latency to throughput for MQTT QoS.

	Encrypted payload	Non-encrypted payload
QoS 0	0.553594	0.689427
QoS 1	0.582387	0.610147
QoS 2	0.802049	0.558472

CHAPTER 5. CONCLUSION AND FUTURE STUDIES

This chapter is set aside to bring every aspect of this thesis into a conclusion. The thesis was introduced in the first chapter where a general theoretical background into the definition of internet of things and architecture, protocols available and IoT security was established in the context of the thesis. Furthermore, we looked at the research statement and questions that led to the basis of this thesis work. The purpose, motivation and limitations of this thesis was also introduced in the beginning chapter. The second chapter took a look into the MQTT protocol where a general overview, main features and technicalities involved in MQTT packets was discussed. The next chapter introduced the research method that was used by establishing the experimental set-up, design and implementation and the method of analysis applied in the experiment. Finally, prior to this chapter, the outcomes and results of the experiments was established accordingly and discussed. Furthermore, in this chapter we take a look at future research and studies.

In this thesis, a series of experiments were performed using a low-end/resource constrained (Class 0-IoT) device with encrypted and non-encrypted payload (plaintext) based on MQTT. The results showed that non-encrypted payload have a lower network load effect and hence produces a relatively better network performance using MQTT in terms of percentage loss and message delivery than the encrypted payload. However, the effects on network performance may be negligible, and this may depend on the amount of resources available.

Numerically, QoS level 2 was observed to be efficient in terms of better delivery as expected and minimal message loss for non-encrypted payload. Furthermore, encryption and decryption processing time are observed to be lower and nearly equal at payloads less than 16 bytes (i.e., approximately 2 bytes). Hence a well optimized code for encryption and an optimum size of payload can make encryption and decryption processing time nearly negligible or small. We also calculated the correlation coefficients of end-to-end latency and average throughput based on different QoS levels. The results showed that the end-to-end latency is closely related to the throughput for both encrypted and non-encrypted payloads.

So as part of the conclusion of this thesis, there is a need to establish the answers to the initial questions from the first chapter of the thesis.

Q1: Can encrypted payload based on AES-128 bits affect network performance or characteristic?
From the results in the previous chapter, it was clear that the encryption mechanism applied on the payloads had an effect on network parameters numerically; especially message loss.

Q2: Will it have the similar effect as compared to an unencrypted payload/plaintext for different MQTT QoS levels and different payload sizes?
According to the presented results in the previous chapter, the difference in effects was not close to being called same; especially for message loss. However, it could be regarded as negligible for device that are much resource-enabled. It showed that encrypted messages get a higher loss in transmission that plaintext. The throughput and latency results were marginally close to each other numerically.

As part of the future studies, similar analysis by making room for the case of wireless connections, publishing through an untrusted network via the internet and implementing a security analysis will be studied. There could be researches into similar or different cryptographic algorithms as well. Also, further studies will seek to find the optimal QoS level and optimal payload size for Class-0 IoT device and compute performance ratings of MQTT broker server and clients.

REFERENCES

[1] Weyrich, M., Ebert, C., Reference Architectures for the Internet of Things, IEEE Software, vol. 33, no. 1, pp. 112-116, 2016.

[2] Haller, S., Karnouskos, S., Schroth, C., The Internet of Things in an Enterprise Context, in: J. Domingue (ed.), D. Fensel (ed.), P. Traverso (ed.), Future Internet – FIS 2008, Springer, pp. 14–28, 2009.

[3] King, J., Awad, A.I., A distributed security mechanism for Resource-Constrained IoT Devices, Inform., vol. 40, no. 1, pp. 133–143, 2016.

[4] Höller, J., Tsiatsis, V., Mulligan, C., Karnouskos, S., Avesand, S., Boyle, D., From Machine-to-Machine to the Internet of Things: Introduction to a New Age of Intelligence. Elsevier, 1st edn. (2014).

[5] Upadhyay, Y., Borole A., Dileepan, D., MQTT based secured home automation system, 2016 Symp. Colossal Data Anal. Networking, CDAN 2016.

[6] Salunke, P., Nerkar, R., IoT Driven Healthcare System for Remote Monitoring of Patients, no. June, pp. 3–6, 2017.

[7] Singh, M., Rajan, M.A., Shivraj, V.L., Balamuralidhar, P., Secure MQTT for Internet of Things (IoT), Proc. - 2015 5th Int. Conf. Commun. Syst. Netw. Technol. CSNT 2015, pp. 746–751, 2015.

[8] Aziz, B., A formal model and analysis of an IoT protocol, Ad Hoc Networks, vol. 36, pp. 49–57, 2016.

[9] Biswas, D., Ramamurthy, R., Edward, S.P., Dixit, A., The Internet of Things: Impact and Applications in the High-Tech Industry, Cognizant 20-20 Insights, Cognizant, 2015.

[10] Howard, P.N., Sketching out the Internet of Things trendline, Brookings, 2015, https://www.brookings.edu/blog/techtank/2015/06/09/sketching-out-the-internet-of-things-trendline., Accessed date: 13.09.2017.

[11] Fysarakis, K., Askoxylakis, I., Soultatos, O., Papaefstathiou, I., Manifavas, C., Katos, V., Which IoT protocol? Comparing standardized approaches over a common M2M application, 2016 IEEE Glob. Commun. Conf. GLOBECOM 2016 - Proc., 2016.

[12] Rahman, R.A., Shah B., Security analysis of IoT protocols: A focus in coap, 2016 3rd MEC Int. Conf. Big Data Smart City, ICBDSC 2016, pp. 172–178, 2016.

[13] Palattella, M.R., et al., Standardized protocol stack for the internet of (important) things, IEEE Commun. Surv. Tutorials, vol. 15, no. 3, pp. 1389–1406, 2013.

[14] Bassi, A., Horn, G., Internet of Things in 2020, The European Technology Platform on Smart Systems Integration (eposs), 2008, http://www.smartsystemsintegration.org/public/documents/publications/Internet-of-Things_in_2020_EC-eposs_Workshop_Report_2008_v3.pdf., Access date:13.09.2017.

[15] Building the Internet of Things, Cisco, 2014, http://cdn.iotwf.com/resources/72/IoT_Reference_Model_04_June_2014.pdf. , Access date: 13.09.2017.

[16] Green, J., The Internet of Things Reference Model, Internet of Things World Forum, pp. 1–12, 2014.

[17] Tuwanut, P., Kraijak, S., A survey on IoT architectures, protocols, applications, security, privacy, real-world implementation and future trends, 11th Int. Conf. Wirel. Commun. Netw. Mob. Comput. (wicom 2015), pp. 6, 2015.

[18] Li, S., Da Xu, L., Zhao, S., The internet of things: a survey, Inf. Syst. Front., vol. 17, no. 2, pp. 243–259, 2015.

[19] Gubbi, J., Buyya, R., Marusic, S., Palaniswami, M., Internet of Things (IoT): A vision, architectural elements, and future directions, Futur. Gener. Comput. Syst., vol. 29, no. 7, pp. 1645–1660, 2013.

[20] Bormann, A.K.C., Ersue, M., Terminology for Constrained Node Networks, Internet Eng. Task Force (IETF), Informational 2070-1721, pp. 1–17, 2014.

[21] Jain, R., Constrained Application Protocol for Internet of Things vol. 857, pp. 1–12, 2014, https://www.cse.wustl.edu/~jain/cse574-14/ftp/coap.pdf., Access date: 13.09.2017

[22] Taneja, M., Lightweight security protocols for smart metering, 2013 IEEE Innov. Smart Grid Technol. - Asia, ISGT Asia 2013, pp. 1–5, 2013.

[23] Babovic, Z., Protic, J., Milutinovic, V., Web Performance Evaluation for Internet of Things Applications, IEEE Access, vol. PP, no. 99, 2016.

[24] Katsikeas, S., Fysarakis, K., Miaoudakis, A., Van Bemten, A., Askoxylakis, I., Lightweight & Secure Industrial IoT Communications via the MQ Telemetry Transport Protocol, 22nd IEEE Symposium on Computers and Communications (ISCC 2017), Crete, 2017.

[25] Cucinotta, T., Mancina, A., Anastasi, G.F., Lipari, G., Mangeruca, L., Checcozzo, R., Rusina, F., A Real-Time Service-Oriented Architecture for Industrial Automation, IEEE Trans. Ind. Informatics, vol. 5, no. 3, pp. 267–277, 2009.

[26] Lee, C., Zappaterra, L., Choi, K., Choi, H.A., Securing smart home: Technologies, security challenges, and security requirements, 2014 IEEE Conf. Commun. Netw. Secur. CNS 2014, pp. 67–72, 2014.

[27] Abomhara, M., Security and Privacy in the Internet of Things : Current Status and Open Issues, Priv. Secur. Mob. Syst. (PRISMS), 2014 Int. Conf., pp. 1–8, 2014.

[28] Luzuriaga, J.E., Cano, J.C., Calafate, C., Manzoni, P., Perez, M., Boronat, P., Handling mobility in IoT applications using the MQTT protocol, 2015 Internet Technol. Appl. ITA 2015 - Proc. 6th Int. Conf., pp. 245–250, 2015.

[29] Aziz, B., A formal model and analysis of the MQ telemetry transport protocol, Proc. - 9th Int. Conf. Availability, Reliab. Secur. ARES 2014, pp. 59–68, 2014.

[30] Masek, P., et al., Implementation of True IoT Vision: Survey on Enabling Protocols and Hands-On Experience, Int. J. Distrib. Sens. Networks, vol. 2016, 2016.

[31] Thangavel, D., Ma, X., Valera, A., Tan, H.X., Tan, C.K.Y., Performance evaluation of MQTT and coap via a common middleware," IEEE ISSNIP 2014 - 2014 IEEE 9th Int. Conf. Intell. Sensors, Sens. Networks Inf. Process. Conf. Proc., no. April, pp. 21–24, 2014.

[32] Durkop, L., Czybik, B., Jasperneite, J., Performance evaluation of M2M protocols over cellular networks in a lab environment, 2015 18th International Conference on Intelligence in Next Generation Networks, Paris, 2015, pp. 70-75.

[33] Mathur, A., Newe, T., Elgenaidi, W., Rao, M., Dooly, G., Toal D., A Secure End-to-End IoT Solution, Sensors Actuators A Phys., no. 1, pp. 1–29, 2017.

[34] Vučinić, M., Tourancheau, B., Rousseau, F., Duda, A., Damon, L., Guizzetti, R., OSCAR: Object security architecture for the Internet of Things, Proceeding of IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks 2014, Sydney, NSW, 2014, pp. 1-10.

[35] Behrens, R., Ahmed, A., Internet of Things: An end-to-end security layer, Proc. 2017 20th Conf. Innov. Clouds, Internet Networks, ICIN 2017, pp. 146–149, 2017.

[36] Islam, K., Shen,W., Wang, X., Security and privacy considerations for Wireless Sensor Networks in smart home environments, Proc. 2012 IEEE 16th Int. Conf. Comput. Support. Coop. Work Des., pp. 626–633, 2012.

[37] Rescorla, E., Dierksv, T., INC RTFM, The Transport Layer Security (TLS) Protocol Version 1.3, Internet Engineering Task Force (IETF), Standards Track RCF5246, March 2008, http://tools.ietf.org/html/draft-ietf-tls-tls13-05, Access date: 13.09.2017.

[38] Ukil, A., Bandyopadhyay, S., Bhattacharyya, A., Pal, A., Bose, T., Lightweight security scheme for IoT applications using coap, Int. J. Pervasive Comput. Commun., vol. 10, no. 4, pp. 372–392, 2014.

[39] Federal Information, Announcing the ADVANCED ENCRYPTION STANDARD (AES), November, 2001. http://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.197.pdf., Access date: 13.09.2017.

[40] Stallings, W., Cryptography and Network Security: Principles and Practice, 4th ed. Upper Saddle River, NJ, USA: Prentice Hall Press, 2005.

[41] Goyal, T.K., Sahula, V., Lightweight security algorithm for low power IoT devices, 2016 Int. Conf. Adv. Comput. Commun. Informatics, ICACCI 2016, pp. 1725–1729, 2016.

[42] Thatmann, D., Zickau, S., Förster, A., Küpper, A., Applying Attribute-Based Encryption on Publish Subscribe Messaging Patterns for the Internet of Things, 2015 IEEE International Conference on Data Science and Data Intensive Systems, Sydney, NSW, 2015, pp. 556-563.

[43] Lee, S., Kım, H., Hong, D.K., Ju, H., Correlation analysis of MQTT loss and delay according to qos level, Int. Conf. Inf. Netw., pp. 714–717, 2013.

[44] Mektoubi, A., Hassani, H.L., Belhadaoui, H., Rifi, M., Zakari, A., New approach for securing communication over MQTT protocol A comparaison between RSA and Elliptic Curve, Proc. - 2016 3rd Int. Conf. Syst. Collab. Sysco 2016, vol. 0, 2017.

[45] Arduino, https://www.arduino.cc., Access date: 20.08.2017.

[46] IBM AND EUROTECH, MQTT V3.1 Protocol Specification, pp. 1–42, 2010.

[47] Cohn, R.J., Coppen, R.J., Banks, A., Gupta, R., MQTT Version 3.1.1, OASIS Stand., no. December, pp. 1–81, 2015.

[48] MQTT Essentials Part 2: Publish and Subscribe, http://www.hivemq.com/blog/mqtt-essentials-part2-publish-subscribe., Access date: 13.09.2017.

[49] MQTT Essentials: Client, Broker and Connection Establishment, http://www.hivemq.com/blog/mqtt-essentials-part-3-client-broker-connection-establishment., Access date: 13.09.2017.

[50] MQTT topics and subscription, https://mosquitto.org/man/mqtt-7.html., Access date: 13.09.2017.

[51] Lesjak, C., et al., Securing smart maintenance services: Hardware-security and TLS for MQTT, Proceeding - 2015 IEEE Int. Conf. Ind. Informatics, INDIN 2015, pp. 1243–1250, 2015.

[52] Stansberry, J., MQTT and coap: Underlying Protocols for the IoT, http://www.electronicdesign.com/iot/mqtt-and-coap-underlying-protocols-iot, Access date: 14.09.2017.

[53] Kali Linux, https://www.kali.org., Access date: 13.08.2017.

[54] Collina, M., Corazza, G.E., Vanelli-Coralli A., Introducing the QEST broker: Scaling the IoT by bridging MQTT and REST, IEEE Int. Symp. Pers. Indoor Mob. Radio Commun. PIMRC, pp. 36–41, 2012.

[55] Atmoko, R.A., Riantini, R., Hasin, M.K., IoT real time data acquisition using MQTT protocol, International Conference on Physical Instrumentation And advanced Materials, 2016.

[56] Mosquitto Open Source MQTT v3.1/v3.1.1 Broke, https://mosquitto.org., Access date: 01-07-2017.

[57] Scalagent, Benchmark of MQTT servers, vol. 3, no. January, pp. 1–15, 2015.

[58] Dart, E., Johnston, B., Lake, A., Pouyoul, E., Rotman, L., Tierney, B., Using tcpdump, tcptrace, & xplot to Debug Network Problems, 2013.

[59] Wireshark, https://www.wireshark.org., Access date: 01.07.2017.

[60] Ben-Eid N.A., Ethical Network Monitoring Using Wireshark and Colasoft Capsa as Sniffing Tools, Int. J. Adv. Res. Comput. Commun. Eng., vol. 4, no. 3, pp. 471–478, 2015.

[61] Govindan, K., Azad, A.P., End-to-end service assurance in IoT MQTT-SN, 2015 12th Annu. IEEE Consum. Commun. Netw. Conf. CCNC 2015, pp. 290–296, 2015.

RESUME

Nanabayin was born on 5th April 1990 in Ghana. He completed his primary, middle school, high school and undergraduate education in Accra. In 2008, he graduated from the General Science class of St. Thomas Aquinas senior high school. From 2009 to 2013, he successfully undertook undergraduate level course in BSc. Computer Science and Mathematics and graduated with first class honours. Afterwards, he started work with Blupay Systems Ghana as a software developer. In 2014, he won a Turkish Government scholarship and undertook a Turkish language preparation course at Sakarya University (TÖMER) where he completed successfully in mid-2015. Afterwards, he started MSc. Computer and Information Engineering in Sakarya University, Turkey. During his time in 2016, he had the opportunity to do a network security and IoT communication research internship at Tomas Bata University in Zlin, Czech Republic for a period of three (3) months. He successfully completed his MSc program in November 2017.