INTEGRATION OF MESSAGE QUEUE TELEMETRY TRANSPORT PROTOCOL AND CONSTRAINED APPLICATION PROTOCOL FOR DATA COMMUNICATION IN WIRELESS SENSOR NETWORKS

BY

NWANKWO, Emmanuel Ikechukwu
MENG/SEET/2017/7555

DEPARTMENT OF TELECOMMUNICATION ENGINEERING, SCHOOL OF ELECTRICAL ENGINEERING AND TECHNOLOGY, FEDERAL UNIVERSITY OF TECHNOLOGY, MINNA

OCTOBER, 2021

INTEGRATION OF MESSAGE QUEUE TELEMETRY TRANSPORT PROTOCOL AND CONSTRAINED APPLICATION PROTOCOL FOR DATA COMMUNICATION IN WIRELESS SENSOR NETWORKS

BY

NWANKWO, Emmanuel Ikechukwu

MENG/SEET/2017/7555

A THESIS SUBMITTED TO THE POSTGRADUATE SCHOOL, FEDERAL UNIVERSITY OF TECHNOLOGY, MINNA, IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE AWARD OF THE DEGREE OF MASTER OF ENGINEERING IN COMMUNICATION ENGINEERING

OCTOBER, 2021

ABSTRACT

Wireless sensor networks (WSN) consist of micro-sensors capable of monitoring physical and environmental factors. These are often mainly made up of resource constrained sensor nodes and gateways. Networking these nodes presents several challenges because the devices have limited computational capability, data storage, energy and communication bandwidth. Therefore, various lightweight communication protocols are emerging for Machine to Machine (M2M) communications. Among the various application layer protocols for data communication in WSNs, the two most popular protocols for constrained devices are the Message Queue Telemetry Transport Protocol (MQTT) with a variant for sensor nodes (MQTT-SN) and the Constrained Application Protocol (CoAP). Studies have shown that the performance of these different protocols are dependent on different network conditions. CoAP is more efficient in terms of message overhead while MQTT-SN is more efficient in terms of client complexity. Studies have further emphasized the levels of difficulties implementing any of these protocols regarding application requirements. This project proposes an integration of MQTT-CoAP protocols using an abstraction layer that enables both MQTT-SN and CoAP protocols to be used in a sensor node. The performance of the system was evaluated in terms of latency per message size in bytes transmitted for different quality of service (QoS) levels and energy consumption per node. The result of the study showed that latency values slightly increase as the packet size increased. The lowest latency was observed in MQTT-SN QoS 0 while similar latency values were obtained for the CoAP and MQTT-SN QoS 1. The average latency was observed to be 163.2ms, 188.5ms and 191.5ms for MQTT-SN QoS 0, MQTT-SN QoS 1 and CoAP respectively. Energy consumption of the node when using MQTT-SN for a single Tx/Rx operation in a 10s interval showed an average of 261.6mJ for both QoS 0 and QoS 1 while an average of 261.3mJ was observed for CoAP. Performance evaluation of these protocols when integrated shows that the system is feasible. While CoAP performs better in terms of energy consumption, the two protocols perform almost equally in latency. The observed values of latency and energy consumption in the developed integration technique is comparable to other studies. This work has shown that the two protocols can coexist in a single sensor node without impacting negatively on its performance. Future work will be required to test the integrated system in a more complex network conditions.

TABLE OF CONTENTS

CONTENTS **PAGE**

LIST OF TABLES

LIST OF FIGURES

Figure **Page**

CHAPTER ONE

1.0 INTRODUCTION

1.1 Background of the study

In recent years, development of smart sensors has caused interesting advancements in the development of wireless sensor networks (WSNs). Smart sensor nodes are low power devices equipped with one or more sensors, possibly with an actuator, a processor unit, memory/storage unit, a power supply and a wireless communication radio (Akyildiz *et al*., 2002).

Wireless sensor networks (WSNs) consist of small sensors that can monitor environmental and physical factors such as seismic events, motions, vibrations, humidity, temperature. These have been applied in different areas, such as, but not limited to, industrial process monitoring and control (Peng *et al*., 2006), environment observation and habitat monitoring (Chen & Liao, 2011), healthcare applications (Alemdar & Ersoy, 2010; Kulkarni & Ozturk, 2011), home automation (Ding *et al*., 2011; Kaiwen *et al*., 2017), traffic control (Xu & Lee, 2007) and forecast systems (Selavo *et al*., 2007).

Research in WSN technology in resource constrained devices and using the Internet Protocol (IP) has greatly changed the Internet landscape. This showed that in the near future, trillions of smart objects would be connected to the Internet to form the Internet of Things (IoT) (Colitti *et al*., 2011). Internet of Things (IoT) is a prominent technology used in today's era for the establishment of the WSNs for anywhere and anyplace communication between sensor nodes (Sharma *et al*., 2020). The main aim of IoT is to create an interconnected network or infrastructure of devices that integrate all the current technologies (Gluhak *et al*., 2011). By the introduction of IoT, everything can be connected to the internet. This enables ubiquitous communication and the data sensed for communication can be physical world control signal, a device or regular internet data communication (Mohan *et al*., 2015).

Wireless sensor networks (WSNs) are made up of sensor nodes which are small, inexpensive, and intelligent (Yick *et al*., 2008) thanks to the significant advancements in the development of Micro Electrical Mechanical Systems (MEMS) development. Networking these nodes presents several challenges because the devices have limited computational capability, data storage, energy and communication bandwidth. This implies that, WSNs would currently not be able to adequately meet the needs of the IoT unless these challenges are resolved appropriately.

As the Internet of Things (IoT) expands to numerous applications through the availability of different sensors, increasing minimization of hardware, and "smart objects" (Giusto *et al*., 2010), many potential protocols are emerging for machine to machine (M2M) communications thus, the question of which protocol is most appropriate to use for the Internet of Things (IoT) becomes a very interesting topic for research.

The most important IoT protocols are divided into three main scategories: application layer protocols, infrastructure protocols and service discovery protocols (Martí *et al*., 2019). The most relevant infrastructure protocols for resource constrained devices in Lossy Networks are the Routing Protocol for Low-Power and Lossy Networks (RPL), considered the routing layer standard for IoT (Iova *et al*., 2016), 6LowPAN (Shelby & Bormann, 2009) for the network layer, and IEEE 802.15.4 (Molisch *et al*., 2006), LTE-A (Ghosh *et al*., 2010), EPCglobal (Kürschner *et al*., 2008), and Z-Wave (Gomez & Paradells, 2010) for the data link and physical layers. IETF Routing over Lows-power and Lossy networks (ROLL) Working Group has specified and designed a new IP routing protocol for smart object internetworking in addition to 6LowPAN. The protocol is called IPv6 Routing Protocol for Low-power and Lossy networks (RPL) (Vasseur &

Dunkels, 2010). The most widely used service discovery protocols are DNS Service Discovery (DNS-SD) and multicast DNS (mDNS), which are used for discovering resources and services offered by IoT devices (Al-Fuqaha *et al*., 2015). In the application layer, several popular protocols available today are CoAP (Constrained Application Protocol) (Shelby, 2013), MQTT (Message Queuing Telemetry Transport) (MQTT, 2014), DDS (Data Distribution Service) (DDS, 2015), AMQP (Advanced Message Queueing Protocol) (OASIS, 2012) and XMPP (eXtensible Messaging and Presence Protocol) (Saint-Andre, 2011).

One of the most significant advantages of IP based networking in Low-Power and Lossy Networks (LLNs) is that typical web service architectures can be used without the necessity of application gateways. This has made it possible for smart objects to not only be connected to the internet but to be connected with the Web. This connection is defined as the Web of Things (WoT) where smart object applications are built on Representational State Transfer (REST) architectures. REST architectures allow applications to depend on services that are loosely coupled which can be reused and shared (Colitti *et al*., 2011).

From an end-to-end view, a Wireless Sensor Network (WSN) can be viewed as comprising of two subnets; a subnet connecting one or more gateway nodes to sensor nodes in which sensor nodes route data until it reaches one of the gateways using WSN protocols (Gnawali *et al*., 2009), and another subnet connecting the backend server or broker and the gateway. Sensor data generated by sensor nodes are delivered through the gateway to the server. Meanwhile, clients that are interested to receive sensor data connect to the server to obtain the data. To transfer all the sensor relevant data collected by a gateway node to a server, the gateway requires a protocol that is energy-efficient, bandwidth-efficient and can work with really limited resources in terms of hardware capacity. For this purpose, protocols like Message Queue Telemetry Transport (MQTT) (MQTT,

2014) and Constrained Application Protocol (CoAP) (Shelby, 2013) have been developed to specifically address the stringent requirements of real-world WSN deployment outcomes.

One way data transfer is handled between gateway and clients in WSNs is the “request-response” also known as “client-server” architecture which is supported by CoAP. In the client-server architecture, request messages begin a transaction with a server, which may send a response to the client identified by a matching transaction ID since it uses UDP and this is based on a polling method (Davis *et al*., 2013). CoAP was designed on a web transfer protocol based on REST (Representational State Transfer) on top of HTTP (Hypertext Transfer Protocol) functionalities (Al-Fuqaha *et al*., 2015). CoAP works using UDP unlike HTTP, removing all the overhead TCP incurs, which makes it simpler, reduces bandwidth requirements, and makes it a good option for IoT applications. CoAP uses a client/server or request/response architecture like HTTP. As a result, it uses same methods as HTTP: GET, POST, PUT, and DELETE (Richardson & Ruby, 2007). CoAP also supports unicast as well as multicast (Martí *et al*., 2019).

Another way data transfer is handled in WSNs is the “publish-subscribe” architecture (Eugster *et al*., 2003). In this architecture, a client needing data (known as subscriber) registers its interests with a server (also known as broker). The client producing data (known as publisher) sends the data to a server and this server forwards the fresh data to the subscriber. One of the most significant advantages of this design is the separation of data-sending and data-receiving clients, i.e., sensor nodes do not need to know the network address or location or any other information regarding clients that are interested in their data and conversely, clients do not need to know about the sensor nodes generating the sensor data. This decoupling enables the architecture to be highly scalable (Eugster *et al*., 2003). The “publish-subscribe” architecture is supported by MQTT and CoAP (Davis *et al*., 2013; Thangavel *et al*., 2014). The publish-subscribe architecture was designed because there was a need to provide loosely coupled, asynchronous and distributed communication between data generators and destinations. This solution is seen today in the form of numerous publish-subscribe Message-Oriented Middleware (MoM) (Jia *et al*., 2014) and recently has been a subject of many research efforts (Chelloug & El-Zawawy, 2018; Hakiri *et al*., 2017; Veeramanikandan & Sankaranarayanan, 2017).

In this study a variant of the MQTT protocol designed for very resource constrained nodes was used which is the MQTT for Sensor Nodes (MQTT-SN). While the original MQTT works over TCP and TLS, MQTT-SN was designed to work in Wireless Sensor Networks (WSNs) over UDP (Lesjak *et al*., 2015). This protocol was designed to operate and function very much like MQTT. This means that because it provides the same semantics as MQTT, it can function with the same infrastructure. The sole architectural difference is that MQTT-SN needs a new system entity, known as a gateway, which has to convert all MQTT-SN messages sent over UDP to MQTT messages sent over TCP. Because of the fact that many brokers currently have this capability built in, all the complexity resides at the broker/gateway side, making the client side a lot simpler (Martí *et al*., 2019).

Studies have shown that the performance of these different protocols are dependent on different network conditions. MQTT messages had shorter delays than CoAP when there was less packet loss and larger delays when there was more packet loss. (Chen & Kunz, 2016; Thangavel *et al*., 2014). In addition, the publisher-subscriber architecture using the broker mechanism of the MQTT protocol offers two major advantages over the CoAP technique and these are; publishers and subscribers do not need to know about each others existence or presence; to share information, subscriber and publisher do not need to be online/active at the same time, because the broker is capable of storing data for clients that are not connected at the time a message is published (Sachs *et al*., 2010).

As a result of the fact that the disadvantages offered by one protocol is complemented by the other protocol and vice-versa, this study proposes an integration of MQTT and CoAP protocols that bring the advantages of both protocols to be utilized in data communication.

1.2 Research Motivation

Research has revealed that the two most predominantly used communication architectures in internet of things are the request/response architecture implemented in CoAP and the publish/subscribe architecture implemented in MQTT. Research has also shown that MQTT-SN and CoAP are the two most predominantly used application layer communication protocols for resource constrained wireless sensor nodes. Because of the design and architecture of each of these two protocols, there are certain network conditions and certain network designs that each protocol is better suited for. In other words, if the sensor nodes are programmed to work with the MQTT-SN protocol because of the network design/conditions, any change to the design/condition that necessitates the need for CoAP protocol to be used would require the sensor nodes to be reprogrammed. Furthermore, this would constitute a major roadblock to dynamic application layer communication protocol switching in resource constrained sensor nodes.

The effort to solve problems that arise from stereotyped application layer architecture for communication in WSNs is the main motivation for this research. There is no prior research that attempts to implement an integration of the MQTT-CoAP protocols in resource constrained devices.

1.3 Statement of the Research Problem

There are often expensive tradeoffs when either CoAP or MQTT Protocols are used individually and switching them would often require re-programming the node which is difficult in real life scenario. Identifying the problems that could arise from using either protocol can only be adequately realized after deployment. For better communication efficiency, the engineer would be required to retrieve all the nodes and reprogram them which would be an extremely tedious task depending on the size of the wireless sensor network.

1.4 Aim and objectives of the study

The aim of this work is to integrate the MQTT and CoAP protocols for data communication in wireless sensor networks. To achieve the aim, the objectives of this research are to:

i. Develop an abstraction layer that manages both MQTT and CoAP protocols thereby integrating both protocols in a system

ii. Test the integrated protocols by simulating the protocol and measuring Energy consumption and Latency

iii. Evaluate the performance of each protocol in the integrated system and compare with the performance of MQTT and CoAP protocols for diverse QoS requirements.

1.5 Justification of the Study

Various studies have highlighted the advantages, disadvantages and the performance evaluation of the MQTT and CoAP protocols. These show that some disadvantages of MQTT protocol is addressed in the CoAP protocol and some disadvantages of the CoAP protocol is addressed in the MQTT protocol. For example MQTT messages had shorter delays than CoAP when there was less packet loss and larger delays when there was more packet loss (Thangavel *et al.*, 2014). CoAP uses RESTful (Representational State Transfer) technique which provides flexibility and interoperability with other HTTP schemes. This enables a request pattern like GET, PUT, POST request for accessing or executing a specific function on the Node while the MQTT does not support RESTful methods (Dizdarevic *et al*., 2019). The REST technique would greatly simplify node specific configurations as opposed to setting up a subscription topic on the broker for node specific configurations in the MQTT protocol. The complementary advantages and disadvantages of the MQTT and CoAP protocol justifies the need for a study on integrating both protocols and utilizing them for data communication.

1.6 Significance of the Study

This study focuses on creating a technique that uses both MQTT and CoAP protocols for data communication in WSNs by creating an abstraction layer that enables leveraging any or both protocols for communication. This can then be further improved to facilitate identifying the data transmission type, network architecture and utilizing the most appropriate protocol for communication.

1.7 Organization of the thesis

Chapter two contains the theoretical background and review of some published literature related to this work. The methods and materials used in achieving the research objectives are presented in chapter three. In chapter four, the performance evaluation of the two protocols were carried out, the results of the research work were presented and discussed. Chapter Five is the Conclusion and Recommendations for future research.

CHAPTER TWO

2.0 LITERATURE REVIEW

The Internet of Things (IoT) is gaining traction throughout the world where millions of different devices will be interconnected, consuming and providing information accessible on the network. One of the most popular application of IoT is Smart city. Smart city includes transportation automation, smart surveillance, smart grid (smart energy management), smart environmental monitoring, smart water distribution and smart security etc. (Yun & Yuxin, 2010). The ongoing improvement in the development of the low cost and high performance hardware manufacturing, low cost mobile internet and high speed machine to machine (M2M) communication has led to the huge growth of IoT technologies. Internet of Things (IoT) devices will cause huge internet traffic. CISCO predicted that Machine 2 Machine (M2M) traffic would increase from 5 billion in 2015 to 12 billion by 2020 (CISCO, 2016). This increase in traffic has put load on existing hardware and network infrastructure therefore a suitable protocol can ease the burden on existing computing and network infrastructure (Tandale *et al*., 2017).

2.1 Wireless Sensor Networks (WSNs) and Internet of Things (IoT)

The Internet of Things (IoT) have been defined in several ways by different researchers. Pena-Lopez *et al*. (2005) defines the Internet of Things as a concept in which networking and computing capabilities are embedded in any kind of conceivable object. These capabilities are used to query the device status and if possible to alter it. Another definition by Vermesan *et al*. (2011) defines the Internet of Things (IoT) as a simple interface between the physical and digital worlds. Using a variety of actuators and sensors, the digital world interacts with the physical world. In common parlance, the Internet of Things (IoT) is a term used to define a new kind of world or an era where almost all appliances and equipment that we use are connected to a network. These devices can then be used all together to accomplish difficult tasks that require much intelligence (Sethi & Sarangi, 2017). For this interconnection and intelligence, IoT devices are developed and designed to have embedded transceivers, processors, sensors and actuators. IoT technology is not a singular; rather it is an accumulation of many technologies working together at the same time.

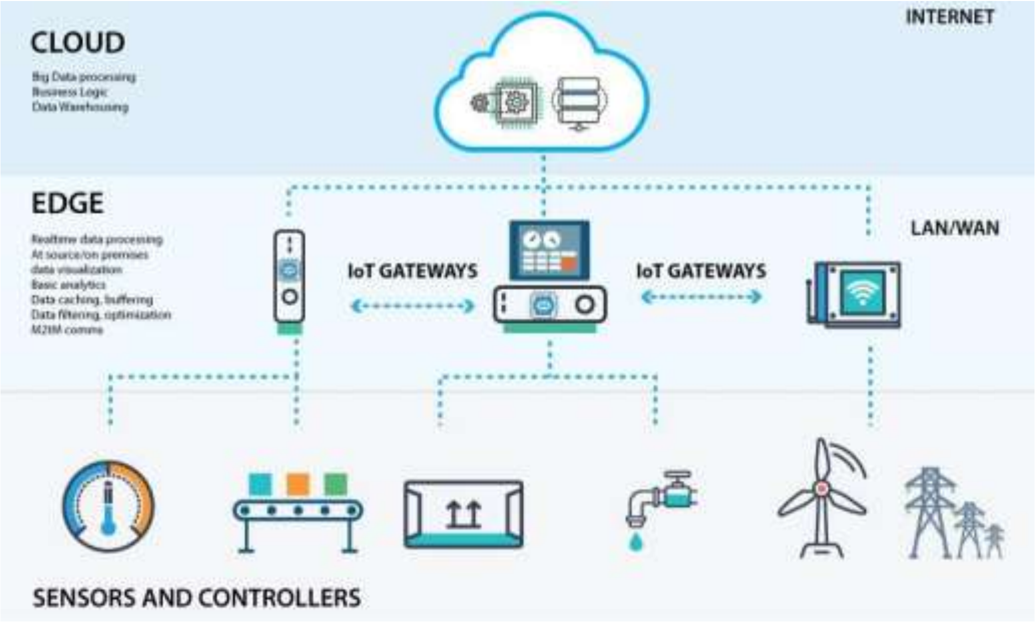


Figure 2.1: IoT communication model through gateway (Open Automation Software, 2021)

Wireless Sensor Networks (WSNs) have gotten a lot of interest because of their various potential uses and benefits. They constitute a prospective area where IoT may be incorporated to improve efficiency. When a wireless sensor Network is connected to the internet, often done through a gateway or sink node, it can also be referred to as internet of things (IoT). A technique used to connect WSN to the Internet is through a single gateway or multiple gateways in the case of a hybrid network (Christin *et al*., 2009). In situations where low latency is a critical issue, single hop internet connection can be employed. In Most cases, WSN is organized in a star topology where a central gateway can be utilized. The gateway is directly connected to each sensor nodes and also connected to the server autonomously. GPRS or LTE and Internet can be used for the out of field communication from gateway to the server. The information processed and stored in the gateway is sent to the server periodically (Viswanathan *et al*., 2018). Figure *2.1* shows the IoT communication model with the flow of information and some functions of the gateway and the server/cloud.

Wireless Sensor Networks (WSNs) often consist of sensors, actuators and gateways. Actuators and sensors are devices, that help us interact with the physical environment. In order for us to derive useful inferences from the data collected by sensors, they must be stored and processed intelligently. Note that the term sensor was broadly defined; a refrigerator, television or even a mobile phone can act/count as a sensor if it provides inputs about its current state (internal state and/or environment). A device that can be used to cause some change in the environment is referred to as an actuator e.g. the light intensity or temperature controller in smart homes. The Figure *2.2* shows a typical communication model for WSN between nodes, between a node and the gateway and between the gateway and the internet or a remote data processing and monitoring server.

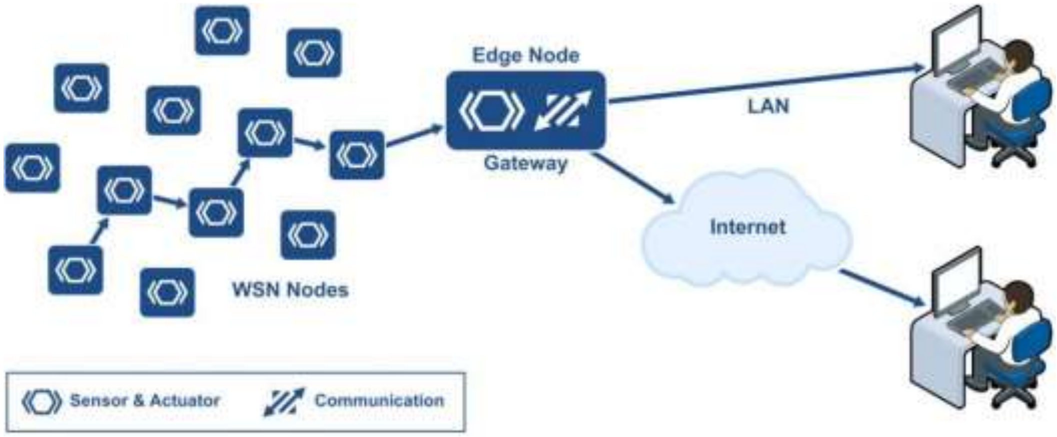


Figure 2.2: WSN communication model from Node to Node (SATHYABAMA, 2021)

Data storage and processing can be done on the edge of the network or on a remote server as shown in Figure 2.2. Possible data preprocessing is typically done on the sensor or any other reachable device and then sent to a remote server. The processing and storage functions of an IoT object are also limited by the resources available, which are often very constrained due to limitations of energy, size and computational capability. As a result, one major focus in research is to ensure that with respect to some predefined accuracy level, the right data is gotten. Another challenge is communication aside the challenges of data handling and collection. The connection between IoT devices is usually wireless because they are typically deployed at geographically distant locations. The wireless channels are unreliable and often frequently distorted. In this setting, communicating data reliably with minimal retransmissions is a problem and thus an important research interest in IoT are the communication technologies employed.

Now, after working on the data received, some actions, diverse in nature, needs to be taken based on certain assumptions. We can change the state the physical world using actuators or we may do something virtually like sending some data to other smart things. Context awareness is when the state of the physical world or a system at any point in time affects the process of effecting a change in that system. Since it is possible for an application to act differently in different contexts, the context is seriously considered before any action is taken. For example, a person may not like messages sent to him from his office when on vacation to interrupt him.

Communication network, compute servers, Sensors and actuators form the core system of an IoT framework. However, there are several software components that need to be put into consideration. A lot of standardization is needed to connect different hardware systems to avoid a lot of disparity in IoT devices operation from different vendors. We also need a middleware that can be used to link and handle all these diverse components.

The Internet of Things (IoT) is largely applied in education, entertainment, health care, fitness, energy conservation, social life, transport, environment monitoring, and home automation. It is evident that, in all these areas of application, IoT technologies have been able to significantly improve the quality of living and reduce human effort (Sethi & Sarangi, 2017).

2.2 Architectures of IoT

There is no universally accepted or agreed upon architecture for IoT. Different architectures have been proposed by different researchers. The most commonly seen architectures in literature is the Three-Layer and Five-Layer architectures as shown in Figure 2.3. The most basic architecture is a three-layer architecture (Mashal *et al*., 2015; Said & Masud, 2013; Yun & Yuxin, 2010).

The three-layer architecture was introduced much earlier in IoT architecture research. It has three layers, namely, the perception, network, and application layers. The five-layer architecture has the following layers namely, business, application, processing, transport and perception layers (see Figure *2.3*). The perception layer plays a similar role in the three layer architecture as it does in the five layer architecture. The main idea of the Internet of Things (IoT) is defined in the three-layer architecture but this is not enough for research in IoT because research often focuses on finer aspects of the IoT.

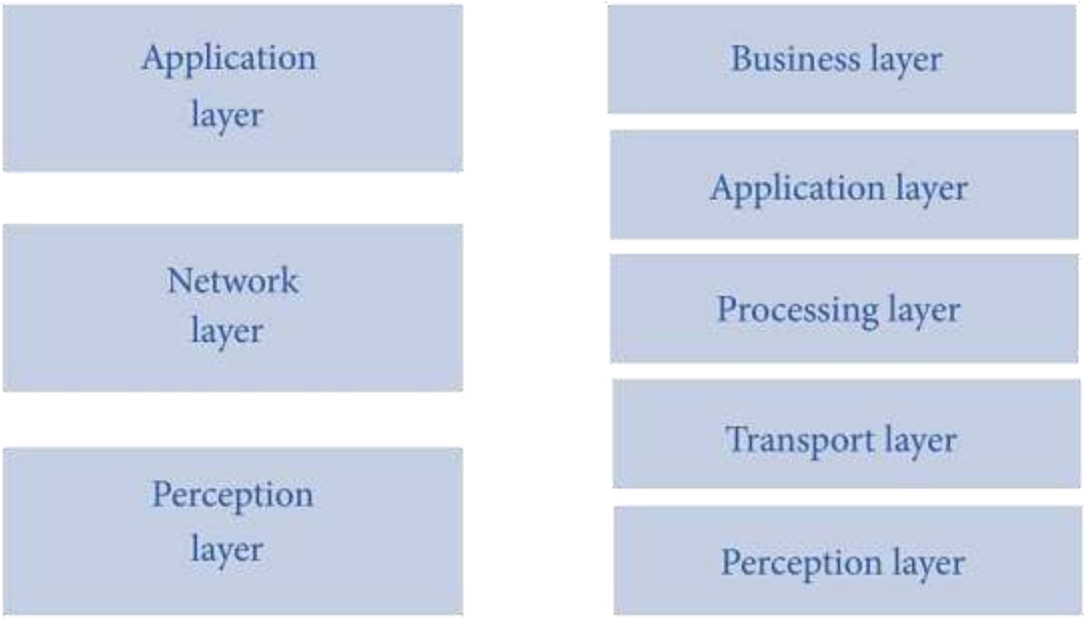


Figure 2.3: Architecture of IoT (Left: Three layers) (Right: Five layers)

Data sensing from the physical environment or system is handled in the perception layer also referred to as the physical layer. It senses some physical parameters or identifies other smart objects in the environment. Network connection to other network devices, gateways, servers or smart things is carried out in the network layer. Its responsible for processing and transmitting sensor data. The application layer defines the applications in which the IoT is deployed. It is responsible for delivering application specific services to the user. The transport layer transfers the sensor data from the processing layer to the perception layer and vice versa through networks such as wireless, LAN, 3G, NFC, RFID and Bluetooth. The processing layer stores, analyzes, and processes huge amounts of data that comes from the transport layer. It is also known as the middleware layer. It can manage and provide a diverse set of services to the lower layers. It employs many technologies such as cloud computing, databases and big data processing modules.

Then the business layer is in charge of overseeing the entire IoT system, including applications, business and profit models, and users' privacy.

Ning and Wang (2011) proposed another model or system architecture that is based on the layers of processing in the human brain and the ability of human beings to feel, have thoughts, remember, decide on things, and react to their physical environment. This is made up of the human brain, spinal cord and the network of nerves. The human brain is analogous to the data processing and management unit or the data center. The spinal cord is analogous to the distributed network of smart gateways and various data processing nodes. The network of nerves refers to the networking components and sensors.

There has recently been a shift towards another system model referred to as fog computing (Bonomi *et al.*, 2014; Stojmenovic & Wen, 2014) where the sensors and network gateways in the local WSN do a part of the data processing and data analysis. A fog architecture (Aazam & Huh, 2014) presents a layered approach as shown in Figure 2.4, which inserts security, storage, preprocessing and monitoring layers between the transport and physical layers. The monitoring layer monitors services, resources, power and responses. The preprocessing layer is responsible for performing processing, filtering and analytics of sensor data for storage. The temporary storage layer stores, replicates and distributes processed data. Finally, the security layer ensures data privacy and integrity by performing encryption/decryption of data. Before sending the data from sensors to the cloud, monitoring and information preprocessing are done on the edge of the network.

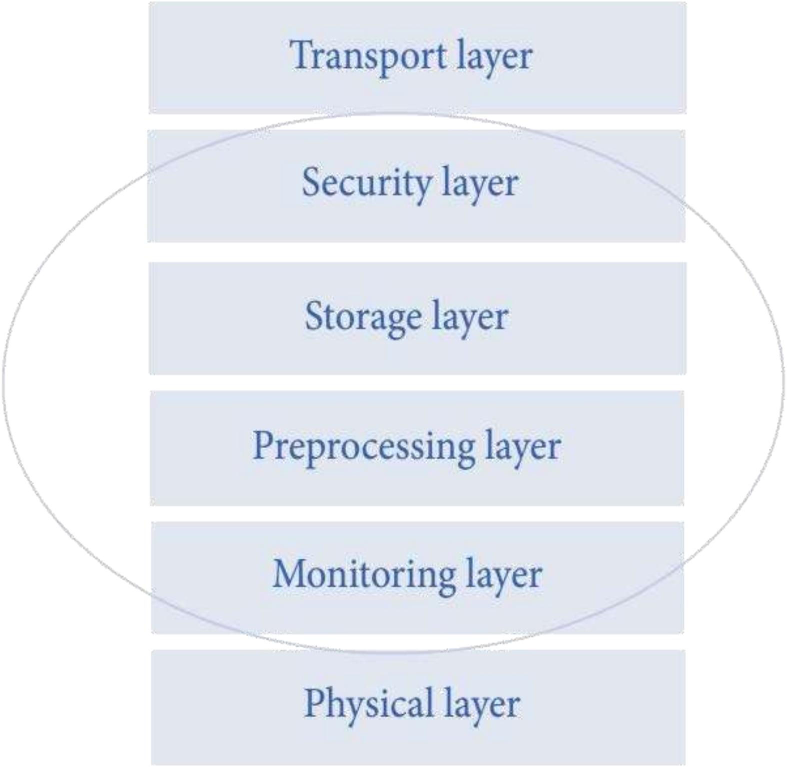


Figure 2.4: Fog architecture for IoT gateway

Often the terms “edge computing” and “fog computing” are used interchangeably. The former term predates the later and is construe to be a more general terminology. Fog computing initially defined by Cisco refers to smart sensors and smart gateways, whereas edge computing is slightly more penetrative and inclusive in nature. This paradigm is aimed at making physical devices like lights, pumps or motors able to preprocess information they collect. The purpose of this is to handle as much of information preprocessing as possible in these devices, which are referred to as network edge devices. The layered architectural diagram is not significantly different from Figure *2.4* when considering the system architecture. Thus, we do not describe edge computing as a separate entity.

2.3 Communication in WSN and IoT

The Internet Protocol (IP) stack is used to link IoT devices to the Internet. The IP stack is uses a lot of memory and power from connecting devices and its also very complex. As a result of the issues associated with the IP stack many WSN networks utilize sensors that connect locally through non-IP networks, which take up much less power, and connect to the Internet through a gateway. Fairly popular non-IP communication channels with limited range (up to a few meters) include Bluetooth, RFID, and NFC. Longer range Non-IP communication channels such as LoRa and SigFox are equally being widely used. The most widely used communication technologies in the IoT world are LoraWAN, SigFox, NFC, RFID, 6LoWPAN, low power WiFi, IEEE 802.15.4 and other exclusive protocols for wireless networks (Sethi & Sarangi, 2017).

2.4 IoT Network Stack

Notwithstanding the problems associated with IP communication in resource constrained devices, IP is a robust and efficient standard. This has led the Internet Engineering Task Force (IETF) to develop other protocols for IoT devices to communicate and share information IP (Sheng *et al*., 2013). Various white papers talking about other protocols and standards for the layers of the IP stack and an additional adaptation layer used for communication has been published by the Internet Protocol for Smart Objects (IPSO) Alliance (Sheng *et al*., 2013; Vasseur *et al*., 2011; Vasseur & Bertrand, 2010) between smart objects.

2.4.1 Physical and MAC Layer (IEEE 802.15.4)

The IEEE 802.15.4 protocol defines standards for the physical and MAC (link) Layer of the IP stack. It is designed to enable communication between inexpensive, compact and low power embedded devices that need to last long on battery life. It supports low cost, low power and short range communication. Low bandwidth, low transmit power and small frame size is needed in networks with resource constrained devices. In IEEE 802.15.4, transmitting data requires very little power (~ 1mW), which is just about one percent (1%) of that used in cellular or WiFi networks and this makes the communication range relatively shorter. As a result of the shorter range for communication, the devices have to operate together enabling multi-hop routing of data over longer distances. The data rate is limited to 250 kbps and the packet size is limited to 127 bytes only. Error detection in terms of losses in transmission is handled by the coding scheme in IEEE 802.15.4 which has redundancy built in. This makes the communication robust and causes the packets lost during transmission to be retransmitted. The protocol also supports short 16-bit link addresses to decrease the size of the header, communication overheads, and memory requirements (Hui & Culler, 2008). Vasseur *et al.* (2010) has carried out a survey that contains more data on the different physical and link layer technologies for communication between smart objects.

2.4.2 Adaptation Layer

The scalability and stability of IPv6 has made it to be considered the best IP addressing scheme in the IoT domain. IP protocols were initially not considered suitable for communication in IEEE 802.15.4 or any other low power wireless link due to their complexity and high resource usage. 6LoWPAN is a widely known standard for wireless communication using IPv6 over the IEEE 802.15.4 protocol. 6LoWPAN is an acronym for IPv6 over low power wireless personal area networks (Hui *et al.*, 2011). This standard defines an adaptation layer between the 802.15.4 transport layer and link layer. 6LoWPAN devices are capable of communicating with all other IP based devices on the Internet. IPv6 has a large addressing space available therefore it was chosen as the standard in 6LoWPAN. The internet space today has a majority of networks operating in the IPv4 standard therefore 6LoWPAN networks connect to the Internet via a gateway (Ethernet or WiFi) that supports protocol conversion between IPv4 and IPv6. IPv6 headers are not small enough to fit within the small 127 byte MTU of the IEEE 802.15.4 standard. Hence the adaptation layer performs an optimization that fragments the packets to carry only the most useful information. The adaptation layer specifically performs the following three optimizations for the purpose of reducing overhead in communication (Hui & Culler, 2008):

i. ***Header compression***: Some fields that can be gotten from the link level information are deleted since they can be shared across packets. This defines header compression of IPv6 packets for decreasing the overhead of IPv6.

ii. ***Fragmentation:*** IPv6 has a minimum MTU (maximum transmission unit) size of 1280 bytes. On the other hand, IEEE 802.15.4 has a maximum frame size of 127 bytes. Therefore the adaptation layer fragments the IPv6 packet.

iii. ***Link layer forwarding:*** 6LoWPAN also has the mesh under routing feature where packets are forwarded to destination over multiple radio hops. This is carried out at the link layer using link level short addresses and can be used within a 6LoWPAN network for communication.

2.4.3 Network Layer

The network layer primarily performs the task of routing packets received from the transport layer. The IETF Routing over Low Power and Lossy Networks (ROLL) working group has developed a routing protocol for Low Power and Lossy Networks (LLNs) referred to as RPL protocol (Vasseur *et al.*, 2011). RPL is an open routing protocol based on distance vectors where a set of constraints and an objective function is used to build a graph with the best path. This describes the process taken to build a destination oriented directed acyclic graph (DODAG) with the nodes after distance vectors are exchanged (Vasseur *et al.*, 2011). Design requirements determine the objective function and constraints and that’s why they can differ. For example, constraints can be to prefer encrypted links or avoid battery powered nodes. The objective function aims to minimize the number of packets expected that need to be sent or the latency. The creation of this graph begins from the root node. The root begins to transmit messages to nearby nodes. The neighboring nodes process the message received and depending on the objective function, they decide whether or not to join the tree. After that, the nodes forward the message they received to nearby nodes. The process continues in this way and the message keeps moving till they arrive the leaf nodes then a graph is formed. All nodes in the graph can now transmit packets to the root, hop by hop. The following is how we can implement a point-to-point routing algorithm. We transmit packets to a common ancestor, from whence they move downward (towards leaves) to arrive their destination.

2.4.4 Transport Layer.

Transmission Control Protocol (TCP) is not a viable option for transmitting data in low power and low data rate environments since it has a large overhead because it's a connection oriented protocol. Therefore, User Datagram Protocol (UDP) is the preferred option because it is a connectionless protocol and it has much lower overhead.

2.4.5 Application Layer.

The application layer handles the responsibility of data presentation and formatting. In the internet, the application layer is typically based on HTTP. However, HTTP is not an ideal consideration in resource constrained networks because it produces a large parsing overhead due to its verbose nature. IoT systems must be able to function in potentially low bandwidth, intermittent and unreliable connections because of the remote nature and need for wireless networking of smart objects for its access network. Many popular protocols are available today at the application layer and they are designed for Machine to Machine (M2M) communication. The most popular ones are MQTT (Message Queuing Telemetry Transport) (MQTT, 2014), CoAP (Constrained Application

Protocol) (Shelby, 2013), DDS (Data Distribution Service) (*Data Distribution Service (DDS)*, 2015), AMQP (Advanced Message Queueing Protocol) (OASIS, 2012) and XMPP (eXtensible Messaging and Presence Protocol) (Saint-Andre, 2011).

These communication protocols in general, are different in their models of interaction, i.e., publish-subscribe and request-reply. The request-reply model of communication is a basic communication paradigm that shows a message pattern most commonly used in client/server architectures. In this communication model, a client requests information from a server, the server receives the request message, processes it and returns a response message back to the client. This type of data is often maintained and transferred centrally.

The two most popular protocols based on the request/reply communication architecture are REST HTTP and CoAP. Figure 2.6 shows the different client/server interaction processes, for three HTTP versions (i.e., v.2.0, v.1.1 and v.1.0) and also for CoAP. After a single HTTP request/reply pair, the TCP is closed in HTTP 1.0. In HTTP 1.1, a TCP connection could be used repeatedly for sending many requests to the server without awaiting a response (pipelining) from the server via a keep-alive-mechanism. The client starts listening for responses immediately after all requests are sent. A server must send its responses to client requests in the same order that the requests were received as required by the HTTP 1.1 specification. The new HTTP 2.0 introduces a method for multiplexing by which a single TCP connection can be adequately used in sending multiple HTTP requests and can also be used to receive responses asynchronously. The fourth process shown in the figure is for CoAP. This makes use of UDP connectionless protocol and does not depend on the reliability of TCP connection for the client and server to exchange request/reply messages.

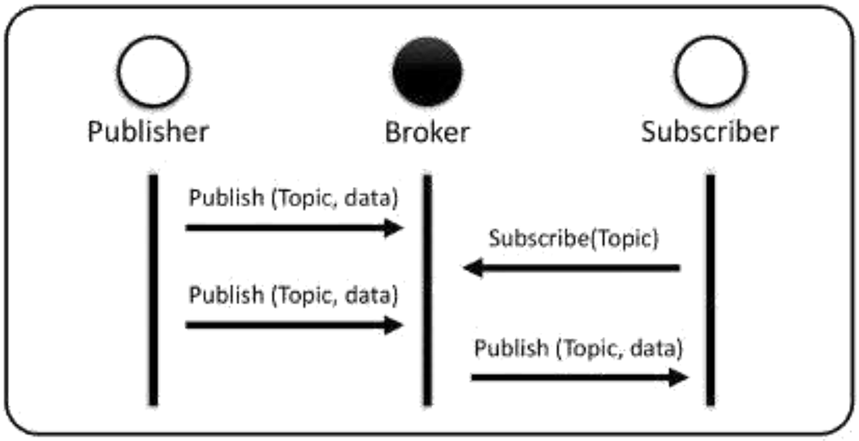


Figure 2.5: Publish-Subscribe architecture e.g. MQTT, DDS, AMQP (Dizdarevic *et al.* 2019)

In the publish-subscribe based protocols as shown in the Figure *2.5*, the client having a subscriber role does not make any direct request for information from the server or publisher in this scenario. Instead of making a request, the subscriber that wants to receive messages will subscribe to particular topics (events) within the system. The subscriber or client in this architecture subscribes to the broker. The central point in this system is the broker and it is responsible for routing messages between publishers and subscribers. The broker also filters incoming messages from publishers (Banavar *et al*., 1999). The publisher is the third party that functions as the information provider. When an event about a certain topic occurs, it publishes it to the broker who transmits the data on the desired topic to the subscriber. The publish-subscribe communication model can therefore be described as an event-based architecture for the previously stated reasons (Hinze *et al*., 2009).

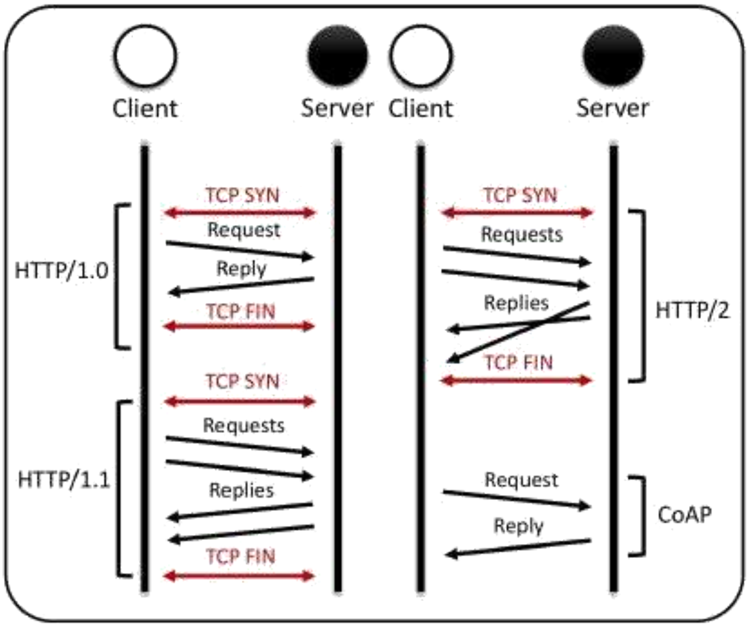


Figure 2.6: Request-Reply Protocols; HTTP and CoAP (Dizdarevic *et al.*, 2019)

2.4.5.1 Constrained Application Protocol (CoAP)

This protocol was developed for use in constrained devices having limited processing capability by the Constrained RESTful Environments (CoRE) working group of IETF (Shelby, 2013). Similar to HTTP it is based on the request/response architecture and one of its most prominent features is its use of tested and well accepted REST architecture. CoAP like HTTP supports the REST request/response architecture in constrained environments. CoAP is considered a lightweight protocol therefore the status codes, methods and headers are all binary encoded, thus reducing the protocol overhead when compared with many other protocols. Instead of transmission control protocol (TCP), CoAP runs over the less complex user datagram protocol (UDP) protocol, further reducing the overhead. CoAP request and response between a CoAP client and server are exchanged asynchronously over CoAP messages and not over some established connection. This makes CoAP less reliable in comparison to connection oriented protocols like HTTP. Since UDP features reduced reliability, IETF has recently created an additional standard document, highlighting the possibility of CoAP running over TCP (Bormann *et al*., 2018).

CoAP protocol architecture is divided into two logically different layers; the request/response layer and the message layer. The request/response layer, implements RESTful paradigm which allows for CoAP clients to communicate using HTTP-like methods when sending requests. This means that clients can use DELETE, POST, PUT or GET methods to manage the resources identified by URI (Universal Resource Indicator) in the network (Nguyen & Iacono, 2015) just like in HTTP. For instance in order to obtain sensor values, a client will make a GET request to the server URL and the server will reply with the requested data. The request and responses contain a token that is used to match them; the token defined in the request has to match the one present in the token. A client can also push data, for example updated sensor data, to a server device by using a POST request method to its URL. It is clear that in this layer, CoAP uses the same methods as REST HTTP for client/server communication. The second layer which is the message layer is what makes CoAP different from HTTP. CoAP relies on the message layer for reliability by retransmitting lost packets since the underlying protocol UDP does not ensure reliable connection. The message layer defines four types of messages: CON (Confirmable message), NON (non-confirmable message), ACK (Acknowledgement message), and RST (reset message). The CON messages are used for ensuring that communication is reliable by demanding an acknowledgement from the receiver side with an ACK message. QoS implementation in CoAP is dependent on this feature that marks whether the messages need the acknowledgement. This is a rather limited implementation.

An option known as the observe option can be added to the GET request in CoAP to improve the request/response model by allowing clients to continue to receive changes on a requested resource from the server (Correia *et al*., 2016). When the server receives this option, the server adds the client to a list known as the list of observers for the specific resource. When the resource state changes all clients on the observers list receive notifications. Just like in the publish-subscribe paradigm where the server alerts the client of changes, a similar technique is implemented in CoAP by setting the observe flag in the GET request. This eliminates the need for repetitive polling to check for changes in resource state. In an attempt to get even more like publish/subscribe paradigm and support nodes with long interruption in connectivity and/or uptime, IETF recently released the draft of Publish-Subscribe Broker that extends the capabilities of CoAP (Koster *et al*., 2019).

As a security mechanism, CoAP has a secure version called CoAPS which uses DTLS (Rescorla & Modadugu, 2012) instead of TLS on its UDP transport protocol with changes necessary to run over an unreliable connection. In cases of lost or out of order packets, DTLS is modified to stop connection termination. For example, there is a possibility of retransmitting handshake messages. The verification technique employed to make sure that messages from the client are authentic is that the server sends a verification query. This is an extension of the handshaking process in TLS where client and server exchange 'hello' messages. This mechanism of authenticity verification helps prevent Denial-of-Service attacks. Client and server also exchange supported cipher suits and keys through these messages and agree on the ones both sides support. This will further be used to protect data being exchanged during the process of communication.

New versions of DTLS have been developed and optimized for IoT and constrained devices since it was not originally designed for lightweight devices (Panwar & Kumar, 2015; Raza *et al*., 2013). Header compression technique in IPv6 over Low-power Wireless Personal Area Network (6LoWPAN) to compress DTLS header is one of the optimization mechanisms employed (Raza *et al*., 2012). There is ongoing research in optimizing DTLS for IoT due to its limitations (Granjal *et al*., 2015; Lakkundi & Singh, 2014).

2.4.5.2 Message Queue Telemetry Transport Protocol (MQTT)

MQTT is a lightweight messaging protocol released by IBM that is based on the publish-subscribe paradigm. This makes it suitable for network conditions with low bandwidth and high latency and also for resource constrained devices. The latest version used for IoT and specified by the OASIS (Cohn & Coppen, 2014) is MQTT v3.1. It has often been recommended as the choice communication solution in IoT because it is simple and has a very small message header compared to other messaging protocols.

Reliability is ensured in MQTT because it runs on the TCP transport protocol. MQTT is designed with a much lighter header compared to other reliable protocols such as HTTP. This has greatly reduced the power consumption requirements in devices making it a prominent solution in constrained environments and IoT. Clients and servers/brokers are the two communication parties in MQTT architecture that take the roles of subscribers and publishers. Clients are the devices that can publish messages, subscribe to message topics/events, or both. The publisher client must know about the broker that it connects to while the subscriber client needs to know the broker as well as the topic so as to receive relevant or corresponding information. When the broker receives new messages, all clients that are subscribed to the topic get updates from the broker. The broker does not only serve as the central component that accepts messages but also uses filtering to deliver the messages to the clients that subscribe to it.

MQTT broker library has to be installed on a device before it can have the role of a broker, for example Mosquito broker (Eclipse, 2019), which is one of the most used open source MQTT brokers. It should be noted that there are various other MQTT protocol brokers that implemented the protocol differently and are open for use. MQTT client libraries are used to drive the protocol in client devices/environments. There is a hierarchical organization of Topics in MQTT where strings separated by slashes indicate the topic level (Tantitharanukul *et al*., 2017). One MQTT publisher can publish messages to set of defined topics. For example, the client will publish to the topic: topic/1. This message will be published to the broker which can store it in a local database for a short time. While the message exists, if a subscriber interested in this topic sends a subscribe message to a broker specifying the same topic, the broker will send the message content of the topic to that subscriber. The broker will also send the message to other clients that subscribed to that topic.

MQTT defines three Quality of Service (QoS) levels, QoS 0, 1, and 2 (Cohn & Coppen, 2014; Luzuriaga *et al*., 2015). The QoS level can be defined both in the publish and the subscribe message body. QoS 0 refers to messages that deliver without any confirmation of received message. This QoS choice is considered in cases where sensor values do not significantly change over long periods of time e.g. when some sensors gather telemetry information. In such cases it is acceptable if some messages get missing since the general sensor value is still known due to the messages that were previously received. QoS 1 refers to messages that require confirmation of receipt to further assure the sender that the message was delivered. When a message with QoS 1 level is published, the receiver must send an acknowledgement packet back and if no acknowledgement is sent back after a defined period of time, the publisher will publish the message again. QoS 2 refers to messages that require confirmation but guarantees that the message will be delivered exactly once without being duplicated at the receiving end.

As the QoS level increases, the amount of resources needed to process MQTT packet increases. It is therefore paramount that the QoS level is adjusted to the specified network condition. MQTT offers another important feature which allows the broker to store published messages for new subscribers. This is achieved by setting a ‘retain’ flag in the message published. The broker normally discards a message if nobody is interested in the published topic. This means that in the default case, new subscribers would have to wait for another newly published state in order to receive a message about a topic. The broker is notified to store the published message when the ’retain’ flag is set to value: true, so it could be delivered to new subscribers.

MQTT runs on TCP transport protocol which is quite resource intensive for constrained devices. For this purpose, MQTT for Sensor Networks (MQTT-SN) has been proposed. This version uses UDP instead and it supports topic name indexing (Stanford-Clark & Truong, 2013a). This solution does not depend on TCP, but instead uses UDP as a more efficient, simpler and faster transport option over a wireless link (Govindan & Azad, 2015). The size of payloads reduces and this is an important improved feature. The reduction is achieved by numbering the packets with numeric topic ids instead of long topic names. Currently, the major disadvantage of MQTT-SN is that only a few platforms support it and there is only one free broker implementation called Really Small Message Broker (Xu *et al*., 2017).

From a security standpoint MQTT has an issue where messages are exchanged as plain text without any encryption mechanism because of its lightweight design. Therefore, encryption needs to be designed separately and integrated, for instance via TLS which would increase the message overhead. Many MQTT brokers implement authentication using one of MQTTs CONNECT control type message packets. Clients should define username/password combination when sending the CONNECT control message and this is used to validate the connection. Security is an important and continuous effort for MQTT (Lesjak *et al*., 2015), since MQTT is one of the most widely utilized communication protocol solutions. Improving the security issue would be a huge leap for MQTT, when compared with other solutions currently available.

2.5 Tools for simulation of WSNs

Simulators are necessary since cost of experimenting with real life WSN nodes over spatially distributed area which could consist of many nodes is expensive. To determine the most suitable simulator based on user requirements various criteria or evaluation parameters are considered; type of simulator or level of detail, license, platform or operating systems, WSN platforms, ease of code, graphical user interface (GUI) and energy consumption model. Considering the type of simulator or level of detail, there are three categories of WSN simulators namely generic, firmware level and code level simulators (Anand, 2015; Jevtić & Zogović, 2009).

Based on these criteria, four simulators could have been used which are ns-2, OMNeT++ based Castalia, TOSSIM and COOJA/MSPSim. A comparative study (Jevtić & Zogović, 2009) of these simulators show that ns-2 and OMNET++ are generic simulators while TOSSIM and COOJA/MSPSim are Code Level Simulators. Generic simulators focus on high-level aspects of WSN, such as data processing, sensing and networking while hardware architecture and operating system (OS) of sensor nodes are neglected. Code level simulators use the same code in a real node as in the simulator trying to emulate the behavior of the node completely. TOSSIM does not natively support graphical user interface (GUI) or Energy consumption modelling and has poor documentation for usage compared to COOJA/MSPSim. The COOJA/MSPSim is the simulator that was used in this research because it natively supports a GUI and also Energy consumption modelling.

2.5.1 Network simulator-2 (ns-2)

Network Simulator-2 (ns-2) (Varadhan, 2009) is one of the most widely used WSN simulators. It began as a general network simulator, and support for mobile ad-hoc wireless networks was added later (Mekni & Moulin, 2008). It is a generic, discrete event and object-oriented (OO) simulator, written in C++, with an Object oriented Tool Command language (OTcl) interpreter as a front-end (Varadhan, 2009). Its components, protocols, modes and simulation kernel are implemented in C++, but are also accessible from OTcl. In order to record simulation results, specify scenarios, set up network topology, configure simulator etc, OTcl scripts are used.

With respect to WSNs, ns-2 provides support for various protocols like IR-UWB, 802.15.4, 802.16, 802.11 etc. (Jevtić & Zogović, 2009). But even though there are lots of contributions from varied researchers around the world, ns-2 faces serious drawbacks in terms of WSN simulations (Singh *et al.*, 2008) like the non-existent sensing model. The parameters which are used during simulation of nodes in WSN like MAC protocols, packet formats and energy model are entirely different as it is seen and used in real world WSN scenario. Another issue regarding the use of ns-2 for simulation is that it lacks application support which is a requirement for sensor network interaction between protocol and application levels. NS-2 supports dual output which can be either graphical-based or text-based. For graphical based simulation NS-2 has an inbuilt tool called network animator (NAM) which shows live simulation scenario, node position, movement of packets in the nodes and also contains XGraphs which presents a graphical analysis of the results at the end of simulation.

2.5.2 Network simulator-3 (ns-3)

Network Simulator-3 (ns-3) is an open-source simulator and was launched in June 2008. The latest version is 3.21 released in August 2014 (Anand, 2015). NS-3 is a discrete-event simulator just line ns-2. It was developed to enhance research in communication networks.

Unlike ns-2 all the programs in ns-3 are written in pure C++ code with optional python bindings and like ns-2 there is still no built in graphical tool except with the Network Animator (NAM).

For simulating wired networks, ns-3 provides device model of a simple ethernet network which uses CSMA/CD as its protocol scheme with an exponentially increasing back-off for contention of the shared transmission medium. For WSN simulations, various modules have been integrated with ns-3 including modules like RPL, 6LoWPAN and 802.15.4.

2.5.3 OMNET++

OMNet++ (*OMNeT++*, n.d.) is another powerful discrete event object oriented network simulator for WSNs. OMNeT++ is currently in version 5.6.2 released in May, 2020 with primary focus on GUI upgrades. It is regarded as cosmponent based, modular and extensible C++ simulation library framework for building simulations for wireless networks. OMNeT++ is primarily not a simulator but it provides tools and frameworks for developing simulation scenarios.

It can run on MAC OSX, Linux and Windows operating systems is compatible to run on different operating systems like Windows, Linux and MAC OS X. OMNeT++ is free to download for research and educational purposes but the company has a licensed version called OMNEST.

OMNeT++ is basically made up of three modules; Simple, compound and network modules. The simple modules is the part written with C++, the compound module links other modules using connections and the network module is a top level compound module. It includes Eclipse IDE for debugging modules and editing NED (Network description) files.

The simulator that is primarily built for WSN in OMNeT++ is called CASTALIA.

2.5.4 TOSSIM

TOSSIM is a code level WSN simulator (Levis & Lee, 2003) and a part of the standard TinyOS (Levis *et al*., 2005) distribution. Replacing some low-level component with simulation implementation, TOSSIM is able to simulate entire TinyOS applications. Simulation events in TOSSIM represent posted tasks, high-level system events and hardware interrupts therefore it is regarded as a discrete event simulator (Levis & Lee, 2003). TOSSIM software modules and the TinyOS application are compiled and linked into a software library. The Python interpreter is used with this library to run simulation, configure simulator, define topology etc. An alternative C++ application that is linked to the library can be used instead of Python.

Modelling the wireless channel can be done by defining the propagation loss for each pair of nodes in the two directions and loss values can be gotten by applying a theoretical model or from real-world measurements. TOSSIM provides a tool for calculating loss values for a given topology using log-normal shadowing. It also offers different low level implementations that is capable of simulating the behaviour of a lot of real radios since it does not provide a specific PHY model. Simulation of CC2420 PHY is done by default. Interference and RF noise from outside sources and other nodes are also simulated. Noise trace is analyzed using Closest Pattern Matching (CPM) algorithm and a statistical model is derived from it. Interference and noise simulation is then gotten from this model (Levis & Lee, 2003).

The shortcomings of TOSSIM is that all simulated nodes run the same application code and there is lack of proper documentation. Energy consumption in TOSSIM is not natively supported but modeled through an add-on called PowerTOSSIM z (Perla *et al*., 2008).

2.5.5 COOJA/MSPSim

COOJA (Osterlind *et al*., 2006) and MSPSim (Eriksson *et al*., 2007) are WSN simulators included in Contiki (Dunkels *et al*., 2004) distribution. MSPSim is integrated into COOJA to form COOJA/MSPSim.

MSPSim is designed based on Texas Instruments MSP430 microcontroller and it is a WSN firmware level simulator. It is capable of more detailed debugging using single stepping, logging, watches and break points. For the purpose of determining power consumption, MSPSim provides various statistics like how much time a component spent in different operating modes. COOJA is primarily a code level simulator in networks with nodes that are running Contiki OS. In COOJA it is possible for nodes with different on-board software and different hardware to coexist and function as part of the simulation. In order for COOJA to achieve code level simulation, special simulation glue drivers, user processes and Contiki core are compiled into object code native to the simulator and then executed within the COOJA environment. All interaction done with the compiled Contiki code is done through JNI (Java Native Interface) because COOJA is a Java application. When user processes and Contiki core are compiled into target platform object code and executed in MSPSim, firmware level simulation is achieved. Generic WSN simulation is also handled in COOJA when non-Contiki nodes with Java functionality is simulated.

2.6 Review of Related Work

An extensive amount of research has been done on both MQTT and CoAP over the last decade. The most recent advances in this area will be presented in this section.

2.6.1 Review of related works on CoAP

The mode of operation and capabilities of CoAP has enabled it to be implemented in various domains ranging from health management systems to home automation systems. CoAP is specifically developed for constrained devices like microcontroller units (MCU) with less than 1kb of RAM (Tandale *et al.*, 2017).

Mi and Wei, (2019) worked on a smartphone proxy architecture using CoAP that is aimed at helping hospitals monitor patients remotely. The results they obtained through a real testbed on common smartphone with modified Bluetooth module, validated their proposed software architecture. This was done by showing that CoAP can be implemented to act as a data handler and communications proxy without much use of resources in terms of CPU, memory and battery.

Ugrenovic and Gardasevic, (2016) also performed a similar work on healthcare remote monitoring. Their solution was to provide patients health conditions through a web browser by implementing

CoAP on Mozilla Firefox browser. They first simulated their solution using Contiki OS and Cooja/MSPSim with 6LoWPAN protocol stack.

Ge *et al.* (2016) proposed the design and implementation of an IoT system for healthcare using CoAP standards and ISO/IEEE 11073 PHD (Personal Healthcare Device) in order to reduce the data loss between the devices and measured information while in transmission and enhance interoperability. To demonstrate their proposed architecture, they further implemented comparative performance evaluation between CoAP and HTTP in terms of syntax usage between XML and JSON, the number of packets by data loss rate during transmission and the number of the number of packets in one transaction.

Tariq *et al.* (2020) presented a survey on enhancements and challenges in CoAP. Their paper presented some applications where CoAP is implemented in different domains and it covers the enhancements made in congestion control mechanism as well as some application specific enhancements made over time in detail. A qualitative and quantitative analysis of enhanced congestion control mechanisms of CoAP is also highlighted in the survey. This analysis will make it easier for researchers to know which scheme is more appropriate to use depending on the requirements of the application.

2.6.2 Review of related works on MQTT

Liao and Lin, (2017) implemented an IPv6 over BLE experimental environment based on Raspberry Pi 3, and ran lightweight application layer protocols on it including MQTT-SN and MQTT. The sensor nodes running MQTT-SN were able to send the data to the broker and then the broker communicated the data to a web server they had developed to display in real-time.

Naik (2017) presented a more detailed and relative analysis of four messaging protocols for IoT namely HTTP, AMQP, CoAP and MQTT. The author analyzed the protocols using some associated criteria and was able to bring out the limitations and strengths of each messaging protocol. These messaging protocols are different from each other because different needs and processes gave birth to their development. Also, their relative and precise comparisons was dependent on the system requirements, specific conditions, applications, resources, devices and types of IoT systems. A similar in depth review of these IoT application layer protocols was done by Dizdarevic *et al.* (2019).

Venanzi *et al.* (2018) worked on the node discovery process in IoT fog environments by briefly presenting the application layer protocols in IoT. They also discussed MQTT-driven node discovery protocols (PEND and SPEND) and investigated the impact of the dynamicity of the advertiser nodes (BLE-A) on the sustainability of battery-powered IoT nodes and device discovery success.

Guha Roy *et al.* (2018) studied the MQTT-SN protocol with a new technique to publish sensor data called smart gateway selection technique. They proposed a mathematical model that helped to estimate end-to-end content delay and message loss. Their results were compared to other researchers (Davis *et al*., 2013; Piyare & Lee, 2013) who also proposed various models to reduce delivery time as message payload size and packet size increased.

2.6.3 Performance and comparative review of MQTT and CoAP

There have been many qualitative reviews of different communication protocols that could potentially be applicable to IoT networks (Asensio *et al*., 2014; Atzori *et al*., 2010; Sheng *et al*., 2013; Sutaria & Govindachari, 2013). For a proper comparative analysis, it important to know the major differences between the two protocols of interest in this research. This is captured in Table *2.1*.

Table 2.1 Major differences between the MQTT-SN and CoAP Protocols (Bandyopadhyay *et al.,* 2013)

	MQTT-SN	**CoAP**
Application Layer	Single Layer	Single layer with 2 conceptual sublayers (Message layer and Request Response Layer)
Reliability	3 Quality of Service (QoS) levels	Confirmable/Non-confirmable messages, Acknowledgments and retransmissions
Architecture	Publish/Subscribe	Request/Response, Resource/observe
Header Size	2 or 4 bytes	4 bytes

Thangavel *et al.* (2014) compared the performance of CoAP and MQTT under a common middleware. In their study they evaluated the effect of packet loss on latency in the different protocols as well as overhead vs. packet size. The experimentation showed that MQTT messages have lower delay than CoAP messages at lower packet loss rates and higher delay than CoAP messages at higher loss rates. Moreover, when the message size is small and the loss rate is equal to or less than 25%, CoAP generates lower additional traffic than MQTT to ensure message reliability. The influence of some other network conditions such as bandwidth cap and latency were not considered.

Bandyopadhyay *et al.* (2013) compared the performance of CoAP's resource-observe mode and request-response mode with MQTT in terms of overhead vs. various packet sizes among two different packet (0% and 20%) conditions. They also evaluated the relationship between power consumption and bytes of data communicated, although packet loss was considered as the only factor characterizing the network condition. The analysis of the experimental results depicts that CoAP is most efficient in terms of energy consumption as well as bandwidth. De Caro *et al*. (2013) compared the performance of CoAP and MQTT in terms of packet received ratio based on a 20% packet loss round trip time (for delay) and per-layer bandwidth usage. They utilized smartphones as sensing platforms in their study.

CHAPTER THREE

3.0 MATERIALS AND METHODS

3.1 Research Materials

The materials used in carrying out this research work are listed below:

i. Hardware materials

   a) HP Elitebook Laptop (Intel core i7, 16GB Ram, 512GB SSD) with Hyper-V enabled

ii. Software materials

   a) VMWare Workstation Pro v15.0

   b) Contiki OS

   c) COOJA/MSPSim Simulator

   d) Tunslip Utility

   e) Mosquitto RSMB (Really small message broker)

   f) Microsoft Office Excel 2016

The hardware specifications was chosen to be able to support the software that was used to carry out the research simulations. The unique feature of the computer hardware is the Microsoft Hyper-V being enabled and the larger random access memory (RAM) size (16GB). The Microsoft Hyper-V enables one to create virtual machines on Windows OS. This made it easy to run the VMWare Workstation tool which is a virtualization software and in it, the Contiki OS. The COOJA/MSPSim, Tunslip Utility and Mosquitto RSMB tools ran in the Contiki OS. The data was processed in the Microsoft Office Excel 2016.

3.2 Research Methodology

After thorough research and feasibility test of the proposed system was conducted, Figure *3.1* shows the steps taken to carry out the research work.

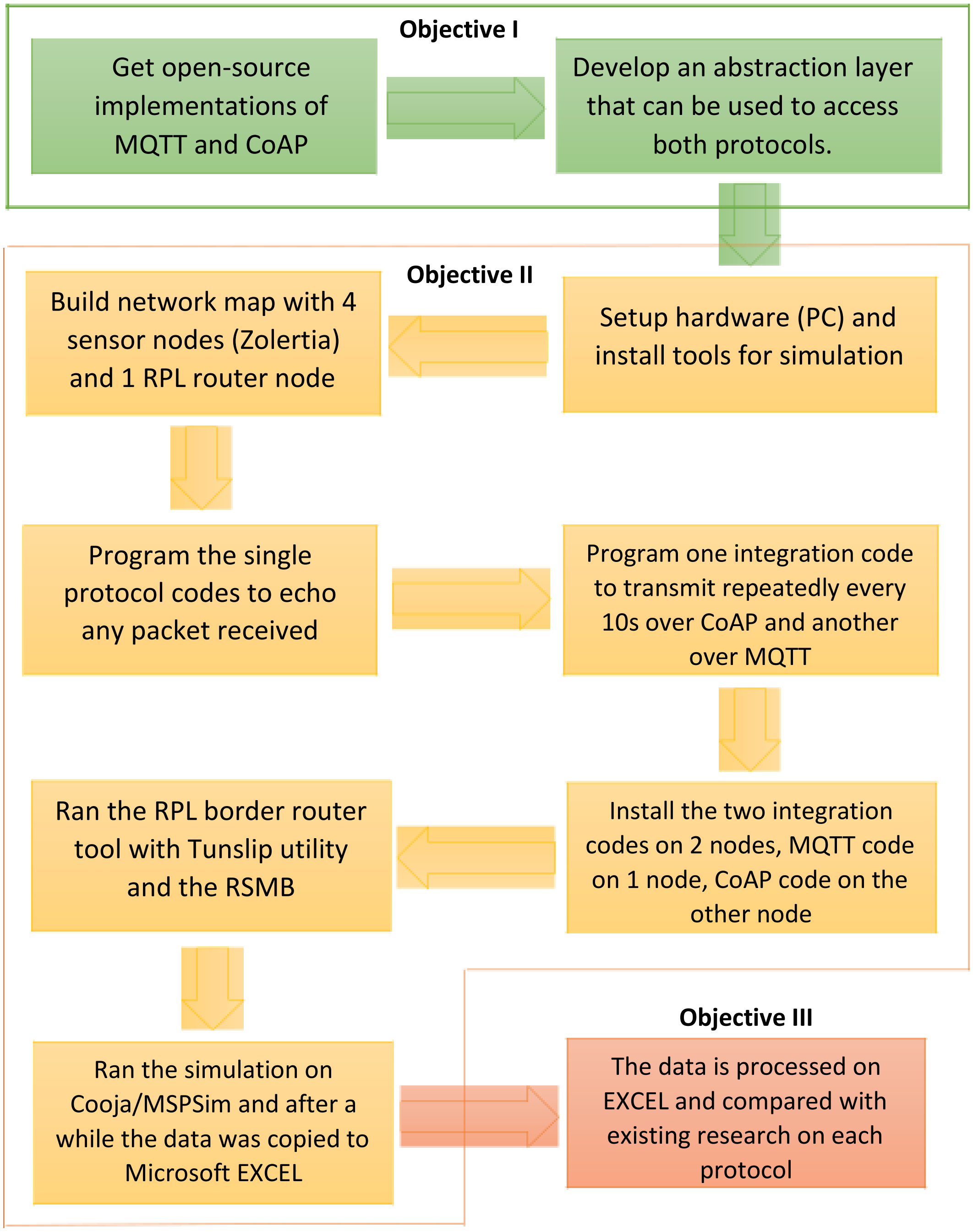

Objective I
Get open-source implementations of MQTT and CoAP
Develop an abstraction layer that can be used to access both protocols.
Objective II
Build network map with 4 sensor nodes (Zolertia) and 1 RPL router node
Setup hardware (PC) and install tools for simulation
Program the single protocol codes to echo any packet received
Program one integration code to transmit repeatedly every 10s over CoAP and another over MQTT
Ran the RPL border router tool with Tunslip utility and the RSMB
Install the two integration codes on 2 nodes, MQTT code on 1 node, CoAP code on the other node
Objective III
Ran the simulation on Cooja/MSPSim and after a while the data was copied to Microsoft EXCEL
The data is processed on EXCEL and compared with existing research on each protocol

Figure 3.1: Research procedure block diagram

3.3 MQTT-CoAP Integration Technique and Algorithm

Message Queue Telemetry Transport (MQTT) protocol is primarily designed to work using the publish/subscribe technique through a broker while Constrained Application Protocol (CoAP) is primarily designed to work using the request/response technique. The resource/observe technique of CoAP that is like the publish/subscribe technique of MQTT does not involve a broker and therefore is not as scalable as MQTT.

The technique of integrating both protocols carried out in this study is carried out such that both protocols are working in a single node and responding based on the communication priority or the node programming. Common communication priorities that were tested for the purpose of data transmission are energy consumption, latency and reliability.

The current design and implementation of the abstraction layer integrates the two protocols such that it exposes APIs that abstracts the complexity of managing the two protocols individually as well as APIs that are suited to advanced users who intend to access core functionality of any single protocol. Figure *3.2* is a block diagram that illustrates the abstraction layer.

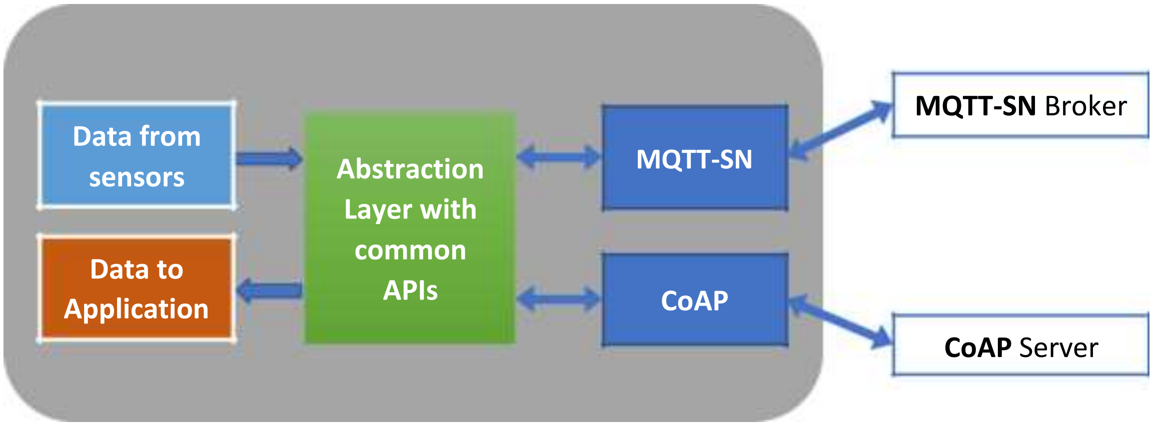


Figure 3.2 Block diagram of how the abstraction layer interfaces the protocols

With respect to the three or five layer architecture for IoT, the abstraction layer like the MQTT and CoAP protocols is implemented and resides within the application layer. The illustration of the three layer architecture with the position of the abstraction layer is shown in Figure 3.3.

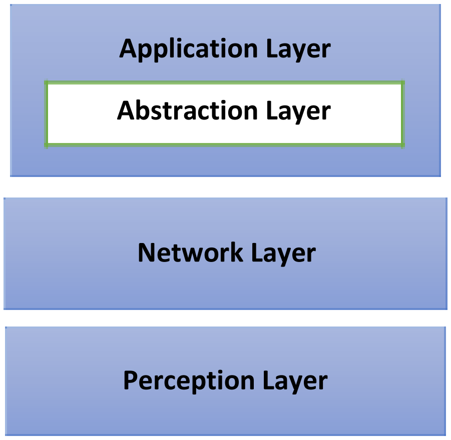


Figure 3.3: Three layer IoT architecture indicating the position of the Abstraction layer

The abstraction layer provides the following APIs; *Setup_Coap_Resource(), Init_Mqtt_Coap(), Mqtt_Sn_Pub(), Mqtt_Sn_Sub()* and *data_handler()*. The *Setup_Coap_Resource()* API is used to setup the CoAP resource method i.e. GET, POST, PUT or DELETE, endpoint URL and handler.

The *Init_MQTT_Coap()* API is used to initialize the CoAP resources that have been previously setup, create the MQTT connection to the broker and register any topic that would be used thereby converting them to numeric topics for MQTT-SN. Immediately after the MQTT-SN connects to the broker, it starts sending keep alive pings to the broker.

The *Mqtt_Sn_Sub()* is specific to the MQTT-SN and it takes two arguments; the topic and the QoS. It is responsible for subscribing to a topic on the broker. The *Mqtt_Sn_Pub()* is also specific to the MQTT-SN protocol and it takes four arguments; the topic, the message, the QoS and a retain flag. It publishes a message on the broker on a specified topic which should have been previously registered during the initialization stage. The retain flag just tells the broker whether to retain the message for new subscribers or to discard it. The APIs in the abstraction layer has been used to determine the latency of MQTT-SN and CoAP protocols when they are both used simultaneously in the same constrained device.

The *data_handler()* is used to process received payload like the application would like to. In our implementation, this contains logic to echo received data for latency computation. This can be specific to each protocol or used for both depending on the mode of initialization.

Figure *3.4* is the algorithm for the integrated protocols during data transmission. The integration is implemented via an abstraction layer making both protocols available at the same time. Before transmission, there is a block indicating the algorithm implemented to select preferred protocol.

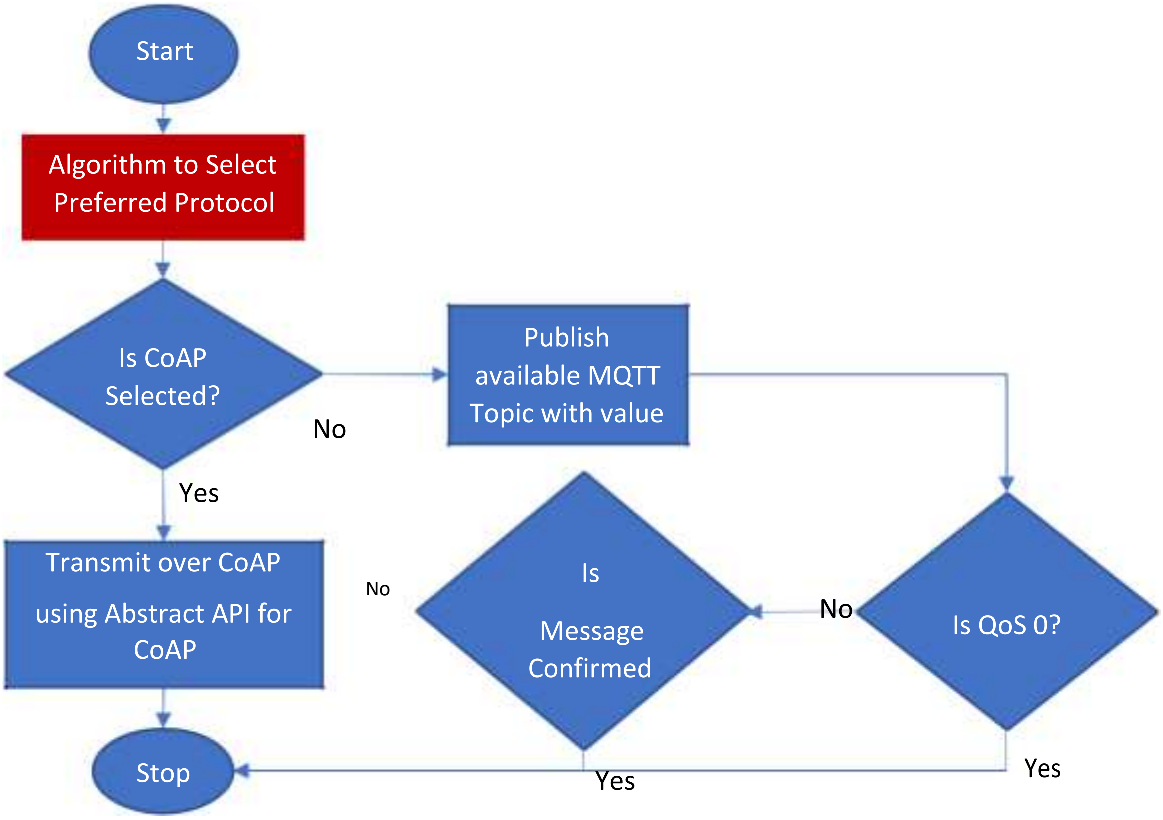

Start
Algorithm to Select Preferred Protocol
Is CoAP Selected?
No
Publish available MQTT Topic with value
Yes
Transmit over CoAP using Abstract API for CoAP
No
Is Message Confirmed
No
Is QoS 0?
Stop
Yes
Yes

Figure 3.4: Algorithm for design of integrated MQTT-CoAP protocol data transmission

This algorithm that selects the preferred protocol for data transmission is user dependent because there are different instances that could necessitate the need for one protocol or the other such as network conditions, presence or absence of a broker serving as the gateway, status of the RPL router for translating MQTT-SN packets etc.

Two user defined algorithms were tested in this study. One of the algorithms is for optimal latency or RTT and the other is for reliability. Figure 3.5 shows a user defined algorithm implemented to select preferred protocol for transmission based on packet RTT. For reliability being the priority, the most reliable is MQTT QoS 1 and that will be the selected protocol.

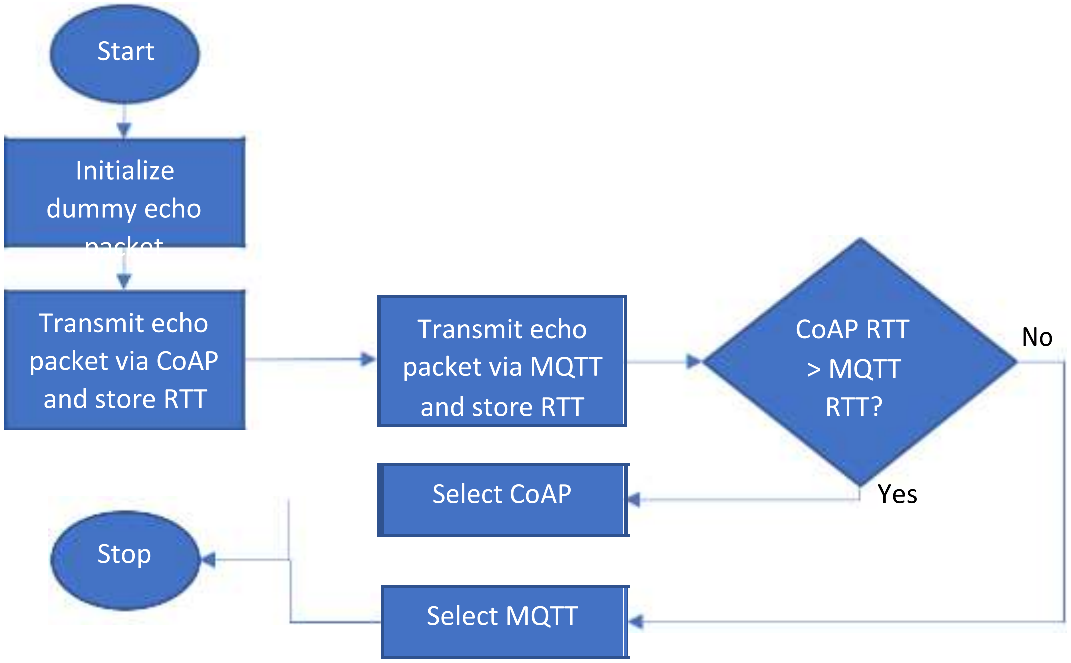

Start
Initialize dummy echo packet
Transmit echo packet via CoAP and store RTT
Transmit echo packet via MQTT and store RTT
CoAP RTT > MQTT RTT?
No
Yes
Select CoAP
Stop
Select MQTT

Figure 3.5: Algorithm for selecting preferred protocol based on RTT

In order to carry out a performance evaluation of the protocols individually when in the integrated system and compare to existing studies evaluated by other researchers, the algorithm was simply implemented to select one protocol at a time and was done twice. The first time CoAP was selected and its performance was evaluated and second time MQTT-SN was selected and its performance was evaluated.

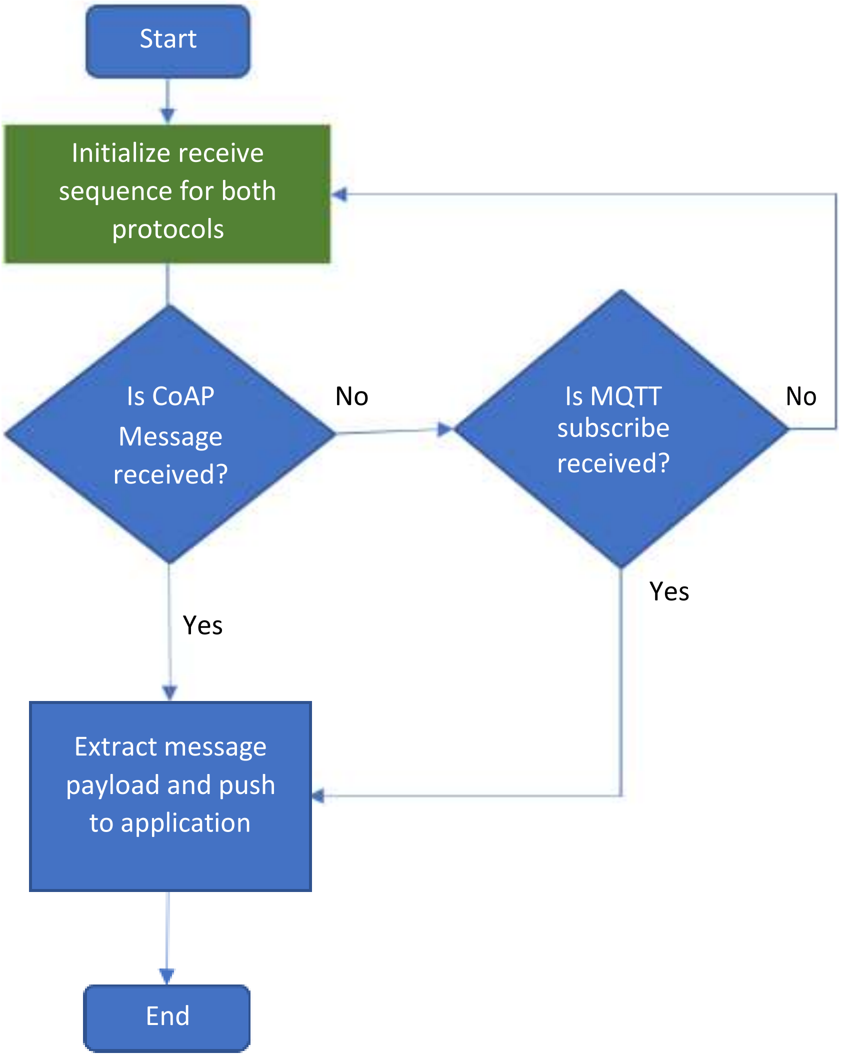

Start
Initialize receive sequence for both protocols
Is CoAP Message received?
No
Is MQTT subscribe received?
No
Yes
Yes
Extract message payload and push to application
End

Figure 3.6: Algorithm for integrated MQTT-CoAP protocols receiving data

Figure 3.6 shows the algorithm for the integrated protocols when receiving data. First of all, both protocols are initialized. This means that all the CoAP endpoints are registered and all the MQTT topics that will be listened for are registered. The algorithm was normally implemented to use the same endpoints in CoAP as topics being subscribed to in MQTT-SN.

When QoS 1 is used for MQTT-SN, if a PUBACK is not received after the MQTT packet is published, the same message is sent via CoAP when a default receiver address is chosen. Although there is no acknowledgement when a message is sent via CoAP but it is assumed that the broker is unresponsive or unreachable so a redundant technique is applied to still have the message transmitted.

Data forwarding is handled in the MAC layer as part of the Routing protocol for low power and lossy networks (RPL) which provides IPv6 connectivity for WSN. The radio duty cycling (RDC) protocol used was the nullRDC keeping the radio always active.

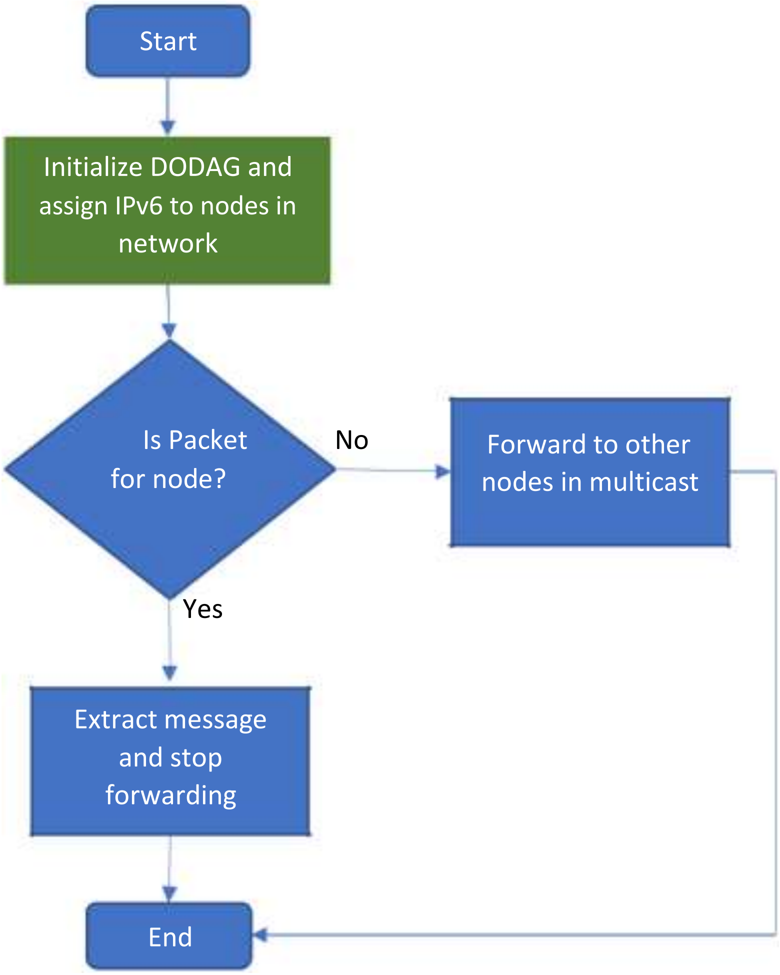

Start
Initialize DODAG and assign IPv6 to nodes in network
Is Packet for node?
No
Forward to other nodes in multicast
Yes
Extract message and stop forwarding
End

Figure 3.7: Algorithm for data forwarding

The RPL protocol is a distance vector proactive routing protocol that creates a tree-like routing topology called the destination-oriented directed acyclic graph (DODAG), routed towards one or more root or sink node. The directed acyclic graphs (DAGs) are created based on a user-specified specific objective function (OF). The default objective function (hop count) routing metric provided by the Contiki RPL border router was used.

The DODAG information object (DIO) and DODAG advertisement object (DAO) messages are used to initialize the nodes in the network and provide IPv6 addressing. Figure 3.7 shows the algorithm for forwarding packets to other nodes in the routing process.

3.4 Simulation procedure

The performance of the system was evaluated when the abstraction layer was implemented on the constrained device. The performance of the protocols was measured in terms of latency per message size in bytes transmitted for different QoS levels in terms of MQTT-SN and for CoAP. For MQTT-SN, the latency was measured as the difference in time between when a message was published and when the subscriber received the message. For CoAP the latency was evaluated as the difference in time between when a request was sent to the node and when an echoed response was received (the receiving nodes were programmed to echo back any message they receive).

3.4.1 Simulation environment setup

The simulator used in this experiment is the Contiki OS based Cooja Network Simulator (Jevtić & Zogović, 2009; Mehmood, 2017; Song *et al.*, 2016). This is a firmware level simulator, which emulates the target device hardware thereby enabling execution of OS code and application compiled for the target platform. Using this simulator, timing-sensitive software can be tested

(Eriksson, 2009). Two types of sensor nodes were used, the Zolertia Z1 mote and the Skymote. To interface the motes and the RSMB (Really Small Message Broker), an IPV6 Router for Low-Power and Lossy Networks (RPL) Border router was setup on a Zolertia Z1 mote and using a tool called tunslip utility, a Serial Line Interface Protocol (SLIP) bridge was created between the RPL network and the local network. Figure *3.8* shows the interface for the simulation as well as introducing parts of the simulator.

The part labelled A is the network map. It shows the nodes added to the simulation as numbered circles in different colors. It also shows the IPV6 address of the node. 10 x 10m^2 background grid for measuring distance, radio coverage area and other relevant information. The part labelled B shows the connection status of the border router to the local network via an open socket connection. The part labelled C is the simulation controls while D shows the mote debug/printed output in different color shades differentiating the nodes. The part labelled E shows the power tracker for the motes indicating the radio duty cycle for each mote.

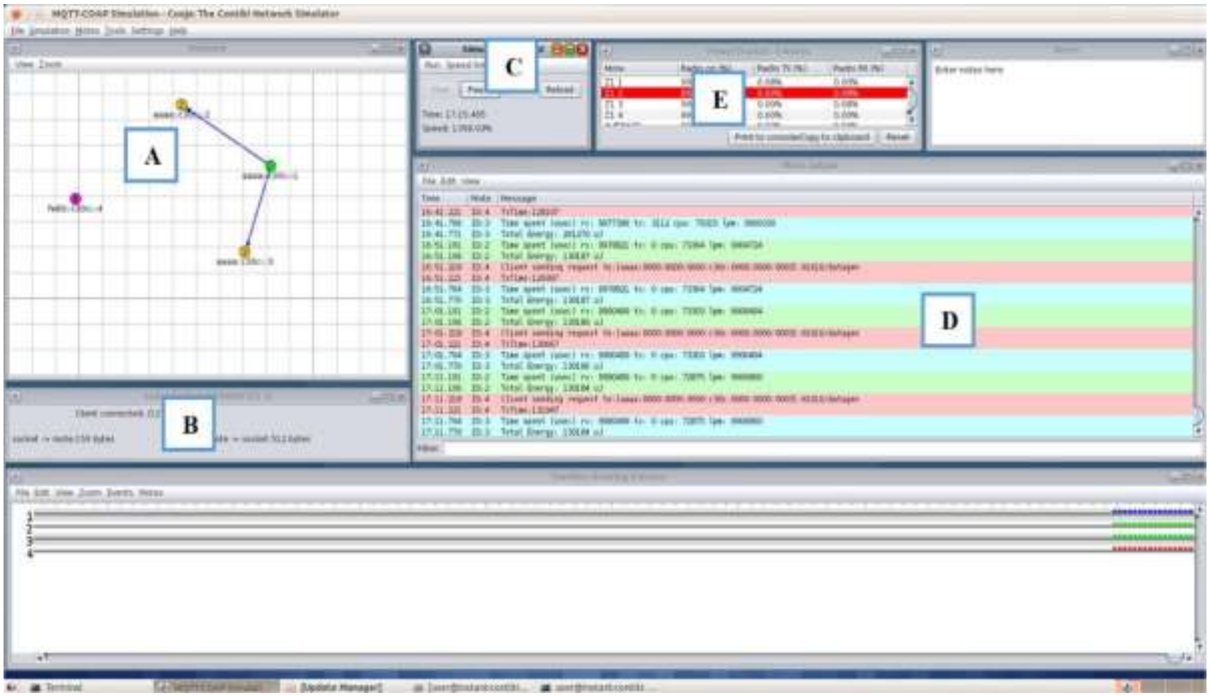


Figure 3.8: COOJA Simulator user interface

Energy estimation was done using the Energest Module in the Cooja Network Simulator. Energest is a software-based mechanism that estimates energy by measuring the accumulated time the sensor node is in different states such as CPU, IRQ, LPM, Rx and Tx (Schandy *et al*., 2015).

The focus for energy estimation is the application layer protocols therefore radio duty cycling was not considered (nullRDC was used to ensure that the radio is always active). Multi-hop transmission and data forwarding was also not considered because the operation of the protocols in the application layer is mainly CPU dependent so the energy consumption is focused on the information processing during transmission and also during reception.

The average energy consumed in any state is calculated using equation (3.1), therefore the total energy consumed on average by the node in the intervals chosen is calculated using equation (3.2).

$$=(\ -)\times \times \therefore =+++ \tag{3.1}$$

Where, (3.2)

→ Average energy at any particular state → Initial time the state was started
→ Time when the state ended
→ Current consumed at the particular state
→ Device supply voltage
→ Average energy consumed
→ Average energy consumed by the CPU

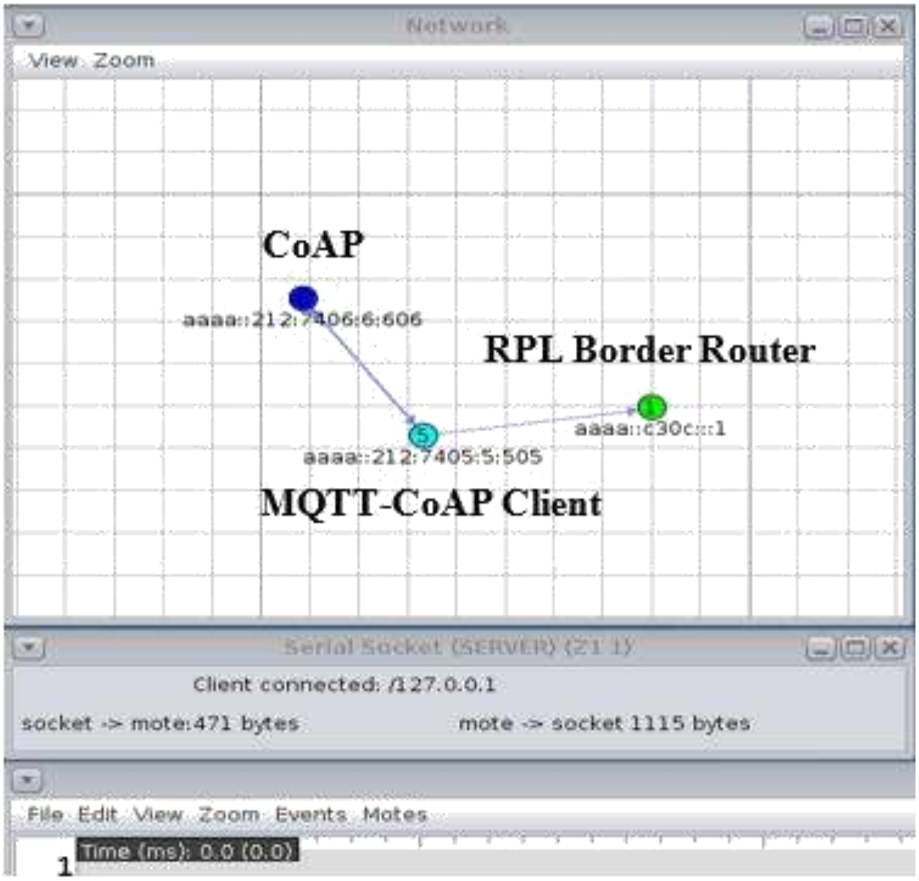
→ Average energy consumed during transmission → Average energy consumed during reception
→ Average energy consumed when in Low Power Mode

For the calculation to be done correctly, the current power consumptions of each state was obtained from the Sky mote datasheet (Moteiv Corporation, 2006).

Latency was estimated as the round trip time (RTT) between when data is transmitted and when it is received (echoed back). Considering Figure 3.7 the MQTT-CoAP node sends data to the CoAP node and the CoAP node echoes the received data back to the MQTT-CoAP node. The time difference between transmitting and receiving same data is calculated and recorded as the latency for CoAP. Same procedure was followed to evaluate the latency for MQTT at different QoS levels.

3.4.2 Software setup

Open source implementations of CoAP and MQTT-SN were CoAP for Contiki OS included in the example implementations and also the MQTT-SN for Contiki (Silva, 2016). The implementations were modified to incorporate the development of the abstraction layer. This was compiled for both mote types, Zolertia Z1 and Skymote and placed according to the network topology in Figure *3.9*. After organizing the nodes, the RSMB was started then the tunslip utility was run to bridge the RPL network and local network thereby enabling the nodes to communicate with the RSMB running on the local machine and listening for connections on port 1883.



```
Client connected: /127.0.0.1
socket -> mote:471 bytes                mote -> socket 1115 bytes
```

Figure 3.9: Network topology for sensor nodes

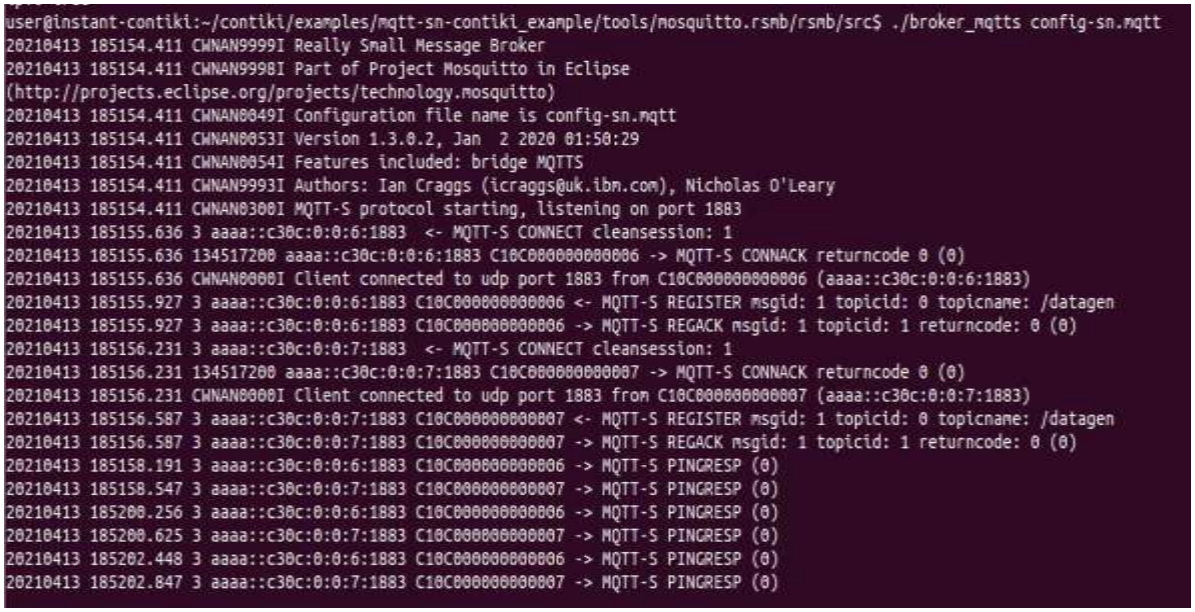
Figure *3.10* shows the RSMB being run using the sensor node config and executing the following command.

> ***./broker_mqtt*** *config-sn.mqtt*

Figure *3.11* shows the tunslip utility being run for the RPL router. The tunslip utility is included in the Contiki tools. It is first compiled in the Contiki tools directory by executing the command '**make tunslip6**'. The tunslip utility was used to bridge the **127.0.0.1** local address to the RPL network on **aaaa::1/64** IPv6 network. The command below was executed to achieve this.

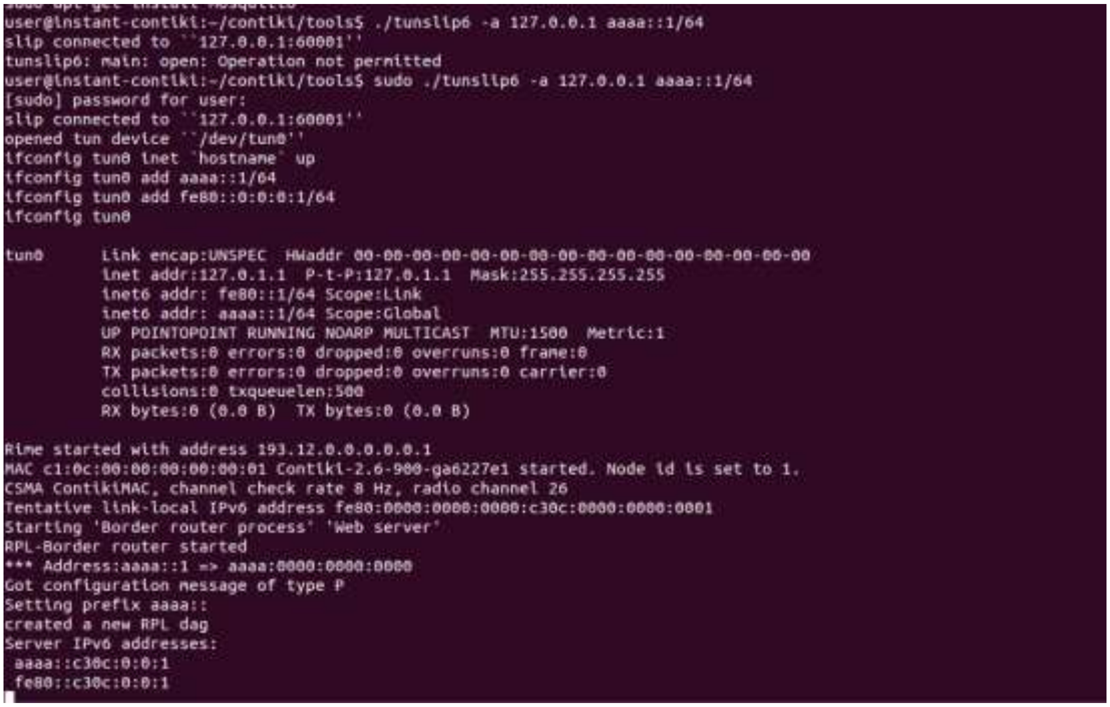
> ***./tunslip6*** *-a 127.0.0.1 aaaa::1/64*

The nodes were programmed to send packets in 10 s interval and to increase the packet size by 10 bytes after sending 10 packets starting from a packet size of 20 bytes.



```
user@instant-contiki:~/contiki/examples/mqtt-sn-contiki_example/tools/mosquitto.rsmb/rsmb/src$ ./broker_mqtts config-sn.mqtt
20210413 185154.411 CWNAN9999I Really Small Message Broker
20210413 185154.411 CWNAN9998I Part of Project Mosquitto in Eclipse
(http://projects.eclipse.org/projects/technology.mosquitto)
20210413 185154.411 CWNAN0049I Configuration file name is config-sn.mqtt
20210413 185154.411 CWNAN0053I Version 1.3.0.2, Jan  2 2020 01:50:29
20210413 185154.411 CWNAN0054I Features included: bridge MQTTS
20210413 185154.411 CWNAN9993I Authors: Ian Craggs (icraggs@uk.ibm.com), Nicholas O'Leary
20210413 185154.411 CWNAN0300I MQTT-S protocol starting, listening on port 1883
20210413 185155.636 3 aaaa::c30c:0:0:6:1883  <- MQTT-S CONNECT cleansession: 1
20210413 185155.636 134517200 aaaa::c30c:0:0:6:1883 C10C000000000006 -> MQTT-S CONNACK returncode 0 (0)
20210413 185155.636 CWNAN0000I Client connected to udp port 1883 from C10C000000000006 (aaaa::c30c:0:0:6:1883)
20210413 185155.927 3 aaaa::c30c:0:0:6:1883 C10C000000000006 <- MQTT-S REGISTER msgid: 1 topicid: 0 topicname: /datagen
20210413 185155.927 3 aaaa::c30c:0:0:6:1883 C10C000000000006 -> MQTT-S REGACK msgid: 1 topicid: 1 returncode: 0 (0)
20210413 185156.231 3 aaaa::c30c:0:0:7:1883  <- MQTT-S CONNECT cleansession: 1
20210413 185156.231 134517200 aaaa::c30c:0:0:7:1883 C10C000000000007 -> MQTT-S CONNACK returncode 0 (0)
20210413 185156.231 CWNAN0000I Client connected to udp port 1883 from C10C000000000007 (aaaa::c30c:0:0:7:1883)
20210413 185156.587 3 aaaa::c30c:0:0:7:1883 C10C000000000007 <- MQTT-S REGISTER msgid: 1 topicid: 0 topicname: /datagen
20210413 185156.587 3 aaaa::c30c:0:0:7:1883 C10C000000000007 -> MQTT-S REGACK msgid: 1 topicid: 1 returncode: 0 (0)
20210413 185158.191 3 aaaa::c30c:0:0:6:1883 C10C000000000006 -> MQTT-S PINGRESP (0)
20210413 185158.547 3 aaaa::c30c:0:0:7:1883 C10C000000000007 -> MQTT-S PINGRESP (0)
20210413 185200.256 3 aaaa::c30c:0:0:6:1883 C10C000000000006 -> MQTT-S PINGRESP (0)
20210413 185200.625 3 aaaa::c30c:0:0:7:1883 C10C000000000007 -> MQTT-S PINGRESP (0)
20210413 185202.448 3 aaaa::c30c:0:0:6:1883 C10C000000000006 -> MQTT-S PINGRESP (0)
20210413 185202.847 3 aaaa::c30c:0:0:7:1883 C10C000000000007 -> MQTT-S PINGRESP (0)
```

Figure 3.10: Really Small Message Broker (RSMB) running for MQTT-SN

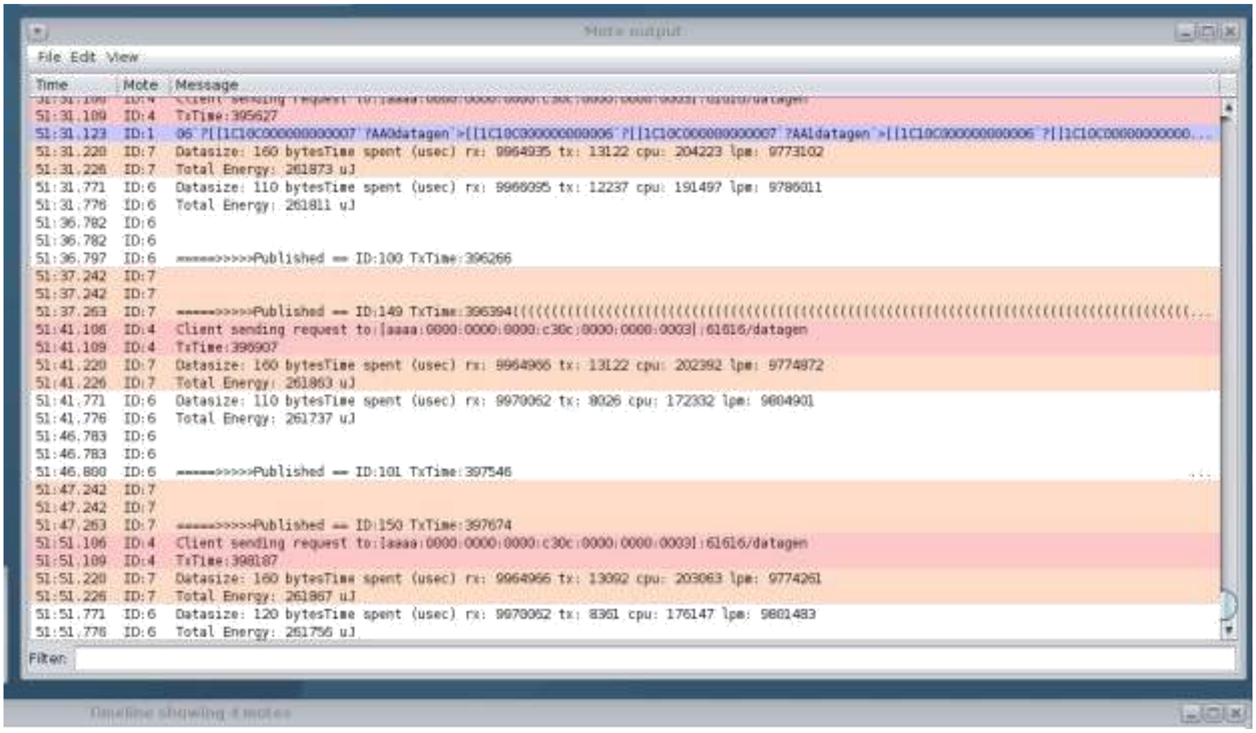


```
user@instant-contiki:~/contiki/tools$ ./tunslip6 -a 127.0.0.1 aaaa::1/64
slip connected to ``127.0.0.1:60001''
tunslip6: main: open: Operation not permitted
user@instant-contiki:~/contiki/tools$ sudo ./tunslip6 -a 127.0.0.1 aaaa::1/64
[sudo] password for user:
slip connected to ``127.0.0.1:60001''
opened tun device ``/dev/tun0''
ifconfig tun0 inet `hostname` up
ifconfig tun0 add aaaa::1/64
ifconfig tun0 add fe80::0:0:0:1/64
ifconfig tun0

tun0      Link encap:UNSPEC  HWaddr 00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00
          inet addr:127.0.1.1  P-t-P:127.0.1.1  Mask:255.255.255.255
          inet6 addr: fe80::1/64 Scope:Link
          inet6 addr: aaaa::1/64 Scope:Global
          UP POINTOPOINT RUNNING NOARP MULTICAST  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:500
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)

Rime started with address 193.12.0.0.0.0.0.1
MAC c1:0c:00:00:00:00:00:01 Contiki-2.6-900-ga6227e1 started. Node id is set to 1.
CSMA ContikiMAC, channel check rate 8 Hz, radio channel 26
Tentative link-local IPv6 address fe80:0000:0000:0000:c30c:0000:0000:0001
Starting 'Border router process' 'Web server'
RPL-Border router started
*** Address:aaaa::1 => aaaa:0000:0000:0000
Got configuration message of type P
Setting prefix aaaa::
created a new RPL dag
Server IPv6 addresses:
 aaaa::c30c:0:0:1
 fe80::c30c:0:0:1
```

Figure 3.11: Tunslip utility running to convert local to RPL network



```
51:31.106  ID:4  Client sending request to:[aaaa:0000:0000:0000:c30c:0000:0000:0003]:61616/datagen
51:31.109  ID:4  TxTime:395627
51:31.123  ID:1  06`?[|1C10C000000000007`?AA0datagen`>[|1C10C000000000006`?||1C10C000000000007`?AA1datagen`>[|1C10C000000000006`?||1C10C00000000000...
51:31.220  ID:7  Datasize: 160 bytesTime spent (usec) rx: 9964935 tx: 13122 cpu: 204223 lpm: 9773102
51:31.226  ID:7  Total Energy: 261873 uJ
51:31.771  ID:6  Datasize: 110 bytesTime spent (usec) rx: 9968095 tx: 12237 cpu: 191497 lpm: 9786011
51:31.776  ID:6  Total Energy: 261811 uJ
51:36.782  ID:6
51:36.782  ID:6
51:36.797  ID:6  =====>>>>>Published == ID:100 TxTime:396266
51:37.242  ID:7
51:37.242  ID:7
51:37.263  ID:7  =====>>>>>Published == ID:149 TxTime:396394((((((((((((((((((((((((((((((((((((((((((((((((((((((((((((((((((((((((((...
51:41.106  ID:4  Client sending request to:[aaaa:0000:0000:0000:c30c:0000:0000:0003]:61616/datagen
51:41.109  ID:4  TxTime:396907
51:41.220  ID:7  Datasize: 160 bytesTime spent (usec) rx: 9964966 tx: 13122 cpu: 202392 lpm: 9774872
51:41.226  ID:7  Total Energy: 261863 uJ
51:41.771  ID:6  Datasize: 110 bytesTime spent (usec) rx: 9970062 tx: 8026 cpu: 172332 lpm: 9804901
51:41.776  ID:6  Total Energy: 261737 uJ
51:46.783  ID:6
51:46.783  ID:6
51:46.800  ID:6  =====>>>>>Published == ID:101 TxTime:397546
51:47.242  ID:7
51:47.242  ID:7
51:47.263  ID:7  =====>>>>>Published == ID:150 TxTime:397674
51:51.106  ID:4  Client sending request to:[aaaa:0000:0000:0000:c30c:0000:0000:0003]:61616/datagen
51:51.109  ID:4  TxTime:398187
51:51.220  ID:7  Datasize: 160 bytesTime spent (usec) rx: 9964966 tx: 13092 cpu: 203063 lpm: 9774261
51:51.226  ID:7  Total Energy: 261867 uJ
51:51.771  ID:6  Datasize: 120 bytesTime spent (usec) rx: 9970062 tx: 8361 cpu: 176147 lpm: 9801483
51:51.776  ID:6  Total Energy: 261756 uJ
```

Figure 3.12: Mote serial output of individual nodes being simulated

From the mote output shown in Figure 3.12, the serial output of the motes being simulated are shown. This contains the total time spent per activity in microseconds, the data size being transmitted, the total round trip time for a packet received after being sent and the total energy that has been consumed per that activity computed in the node. The message payload being transmitted or received is also shown.

The data for latency was collected for packet sizes between 20 and 80 bytes. It was observed that the maximum packet size that could be transmitted from the node is 87bytes. This is due to the fact that the motes ran on Zigbee and this follows the IEEE 802.15.4 standard that has a limit on the data size allocated to user application (Burchfield *et al*., 2007).

For each packet size, 20 latency values and 20 energy consumption values were collected and the average latency and energy consumption values were calculated and recorded.

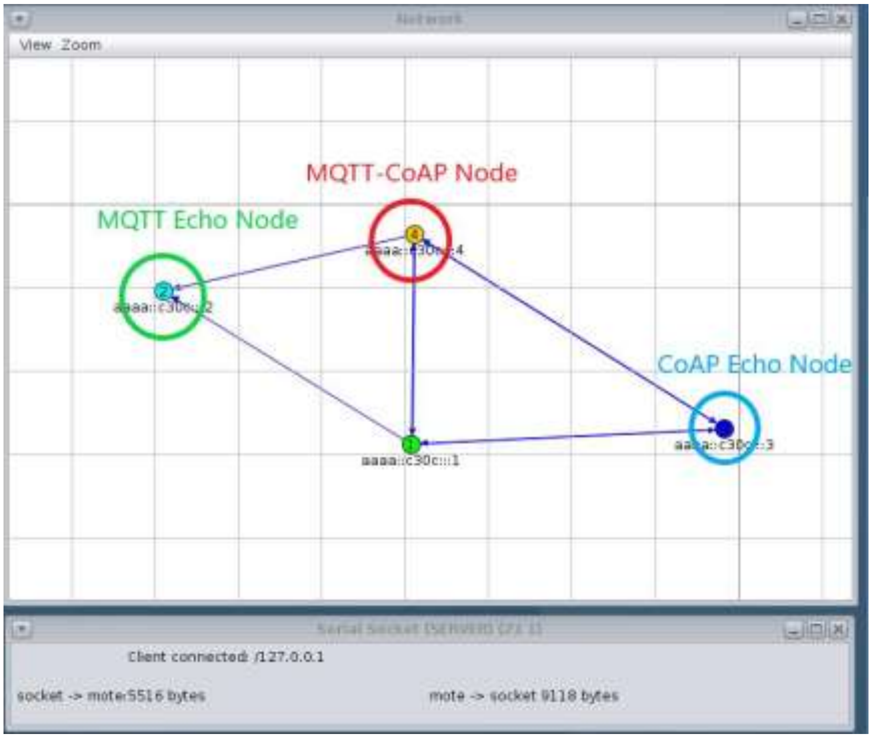
CHAPTER FOUR

4.0 RESULTS AND DISCUSSION

Several results were obtained and are hereby ordered according to the objectives of this study. The abstraction layer results indicate the outcome of the abstraction layer development including the organization of nodes to test data transmission and reception. The results for latency and energy consumption highlight the results obtained when the abstraction layer was used in a sensor node for communication. This section is then concluded with a discussion of the results obtained.

4.1 Results of the abstraction layer development

The results obtained from accomplishing the first objective which is the development of the abstraction layer is illustrated in Figure 4.1 and Figure 4.2.

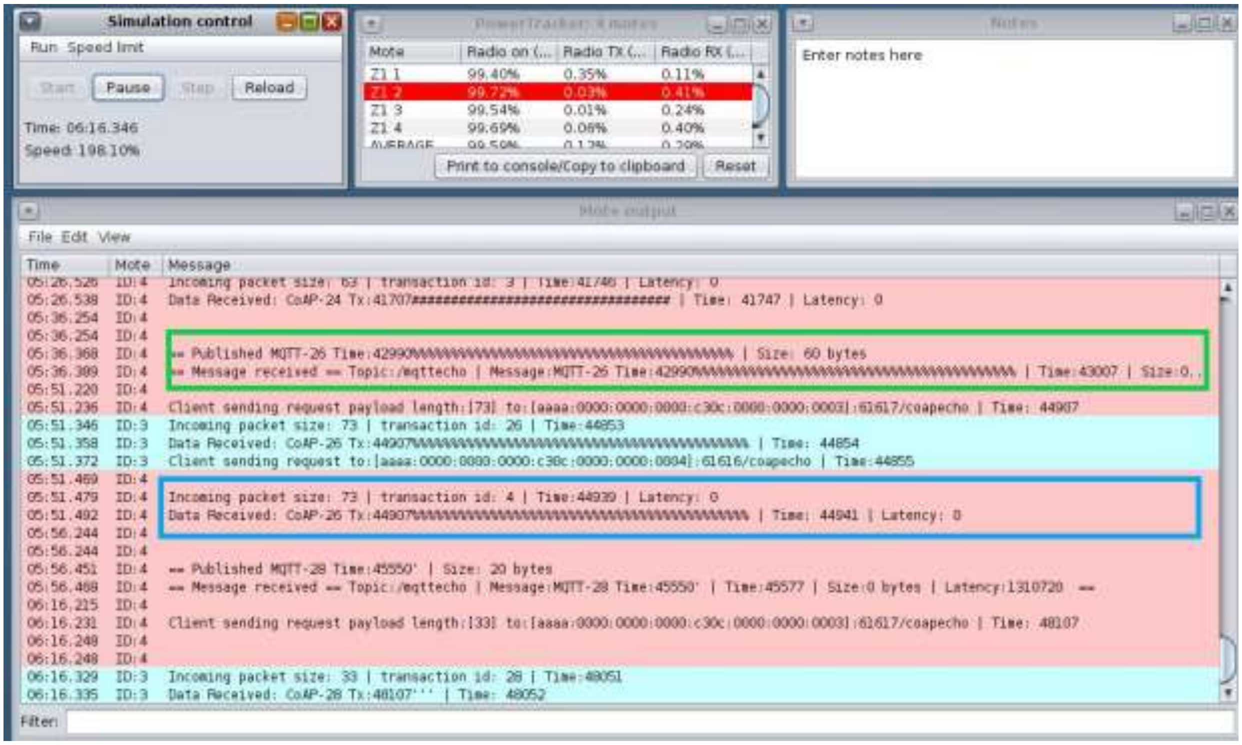


Client connected: /127.0.0.1

socket -> mote:5516 bytes mote -> socket 9118 bytes

Figure 4.1: Simulator nodes setup to test the abstraction layer

Figure *4.1* shows the arrangement of the sensor nodes also labelled appropriately. The function of the MQTT and CoAP Echo Nodes is to echo back any message that is sent from the node running the integrated protocols. This proves that the node running the integrated protocols can receive data from both nodes simultaneously. The node that is not circled is the RPL border router node.



Time: 06:16.346
Speed: 198.10%

Mote	Radio on (...	Radio TX (...	Radio RX (...
Z1 1	99.40%	0.35%	0.11%
Z1 2	99.72%	0.03%	0.41%
Z1 3	99.54%	0.01%	0.24%
Z1 4	99.69%	0.06%	0.40%
AVERAGE	99.59%	0.12%	0.29%

```
Time       Mote  Message
05:26.526  ID:4  Incoming packet size: 63 | transaction id: 3 | Time:41746 | Latency: 0
05:26.538  ID:4  Data Received: CoAP-24 Tx:41707################################### | Time: 41747 | Latency: 0
05:36.254  ID:4
05:36.254  ID:4
05:36.368  ID:4  -- Published MQTT-26 Time:42990%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% | Size: 60 bytes
05:36.389  ID:4  -- Message received -- Topic:/mqttecho | Message:MQTT-26 Time:42990%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% | Time:43007 | Size:0..
05:51.220  ID:4
05:51.236  ID:4  Client sending request payload length:[73] to:[aaaa:0000:0000:0000:c30c:0000:0000:0003]:61617/coapecho | Time: 44907
05:51.346  ID:3  Incoming packet size: 73 | transaction id: 26 | Time:44853
05:51.358  ID:3  Data Received: CoAP-26 Tx:44907%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% | Time: 44854
05:51.372  ID:3  Client sending request to:[aaaa:0000:0000:0000:c30c:0000:0000:0004]:61616/coapecho | Time:44855
05:51.469  ID:4
05:51.479  ID:4  Incoming packet size: 73 | transaction id: 4 | Time:44939 | Latency: 0
05:51.492  ID:4  Data Received: CoAP-26 Tx:44907%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% | Time: 44941 | Latency: 0
05:56.244  ID:4
05:56.244  ID:4
05:56.451  ID:4  -- Published MQTT-28 Time:45550' | Size: 20 bytes
05:56.468  ID:4  -- Message received -- Topic:/mqttecho | Message:MQTT-28 Time:45550' | Time:45577 | Size:0 bytes | Latency:1310720  --
06:16.215  ID:4
06:16.231  ID:4  Client sending request payload length:[33] to:[aaaa:0000:0000:0000:c30c:0000:0000:0003]:61617/coapecho | Time: 48107
06:16.248  ID:4
06:16.248  ID:4
06:16.329  ID:3  Incoming packet size: 33 | transaction id: 28 | Time:48051
06:16.335  ID:3  Data Received: CoAP-28 Tx:48107''' | Time: 48052
```

Figure 4.2: Serial output from sensor nodes

Figure *4.2* shows the serial output from the sensor nodes. The serial output contains logs for publishing and subscribing to messages over MQTT and also sending and receiving messages over CoAP. The green box in the figure indicates that the Node 4 which is the node containing the integrated protocols published some data over MQTT and received the same data that was published because the data was echoed by the MQTT Echo Node. The publish and subscribe actions are shown in time logs 05:36.368 and 05:36.389 respectively.

Time logs 05:51.236 shows when Node 4 sent some payload over CoAP using the endpoint "/coapecho". Time logs 05:51.345-372 shows that the CoAP Echo nodes received the sent payload and sent back the data received over CoAP. The blue boxed enclosing in the figure shows that the Node 4 received the echoed data over CoAP.

This proves that via the abstraction layer, we are able to transmit and receive data over MQTT and CoAP.

4.2 Results for testing latency and energy consumption

The experimental setup has one publisher, one broker and one CoAP client, thus there was zero loss in transmitted packets.

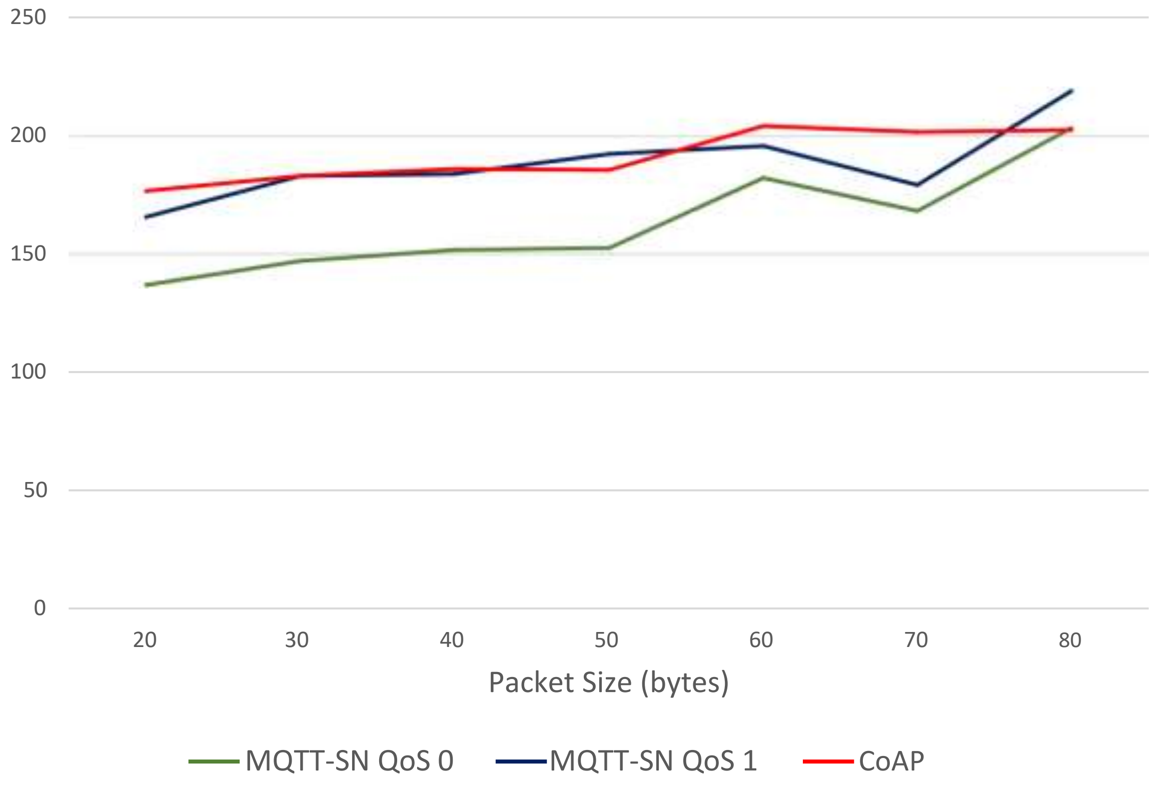


Figure 4.3: CoAP and MQTT-SN Latency at different packet sizes

The results Figure *4.3* shows the graph of latency against packet size for MQTT-SN and CoAP. From the figure, the latency values were observed to slightly increase as the packet size increased. From the results shown in Table *4.1*, the lowest latency for single packet transmission was observed in MQTT-SN QoS 0 which was between 137ms for packet size of 20 bytes and 203ms for packet size of 80 bytes. Similar latency values were obtained for the CoAP and MQTT-SN QoS 1. For CoAP we observed between 176.8ms and 202.6ms while we observed between 165.9ms and 219ms for MQTT-SN QoS 1. The average latency was obtained to be 163.2ms, 188.5ms and 191.5ms for MQTT-SN QoS 0, MQTT-SN QoS 1 and CoAP respectively.

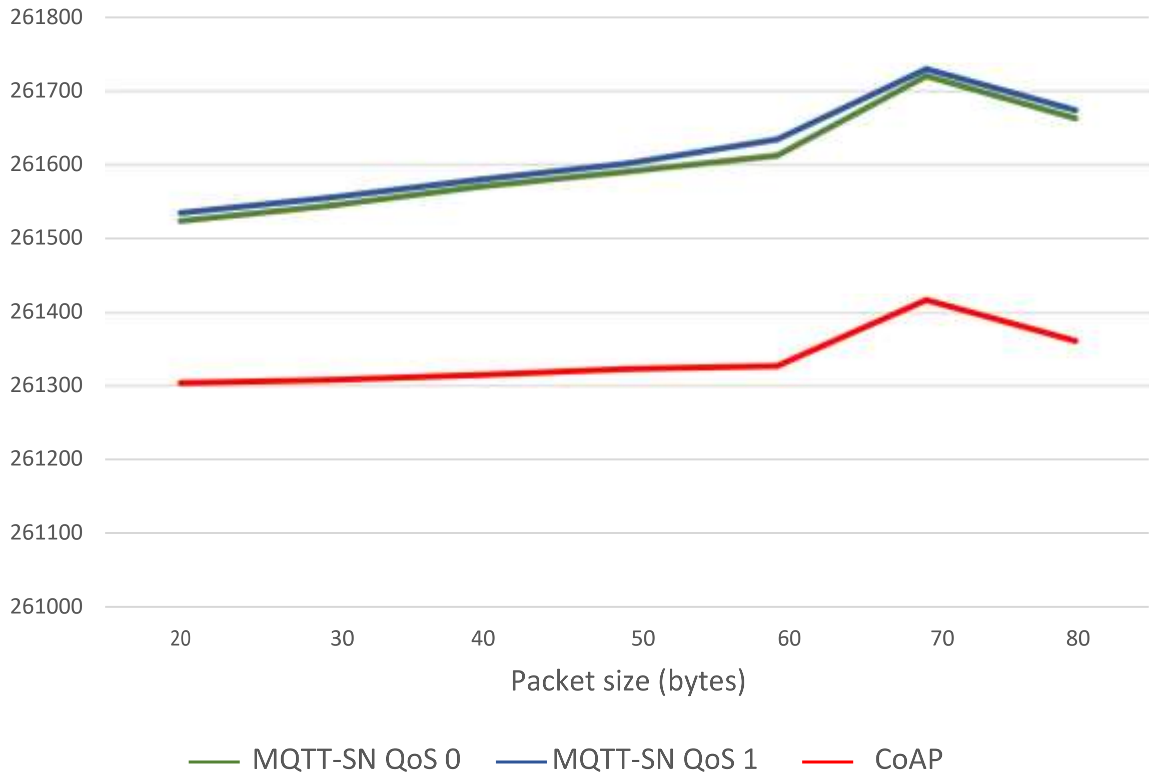


Figure 4.4: Average energy consumption of CoAP and MQTT-SN transmitting and receiving different packet sizes

Energy consumption of the node when using MQTT-SN for a single Tx/Rx operation in a 10s interval showed an average of 261.6mJ for both QoS 0 and QoS 1 while an average of 261.3mJ was observed for CoAP. The results in Figure *4.4* shows the energy consumption for the different protocols when transmitting and receiving different packet sizes. Although it is not entirely clear as to why there are slightly irregular behaviors of latency and energy consumption at above 60 bytes, the MQTT-SN Specification (Stanford-Clark & Truong, 2013b) restricts maximum payload size to 60 bytes over ZigBee network due to the ZigBee network/APS layer.

Table 4.1 Latency and Energy required to transmit different packet sizes for MQTT-SN and CoAP Protocols

	Latency (ms) (RTT)			**Energy(μJ)**		
Packet Size	**MQTT QoS0**	**MQTT QoS1**	**CoAP**	**MQTT QoS0**	**MQTT QoS1**	**CoAP**
20	137	165.9	176.8	261524.4	261535.4	261304.5
30	147.2	183.1	183.2	261544.8	261555	261308.1
40	151.8	183.9	186.1	261570.9	261579.8	261315
50	152.6	192.5	185.5	261590.6	261601.5	261323.8
60	182.3	195.6	204.2	261612.9	261634.7	261327.1
70	168.1	179.2	201.9	261719.4	261730	261417
80	203.4	219	202.6	261663	261673.9	261360.9
AVERAGE	163.2	188.5	191.5	261603.7	261615.8	261336.6

4.3 Discussion of Results

No integration of CoAP and MQTT-SN protocol in constrained WSN devices has been reported in literature, however, various authors have carried out comparative evaluation of both protocols. For example, authors in (Thangavel *et al*., 2014) evaluated the performance of MQTT and CoAP via a common middleware but it was implemented on devices with no resource constrains for running the protocols also classified as high-end IoT devices (Ojo *et al*., 2018). They found that when compared to CoAP, MQTT messages experience lower delays for lower link packet loss and higher delays for higher link packet loss.

The latency results obtained for MQTT-SN were comparable to the results of Guha Roy *et al*., (2018). In their study they observed that the end to end delay for MQTT-SN QoS 0 and QoS 1 were 1.83s and 2.27s respectively for a 1000 byte data size. Since the packet size is directly proportional to the end-to-end delay, the results in this study will transmit the 1000 bytes in 20 different packets of 50 bytes which will amount to a total RTT latency of 3.052s and an end-to-end delay of approximately 1.5s for QoS 0 and 1.93s for QoS 1.

For CoAP a round trip time of 176.8ms was observed which is 88.4ms end-to-end latency for 20 bytes packet size while Safaei *et al*., (2018) recorded 70.4ms latency for 14 byte packet which is similar to the results obtained in this study with respect to the packet size. De Caro *et al*., (2013) compared CoAP with MQTT using various metrics and recorded an average of 127ms RTT latency for a packet size of 140bytes. This is significantly lower than what was observed in this research and may be due to the delays incurred in the border router translating packets between RPL network and local network. Their study was performed using an android smart phone and a Windows PC which possesses more resources and processing capabilities compared to the resource constrained nodes in this study. Mijovic *et al*., (2016) performed several experiments on CoAP and MQTT over different network conditions. First over LAN, then IoT ISP network and Cellular network. The Latency values observed for CoAP was approximately 200ms over cellular network, 42ms over LAN and 75ms over IoT ISP network for a packet size of 80bytes. The results they obtained over cellular network is similar to the results obtained in this study.

When energy consumption was considered, CoAP was observed to be slightly more energy efficient compared to MQTT-SN. This varies with the report of Martí *et al*., (2019) who recorded that MQTT-SN was slightly more efficient than CoAP and went further to recommend using MQTT-SN for communication when energy saving is a priority in the development. The complexity of MQTT-SN lies in the RSMB that does the translation from the UDP protocol supported by MQTT-SN to the normal MQTT TCP protocol. The complexity is less for CoAP using the non-confirmable method of communication hence the reduction in energy consumption.

The results obtained in this study has proven that it is possible to integrate MQTT-SN and CoAP protocols on a constrained sensor node without any significant drop in performance of the individual protocols. At the time of research only one author (Thangavel *et al*., 2014) was able to implement something similar with MQTT and CoAP protocol on a device with a lot of processing capabilities using a common middleware which is also an abstraction layer. This research could also open new doors to the possibility of deploying sensor nodes capable of self-adjusting communication efficiency in the application layer.

CHAPTER FIVE

5.0 CONCLUSION AND RECOMMENDATIONS

5.1 Conclusion

In this research, an abstraction layer that serves to integrate MQTT-SN and CoAP is implemented on constrained devices and the performance of each protocol is evaluated when this technique is implemented. This is to take advantage of both protocols on a single sensor node under varying application requirements.

The result of the study showed that latency values slightly increase as the packet size increased. The lowest latency was observed in MQTT-SN QoS 0 while similar latency values were obtained for the CoAP and MQTT-SN QoS 1. The average latency was observed to be 163.2ms, 188.5ms and 191.5ms for MQTT-SN QoS 0, MQTT-SN QoS 1 and CoAP respectively.

The performance evaluation and comparisons that have been carried out between the results of latency and energy consumption in this study and the results recorded in other studies goes ahead to prove that there was no significant drop in performance of the individual protocols when MQTT and CoAP protocols were integrated in a constrained device.

The scalability of WSNs implemented using the MQTT-SN system because of the publish/subscribe technique through a broker can be further improved using the integrated system with a redundant link through the CoAP protocol that would be enabled in the same system.

Although it is evident that even with the implementation of the integrated protocol system there is no performance impact on each of the protocols, it would be of added value to have multiple algorithms that could be implemented to optimally select a communication protocol based on the network conditions or other mitigating factors. Only two protocol selection algorithms were tested therefore this research work fell short of being able to implement more algorithms that could optimize the use of the protocols.

5.2 Recommendation

As a significant improvement on this work, further studies can be done to leverage this technique and adaptively switching between these protocols in situations of power saving, broker failures, varying link performances, quality of service (QoS) or to satisfy other desired network conditions.

REFERENCES

Aazam, M., & Huh, E. N. (2014). Fog computing and smart gateway based communication for cloud of things. *Proceedings - 2014 International Conference on Future Internet of Things and Cloud, FiCloud 2014*, 464–470. https://doi.org/10.1109/FiCloud.2014.83

Akyildiz, I. F., Weilian Su, Sankarasubramaniam, Y., & Cayirci, E. (2002). A survey on sensor networks. *IEEE Communications Magazine*, *40*(8), 102–114. https://doi.org/10.1109/MCOM.2002.1024422

Al-Fuqaha, A., Guizani, M., Mohammadi, M., Aledhari, M., & Ayyash, M. (2015). Internet of Things: A Survey on Enabling Technologies, Protocols, and Applications. *IEEE Communications Surveys and Tutorials*, *17*(4), 2347–2376. https://doi.org/10.1109/COMST.2015.2444095

Alemdar, H., & Ersoy, C. (2010). Wireless sensor networks for healthcare: A survey. *Computer Networks*, *54*(15), 2688–2710. https://doi.org/10.1016/j.comnet.2010.05.003

Anand Nayyar, R. S. (2015). A Comprehensive Review of Simulation Tools for Wireless Smoke Network (WSNs). *Journal of Wireless Networking and Communications*, *19–47*(1), 19–47. https://doi.org/10.5923/j.jwnc.20150501.03

Asensio, Á., Marco, Á., Blasco, R., & Casas, R. (2014). Protocol and Architecture to Bring Things into Internet of Things. *International Journal of Distributed Sensor Networks*, *10*(4), 158252. https://doi.org/10.1155/2014/158252

Atzori, L., Iera, A., & Morabito, G. (2010). The Internet of Things: A survey. *Computer Networks*, *54*(15), 2787–2805. https://doi.org/10.1016/J.COMNET.2010.05.010

Banavar, G., Chandra, T., Mukherjee, B., Nagarajarao, J., Strom, R. E., & Sturman, D. C. (1999). An efficient multicast protocol for content-based publish-subscribe systems. *Proceedings. 19th IEEE International Conference on Distributed Computing Systems (Cat. No.99CB37003)*, 262–272. https://doi.org/10.1109/icdcs.1999.776528

Bandyopadhyay, S., & Bhattacharyya, A. (2013). Lightweight Internet protocols for web enablement of sensors using constrained gateway devices. *2013 International Conference on*

Computing, Networking and Communications (ICNC), 334–340. https://doi.org/10.1109/ICCNC.2013.6504105

Bonomi, F., Milito, R., Natarajan, P., & Zhu, J. (2014). Fog computing: A platform for internet of things and analytics. *Studies in Computational Intelligence*, *546*, 169–186. https://doi.org/10.1007/978-3-319-05029-4_7

Bormann, C., Lemay, S., Tschofenig, H., Hartke, K., & Silverajan, B. (2018). *CoAP (Constrained Application Protocol) over TCP, TLS, and WebSockets* (B. Raymor (Ed.)). https://doi.org/10.17487/RFC8323

Burchfield, T. R., Venkatesan, S., & Weiner, D. (2007). Maximizing Throughput in ZigBee Wireless Networks through Analysis, Simulations and Implementations. *Proc. Int. Workshop Localized Algor. Protocols WSNs*, 15–29. http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.129.8350

Chelloug, S. A., & El-Zawawy, M. A. (2018). Middleware for Internet of Things: Survey and Challenges. *Intelligent Automation and Soft Computing*, *24*(2), 309–318. https://doi.org/10.1080/10798587.2017.1290328

Chen, C. C., & Liao, C. H. (2011). Model-based object tracking in wireless sensor networks. *Wireless Networks*, *17*(2), 549–565. https://doi.org/10.1007/s11276-010-0296-5

Chen, Y., & Kunz, T. (2016). Performance evaluation of IoT protocols under a constrained wireless access network. *2016 International Conference on Selected Topics in Mobile and Wireless Networking, MoWNeT 2016*. https://doi.org/10.1109/MoWNet.2016.7496622

Christin, D., Reinhardt, A., & Mogre, P. (2009). Wireless Sensor Networks and the Internet of Things: Selected Challenges. *Proceedings of the 8th GI/ITG KuVS*, 31–33. http://www.ti5.tu-harburg.de/events/fgsn09/proceedings/fgsn_031.pdf

CISCO. (2016). *Cisco Visual Networking Index: Forecast and Methodology*. http://www.cisco.com/c/en/us/solutions/collateral/service- provider/visual-networking-index-vni/complete-white-paper-c11- 481360.html

Cohn, R., & Coppen, R. (2014). MQTT Version 3.1.1. In *OASIS Standard*. https://doi.org/10.1073/pnas.1201805109

Colitti, W., Steenhaut, K., & Caro, N. De. (2011). Integrating Wireless Sensor Networks with the Web. *Extending the Internet to Low Power and Lossy Networks (IP+ SN 2011)*, 2–6.

Correia, N., Sacramento, D., & Schutz, G. (2016). Dynamic Aggregation and Scheduling in CoAP/Observe-Based Wireless Sensor Networks. *IEEE Internet of Things Journal*, *3*(6), 923–936. https://doi.org/10.1109/JIOT.2016.2517120

Data Distribution Service (DDS). (2015). https://www.omg.org/spec/DDS

Davis, E. G., Calveras, A., & Demirkol, I. (2013). Improving packet delivery performance of publish/subscribe protocols in wireless sensor networks. *Sensors (Switzerland)*, *13*(1), 648–680. https://doi.org/10.3390/s130100648

De Caro, N., Colitti, W., Steenhaut, K., Mangino, G., & Reali, G. (2013). Comparison of two lightweight protocols for smartphone-based sensing. *IEEE SCVT 2013 - Proceedings of 20th IEEE Symposium on Communications and Vehicular Technology in the BeNeLux*, 0–5. https://doi.org/10.1109/SCVT.2013.6735994

Ding, D., Cooper, R. A., Pasquina, P. F., & Fici-Pasquina, L. (2011). Sensor technology for smart homes. *Maturitas*, *69*(2), 131–136. https://doi.org/10.1016/j.maturitas.2011.03.016

Dizdarevic, J., Carpio, F., Jukan, A., & Masip-Bruin, X. (2019). *Survey of Communication Protocols for Internet-of-Things and Related Challenges of Fog and Cloud Computing Integration*. *1*(1), 1–30. https://doi.org/10.1145/3292674

Dunkels, A., Grönvall, B., & Voigt, T. (2004). Contiki - A lightweight and flexible operating system for tiny networked sensors. *Proceedings - Conference on Local Computer Networks, LCN*, 455–462. https://doi.org/10.1109/LCN.2004.38

Eclipse. (2019). *Eclipse Mosquitto*. Mosquitto.Org. https://mosquitto.org/

Eriksson, J. (2009). *Detailed Simulation of Heterogeneous Wireless Sensor Networks*.

Eriksson, J., Dunkels, A., Finne, N., Österlind, F., & Voigt, T. (2007). Mspsim-an extensible simulator for msp430-equipped sensor boards. *Eprints.Sics.Se*. http://eprints.sics.se/964

Eugster, P. T., Felber, P. A., Guerraoui, R., & Kermarrec, A.-M. (2003). The many faces of publish/subscribe. *ACM Computing Surveys*, *35*(2), 114–131.

https://doi.org/10.1145/857076.857078

Ge, S. Y., Chun, S. M., Kim, H. S., & Park, J. T. (2016). Design and implementation of interoperable IoT healthcare system based on international standards. *2016 13th IEEE Annual Consumer Communications and Networking Conference, CCNC 2016*, 119–124. https://doi.org/10.1109/CCNC.2016.7444743

Ghosh, A., Ratasuk, R., Mondal, B., Mangalvedhe, N., & Thomas, T. (2010). LTE-advanced: Next-generation wireless broadband technology. *IEEE Wireless Communications*, *17*(3), 10– 22. https://doi.org/10.1109/MWC.2010.5490974

Giusto, D., Iera, A., Morabito, G., & Atzori, L. (Eds.). (2010). *The Internet of Things*. Springer New York. https://doi.org/10.1007/978-1-4419-1674-7

Gluhak, A., Krco, S., Nati, M., Pfisterer, D., Mitton, N., & Razafindralambo, T. (2011). A survey on facilities for experimental internet of things research. *IEEE Communications Magazine*, *49*(11), 58–67. https://doi.org/10.1109/MCOM.2011.6069710

Gnawali, O., Fonseca, R., Jamieson, K., Moss, D., & Levis, P. (2009). Collection tree protocol. *Proceedings of the 7th ACM Conference on Embedded Networked Sensor Systems - SenSys ’09*, 1. https://doi.org/10.1145/1644038.1644040

Gomez, C., & Paradells, J. (2010). Wireless home automation networks: A survey of architectures and technologies. *IEEE Communications Magazine*, *48*(6), 92–101. https://doi.org/10.1109/MCOM.2010.5473869

Govindan, K., & Azad, A. P. (2015). End-to-end service assurance in IoT MQTT-SN. *2015 12th Annual IEEE Consumer Communications and Networking Conference (CCNC)*, 290–296. https://doi.org/10.1109/CCNC.2015.7157991

Granjal, J., Monteiro, E., & Sa Silva, J. (2015). Security for the Internet of Things: A Survey of Existing Protocols and Open Research Issues. *IEEE Communications Surveys & Tutorials*, *17*(3), 1294–1312. https://doi.org/10.1109/COMST.2015.2388550

Guha Roy, D., Mahato, B., De, D., & Buyya, R. (2018). Application-aware end-to-end delay and message loss estimation in Internet of Things (IoT) — MQTT-SN protocols. *Future Generation Computer Systems, 89*, 300–316. https://doi.org/10.1016/j.future.2018.06.040

Hakiri, A., Berthou, P., Gokhale, A., Abdellatif, S., Hakiri, A., Berthou, P., Gokhale, A., Publish, S. A., Hakiri, A., Berthou, P., & Gokhale, A. (2017). *Publish / subscribe-enabled software defined networking for efficient and scalable IoT communications To cite this version : HAL Id : hal-01633323 Publish / Subscribe-enabled Software Defined Networking for Efficient and Scalable IoT Communications*.

Hinze, A., Sachs, K., & Buchmann, A. (2009). Event-based applications and enabling technologies. *Proceedings of the Third ACM International Conference on Distributed Event-Based Systems - DEBS '09*, 1. https://doi.org/10.1145/1619258.1619260

Hui, J., Culler, D., & Chakrabarti, S. (2011). 6LoWPAN: Incorporating IEEE 802.15. 4 into the IP architecture. *IPSO Alliance White Paper*.

Hui, J. W., & Culler, D. E. (2008). Extending IP to low-power, wireless personal area networks. *IEEE Internet Computing*, *12*(4), 37–45. https://doi.org/10.1109/MIC.2008.79

Iova, O., Picco, P., Istomin, T., & Kiraly, C. (2016). RPL: The Routing Standard for the Internet of Things... or Is It? *IEEE Communications Magazine*, *54*(11), 16–22. https://doi.org/10.1109/MCOM.2016.1600397CM

Jevtić, M., & Zogović, N. (2009). Evaluation of wireless sensor network simulators. *Proceedings of the 17th*, 1303–1306. http://2009.telfor.rs/files/radovi/10_48.pdf

Jia, Y., Bodanese, E., Phillips, C., Bigham, J., & Tao, R. (2014). Improved reliability of large scale publish/subscribe based MOMs using model checking. *2014 IEEE Network Operations and Management Symposium (NOMS)*, 1–8. https://doi.org/10.1109/NOMS.2014.6838311

Kaiwen, C., Kumar, A., Xavier, N., & Panda, S. K. (2017). An intelligent home appliance control-based on WSN for smart buildings. *IEEE International Conference on Sustainable Energy Technologies, ICSET*, *0*, 282–287. https://doi.org/10.1109/ICSET.2016.7811796

Koster, M., SmartThings, Keranen, A., Jimenez, J., & Ericsson. (2019). *Publish-Subscribe Broker for the Constrained Application Protocol CoAP*. http://datatracker.ietf.org/drafts/current/.

Kulkarni, P., & Ozturk, Y. (2011). MPHASiS: Mobile patient healthcare and sensor information system. *Journal of Network and Computer Applications*, *34*(1), 402–417. https://doi.org/10.1016/j.jnca.2010.03.030

Kürschner, C., Condea, C., Kasten, O., & Thiesse, F. (2008). Discovery Service Design in the EPCglobal Network. *The Internet of Things*, 19–34. https://doi.org/10.1007/978-3-540-78731-0_2

Lakkundi, V., & Singh, K. (2014). Lightweight DTLS implementation in CoAP-based Internet of Things. *2014 20th Annual International Conference on Advanced Computing and Communications, ADCOM 2014 - Proceedings*, 7–11. https://doi.org/10.1109/ADCOM.2014.7103240

Lesjak, C., Hein, D., Hofmann, M., Maritsch, M., Aldrian, A., Priller, P., Ebner, T., Ruprechter, T., & Pregartner, G. (2015). Securing smart maintenance services: Hardware-security and TLS for MQTT. *2015 IEEE 13th International Conference on Industrial Informatics (INDIN)*, 1243–1250. https://doi.org/10.1109/INDIN.2015.7281913

Levis, P., Madden, S., Polastre, J., Szewczyk, R., Whitehouse, K., Woo, A., Gay, D., Hill, J., Welsh, M., Brewer, E., & Culler, D. (2005). TinyOS: An operating system for sensor networks. *Ambient Intelligence*, 115–148. https://doi.org/10.1007/3-540-27139-2_7

Levis, Philip, & Lee, N. (2003). Tossim: A simulator for tinyos networks. *UC Berkeley, September*, 1–17. http://www.tinyos.net/dist-1.1.0/snapshot-1.1.11Feb2005cvs/doc/nido.pdf

Liao, K., & Lin, C. (2017). *Implementation of IoT Applications based on MQTT and MQTT-SN in IPv6 over BLE*. *6*(1), 48–49.

Luzuriaga, J. E., Cano, J. C., Calafate, C., Manzoni, P., Perez, M., & Boronat, P. (2015). Handling mobility in IoT applications using the MQTT protocol. *2015 Internet Technologies and Applications (ITA)*, 245–250. https://doi.org/10.1109/ITechA.2015.7317403

Martí, Garcia-Rubio, & Campo. (2019). Performance Evaluation of CoAP and MQTT_SN in an IoT Environment. *Proceedings*, *31*(1), 49. https://doi.org/10.3390/proceedings2019031049

Mashal, I., Alsaryrah, O., Chung, T. Y., Yang, C. Z., Kuo, W. H., & Agrawal, D. P. (2015). Choices for interaction with things on Internet and underlying issues. *Ad Hoc Networks*, *28*, 68–90. https://doi.org/10.1016/j.adhoc.2014.12.006

Mehmood, T. (2017). *COOJA Network Simulator: Exploring the Infinite Possible Ways to Compute the Performance Metrics of IOT Based Smart Devices to Understand the Working* *of IOT Based Compression & Routing Protocols*. http://arxiv.org/abs/1712.08303

Mekni, M., & Moulin, B. (2008). A survey on sensor webs simulation tools. *Proceedings - 2nd Int. Conf. Sensor Technol. Appl., SENSORCOMM 2008, Includes: MESH 2008 Conf. Mesh Networks; ENOPT 2008 Energy Optim. Wireless Sensors Networks, UNWAT 2008 Under Water Sensors Systems*, 574–579. https://doi.org/10.1109/SENSORCOMM.2008.13

Mi, Z., & Wei, G. (2019). A CoAP-Based Smartphone Proxy for Healthcare with IoT Technologies. *Proceedings of the IEEE International Conference on Software Engineering and Service Sciences, ICSESS, 2018-Novem*, 271–278. https://doi.org/10.1109/ICSESS.2018.8663785

Mijovic, S., Shehu, E., & Buratti, C. (2016). Comparing application layer protocols for the Internet of Things via experimentation. *2016 IEEE 2nd International Forum on Research and Technologies for Society and Industry Leveraging a Better Tomorrow, RTSI 2016*. https://doi.org/10.1109/RTSI.2016.7740559

Mohan, L., Jinesh, M. K., Bipin, K., Harikrishnan, P., & Sathyadevan, S. (2015). Implementation of scatternet in an intelligent IoT gateway. *Advances in Intelligent Systems and Computing*, *338*, 275–287. https://doi.org/10.1007/978-3-319-13731-5_31

Molisch, A. F., Balakrishnan, K., Chong, C. C., Emami, S., Fort, A., Karedal, J., Kunisch, J., Schantz, H., Schuster, U., & Siwiak, K. (2006). IEEE 802.15. 4a channel model-final report. *Ieee P*, *15*, 802.1504--0662. https://mentor.ieee.org/802.15/file/04/15-04-0662-00-004a-channel-model-final-report-r1.pdf

Moteiv Corporation. (2006). Moteiv: tmote sky low power wireless sensor module. In *Product Data Sheet*. https://doi.org/6020-0094-01 Rev. B

MQTT. (2014). *MQ Telemetry Transport*. https://mqtt.org

Naik, N. (2017). Choice of effective messaging protocols for IoT systems: MQTT, CoAP, AMQP and HTTP. *2017 IEEE International Symposium on Systems Engineering, ISSE 2017 - Proceedings*, 1–7. https://doi.org/10.1109/SysEng.2017.8088251

Nguyen, H. V., & Iacono, L. Lo. (2015). REST-ful CoAP Message Authentication. *2015 International Workshop on Secure Internet of Things (SIoT)*, 35–43.

https://doi.org/10.1109/SIOT.2015.8

Ning, H., & Wang, Z. (2011). Future internet of things architecture: Like mankind neural system or social organization framework? In *IEEE Communications Letters* (Vol. 15, Issue 4). https://doi.org/10.1109/LCOMM.2011.022411.110120

OASIS. (2012). *OASIS Advanced Message Queuing Protocol* (Issue October). docs.oasis-open.org/amqp/core/v1.0/os/amqp-core-complete-v1.0-os.pdf

Ojo, M. O., Giordano, S., Procissi, G., & Seitanidis, I. N. (2018). A Review of Low-End, Middle-End, and High-End Iot Devices. *IEEE Access*, *6*, 70528–70554. https://doi.org/10.1109/ACCESS.2018.2879615

OMNeT++. (n.d.). Retrieved July 8, 2021, from http://www.omnetpp.org/

Open Automation Software (2021). IoT Edge Computing Vs. Cloud Computing. Retrieved October 10, 2021 from https://openautomationsoftware.com/open-automation-systems-blog/iiot-edge-computing-vs-cloud-computing/

Osterlind, F., Dunkels, A., Eriksson, J., Finne, N., & Voigt, T. (2006). Cross-Level Sensor Network Simulation with COOJA. *Proceedings. 2006 31st IEEE Conference on Local Computer Networks*, 641–648. https://doi.org/10.1109/LCN.2006.322172

Panwar, M., & Kumar, A. (2015). Security for IoT: An effective DTLS with public certificates. *2015 International Conference on Advances in Computer Engineering and Applications*, 163–166. https://doi.org/10.1109/ICACEA.2015.7164688

Pena-Lopez, I. (2005). *ITU Internet Report 2005: The Internet ofThings* (Issue December).

Peng, W. C., Ko, Y. Z., & Lee, W. C. (2006). On mining moving patterns for object tracking sensor networks. *Proceedings - IEEE International Conference on Mobile Data Management*, *2006*, 41–44. https://doi.org/10.1109/MDM.2006.114

Perla, E., Catháin, A. Ó., Carbajo, R. S., Huggard, M., & Mc Goldrick, C. (2008). PowerTOSSIM z: Realistic energy modelling for wireless sensor network environments. *PM2HW2N'08 - Proceedings of the 3rd ACM International Workshop on Performance Monitoring, Measurement, and Evaluation of Heterogeneous Wireless and Wired Networks*, 35–42.

https://doi.org/10.1145/1454630.1454636

Piyare, R., & Lee, S. R. (2013). Towards Internet of Things (IOTS): Integration of Wireless Sensor Network to Cloud Services for Data Collection and Sharing. *International Journal of Computer Networks & Communications*, *5*(5), 59–72. https://doi.org/10.5121/ijcnc.2013.5505

Raza, S., Shafagh, H., Hewage, K., Hummen, R., & Voigt, T. (2013). Lithe: Lightweight Secure CoAP for the Internet of Things. *IEEE Sensors Journal*, *13*(10), 3711–3720. https://doi.org/10.1109/JSEN.2013.2277656

Raza, S., Trabalza, D., & Voigt, T. (2012). 6LoWPAN Compressed DTLS for CoAP. *2012 IEEE 8th International Conference on Distributed Computing in Sensor Systems*, 287–289. https://doi.org/10.1109/DCOSS.2012.55

Rescorla, E., & Modadugu, N. (2012). *Datagram Transport Layer Security Version 1.2*. https://doi.org/10.17487/rfc6347

Richardson, L., & Ruby, S. (2007). *RESTful Web Services* (1st ed.). O'Reilly Media, Inc.

Sachs, K., Appel, S., Kounev, S., & Buchmann, A. (2010). *Benchmarking Publish/Subscribe-Based Messaging Systems* (pp. 203–214). https://doi.org/10.1007/978-3-642-14589-6_21

Safaei, B., Monazzah, A. M. H., Bafroei, M. B., & Ejlali, A. (2018). Reliability side-effects in Internet of Things application layer protocols. *2017 2nd International Conference on System Reliability and Safety, ICSRS 2017, 2018-Janua*(December), 207–212. https://doi.org/10.1109/ICSRS.2017.8272822

Said, O., & Masud, M. (2013). Towards internet of things: Survey and future vision. *International Journal of Computer Networks*, *5*(1), 1–17. http://www.cscjournals.org/csc/manuscript/Journals/IJCN/volume5/Issue1/IJCN-265.pdf

Saint-Andre, P. (2011). *Extensible Messaging and Presence Protocol (XMPP): Address Format*. https://doi.org/10.17487/rfc6122

SATHYABAMA - Institute of Science and Technology (2021). Unit - II - Internet of Things - SCSA5301. Department of Computer Science and Engineering. Retrieved 20 October, 2021.

https://www.sathyabama.ac.in/sites/default/files/course-material/2020-11/U2.pdf

Schandy, J., Steinfeld, L., & Silveira, F. (2015). Average power consumption breakdown of wireless sensor network nodes using IPv6 over LLNs. *Proceedings - IEEE International Conference on Distributed Computing in Sensor Systems, DCOSS 2015*, *June*, 242–247. https://doi.org/10.1109/DCOSS.2015.37

Selavo, L., Wood, A., Cao, Q., Sookoor, T., Liu, H., Srinivasan, A., Wu, Y., Kang, W., Stankovic, J., Young, D., & Porter, J. (2007). LUSTER: Wireless sensor network for environmental research. *SenSys'07 - Proceedings of the 5th ACM Conference on Embedded Networked Sensor Systems*, 103–116. https://doi.org/10.1145/1322263.1322274

Sethi, P., & Sarangi, S. R. (2017). Internet of Things: Architectures, Protocols, and Applications. *Journal of Electrical and Computer Engineering*, *2017*. https://doi.org/10.1155/2017/9324035

Sharma, R., Prakash, S., & Roy, P. (2020). Methodology, Applications, and Challenges of WSN-IoT. *International Conference on Electrical and Electronics Engineering, ICE3 2020*, 502–507. https://doi.org/10.1109/ICE348803.2020.9122891

Shelby, Z. (2013). *Constrained Application Protocol (CoAP) draft-ietf-core-coap-17*. http://tools.ietf.org/html/draft-ietf-core-coap-17

Shelby, Zach, & Bormann, C. (2009). 6LoWPAN: The Wireless Embedded Internet. In *6LoWPAN: The Wireless Embedded Internet*. https://doi.org/10.1002/9780470686218

Sheng, Z., Yang, S., Yu, Y., Vasilakos, A., McCann, J., & Leung, K. (2013). A survey on the ietf protocol suite for the internet of things: Standards, challenges, and opportunities. *IEEE Wireless Communications*, *20*(6), 91–98. https://doi.org/10.1109/MWC.2013.6704479

Silva, Â. I. da. (2016). *MQTT-SN for Contiki*. Github. https://github.com/aignacio/mqtt-sn-contiki_example

Singh, C. P., Vyas, O. P., & Tiwari, M. K. (2008). A survey of simulation in sensor networks. *2008 International Conference on Computational Intelligence for Modelling Control and Automation, CIMCA 2008*, 867–872. https://doi.org/10.1109/CIMCA.2008.170

Song, Y., Zendra, O., & Zendra, O. (2016). *Using Cooja for WSN Simulations : Some New Uses and Limits*.

Stanford-Clark, A., & Truong, H. L. (2013a). MQTT for sensor networks (MQTT-SN) protocol specification. In *Ibm*. http://mqtt.org/new/wp-content/uploads/2009/06/MQTT-SN_spec_v1.2.pdf

Stanford-Clark, A., & Truong, H. L. (2013b). MQTT for sensor networks (MQTT-SN) protocol specification. *Ibm*, 28. http://mqtt.org/new/wp-content/uploads/2009/06/MQTT-SN_spec_v1.2.pdf

Stojmenovic, I., & Wen, S. (2014). The Fog computing paradigm: Scenarios and security issues. *2014 Federated Conference on Computer Science and Information Systems, FedCSIS 2014*, 1–8. https://doi.org/10.15439/2014F503

Sutaria, R., & Govindachari, R. (2013). Making sense of interoperability: Protocols and Standardization initiatives in IOT. *2nd International Workshop on Computing and Networking for Internet of Things (CoMNet-IoT) Held in Conjunction with 14th International Conference on Distributed Computing and Networking (ICDCN 2013)*, 2–5. http://rsutaria.net/wp-content/uploads/2013/02/Low_power_IoT_ComNet_2013_Mindtree.pdf

Tandale, U., Momin, B., & Seetharam, D. P. (2017). An empirical study of application layer protocols for IoT. *2017 International Conference on Energy, Communication, Data Analytics and Soft Computing (ICECDS)*, 2447–2451. https://doi.org/10.1109/ICECDS.2017.8389890

Tantitharanukul, N., Osathanunkul, K., Hantrakul, K., Pramokchon, P., & Khoenkaw, P. (2017). MQTT-Topics Management System for sharing of Open Data. *2017 International Conference on Digital Arts, Media and Technology (ICDAMT)*, 62–65. https://doi.org/10.1109/ICDAMT.2017.7904935

Tariq, M. A., Khan, M., Khan, M. T. R., & Kim, D. (2020). Enhancements and challenges in coap—a survey. *Sensors (Switzerland)*, *20*(21), 1–29. https://doi.org/10.3390/s20216391

Thangavel, D., Ma, X., Valera, A., Tan, H. X., & Tan, C. K. Y. (2014). Performance evaluation of MQTT and CoAP via a common middleware. *IEEE ISSNIP 2014 - 2014 IEEE 9th*

International Conference on Intelligent Sensors, Sensor Networks and Information Processing, Conference Proceedings, April, 21–24. https://doi.org/10.1109/ISSNIP.2014.6827678

Ugrenovic, D., & Gardasevic, G. (2016). CoAP protocol for Web-based monitoring in IoT healthcare applications. *2015 23rd Telecommunications Forum, TELFOR 2015*, 79–82. https://doi.org/10.1109/TELFOR.2015.7377418

Varadhan, N. T. F. and K. (2009). *"The ns manual", User's manual, UC Berkeley, LBL, USC/ISI, and Xerox PARC.*

Vasseur, J.-P., & Dunkels, A. (2010). Interconnecting Smart Objects with IP: The Next Internet. *Interconnecting Smart Objects with IP*, 335–351. http://www.sciencedirect.com/science/article/pii/B9780123751652000223

Vasseur, J, Agarwal, N., Hui, J., & Shelby, Z. (2011). RPL: The IP routing protocol designed for low power and lossy networks. In *Internet Protocol for*. http://www.academia.edu/download/35564908/RPL_The_IP_routing_protocol_designed_for_low_power_and__lossy_networks.pdf

Vasseur, JP, & Bertrand, C. (2010). A survey of several low power Link layers for IP Smart Objects. *... for Smart Objects (..., June*. http://www.ipso-alliance.org/wp-content/media/low_power_link_layer.pdf

Veeramanikandan, M., & Sankaranarayanan, S. (2017). Publish/subscribe broker based architecture for fog computing. *2017 International Conference on Energy, Communication, Data Analytics and Soft Computing (ICECDS)*, 1024–1026. https://doi.org/10.1109/ICECDS.2017.8389592

Venanzi, R., Kantarci, B., Foschini, L., & Bellavista, P. (2018). MQTT-Driven Node Discovery for Integrated IoT-Fog Settings Revisited: The Impact of Advertiser Dynamicity. *Proceedings - 12th IEEE International Symposium on Service-Oriented System Engineering, SOSE 2018 and 9th International Workshop on Joint Cloud Computing, JCC 2018*, 31–39. https://doi.org/10.1109/SOSE.2018.00013

Vermesan, O., Peter, F., Patrick, G., Sergio, G., Harald, Sundmaeker Alessandro, B., Ignacio Soler, J., Margaretha, M., Mark, H., Markus, E., & Pat, D. (2011). Internet of Things: Strategic Research Roadmap. In *Internet of Things-Global Technological and Societal Trends* (pp. 9–52). http://sintef.biz/upload/IKT/9022/CERP-IoT SRA_IoT_v11_pdf.pdf

Viswanathan, A., Sai Shibu, N. B., Rao, S. N., & Ramesh, M. V. (2018). Security Challenges in the Integration of IoT with WSN for Smart Grid Applications. *2017 IEEE International Conference on Computational Intelligence and Computing Research, ICCIC 2017*, 1–4. https://doi.org/10.1109/ICCIC.2017.8524233

Xu, Yiming, Mahendran, V., Guo, W., & Radhakrishnan, S. (2017). Fairness in fog networks: Achieving fair throughput performance in MQTT-based IoTs. *2017 14th IEEE Annual Consumer Communications & Networking Conference (CCNC)*, 191–196. https://doi.org/10.1109/CCNC.2017.7983104

Xu, Yingqi, & Lee, W. C. (2007). Compressing moving object trajectory in wireless sensor networks. *International Journal of Distributed Sensor Networks*, *3*(2), 151–174. https://doi.org/10.1080/15501320701204756

Yick, J., Mukherjee, B., & Ghosal, D. (2008). Wireless sensor network survey. *Computer Networks*, *52*(12), 2292–2330. https://doi.org/10.1016/j.comnet.2008.04.002

Yun, M., & Yuxin, B. (2010). Research on the architecture and key technology of Internet of Things (IoT) applied on smart grid. *2010 International Conference on Advances in Energy Engineering, ICAEE 2010*, 69–72. https://doi.org/10.1109/ICAEE.2010.5557611

APPENDICES

APPENDIX I: MQTT-CoAP Code for Contiki

```c
/*********************************************************************
 * @file mqtt-coap.c
 * @author Emmanuel N.
 * @date 5th December, 2019
 * @brief Main code to MQTT-COAP-SN hybrid protocol on Contiki-OS
 * @license This project is licensed by APACHE 2.0.
 ********************************************************************/

#include "contiki.h"
#include "contiki-net.h"
#include "rest.h"
#include "lib/random.h"
#include "clock.h"
#include "sys/ctimer.h"
#include "../core/net/uip.h"
#include "../core/net/uip-ds6.h"
#include "mqtt_sn.h"
#include "../core/net/rime.h"
#include "../core/net/uip.h"
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include "sys/energest.h"

// #define LOCAL_PORT 1884
// #define REMOTE_PORT 1883
#define PUBLISH_DELAY 10*CLOCK_SECOND
#define SUBSCRIBE_DELAY 5*CLOCK_SECOND

// #define NUMBER_OF_RESOURCES 10


static uint16_t udp_port = 1883;
static uint16_t keep_alive = 10;
static uint16_t broker_address[] = {0xaaaa, 0, 0, 0, 0, 0, 0, 0x0001};
static struct   etimer time_poll;
static struct   ctimer t_init;
static struct   ctimer t_publish;
static struct   ctimer t_subscribe;

static char     device_id[17];
static char     *topics_mqtt[] = {"/datagen"};
static uint8_t  qos = 1;
//static resource_t resources[NUMBER_OF_RESOURCES];
```

```
static mqtt_sn_con_t mqtt_sn_connection;
//uint8_t resources_pos = 0;
uint64_t current_time;
uint32_t data_size = 20;
uint32_t sent_count = 0;
uint32_t received_count = 0;

//Function declarations
void init_broker(void*);
//void temp_handler(REQUEST*, RESPONSE*);
//void humid_handler(REQUEST*, RESPONSE*);

void data_populate(char * data)
{
  size_t i;
  for (i = strlen(data); i < data_size-1; i++)
      { *(data+i) = (char)((sent_count%10)+31);
  }
  *(data + data_size-1) = 0;
}

/* Define Resource for data generation as used for simulation
*/ RESOURCE(data, METHOD_GET, "datagen");


void data_handler(REQUEST* request, RESPONSE* response)
{
  char cdata[data_size];

  sent_count++;

  sprintf(cdata, "ID:%lu\n", sent_count);

  data_populate(cdata);

  rest_set_response_payload(response,(uint8_t*)cdata,strlen(cdata));
  rest_set_header_content_type(response,APPLICATION_LINK_FORMAT);

  if(sent_count % 10 == 0) data_size += 10;
}

void init_mqtt_coap(uint8_t random_time)
{
  // Init Rest for CoAP
  rest_init();

  rest_activate_resource(&resource_data);

  // Init mqtt at a random interval to ease load on border router
  ctimer_set(&timer_init, random_time, init_broker, NULL);
}

void setup_coap_resources(method_t method, const char*url, const
void(*resource_handler))
{
  // resources[resources_pos] = (resource_t){NULL, method, url,
resource_handler, NULL, NULL, NULL};
  // resources_pos++;
}

void get_current_time()
{
  current_time = (uint64_t) clock_time();
}


void publish_callback(void *ptr)
{
  ctimer_reset(&timer_publish);

  get_current_time();

  sent_count++;

  char cdata[data_size];
  sprintf(cdata, "ID:%lu TxTime:%lu", sent_count,
  current_time); data_populate(cdata);
  mqtt_sn_pub(topics_mqtt[0], cdata, false, qos);

  printf("\n\n=====>>>>>Published == %s\nDatasize: %d bytes", cdata,
data_size);

  get_current_time();

  if(sent_count % 10 == 0) data_size += 10;
}


void subscribe_callback()
{
  if(mqtt_sn_check_status() == MQTTSN_TOPIC_REGISTERED)
  {
    size_t k;
    for(k = 0; k < ss(topics_mqtt); k++)
    {
      mqtt_sn_sub(topics_mqtt[i], qos); //Subscribe to topic with
certain QOS
    }

    init_sub();

    ctimer_stop(&timer_subscribe);
    ctimer_set(&timer_publish, PUBLISH_DELAY, publish_callback, NULL);
  }
  else ctimer_reset(&timer_subscribe);
}


void mqtt_sn_callback(char *topic, char *message){
  //get_current_time();
  uint64_t ct = (uint64_t) clock_time();
  uint64_t latency = ((ct - current_time)*1000L)/CLOCK_SECOND;
  uint8_t mesglen = strlen(message);
  printf("\n===== Message received: Topic:%s Message:%s \nRxTime:%lu
=====\n", topic, message, ct);
  printf("\nDatasize:%d bytes \nLatency:%lu\n", mesglen+1, latency);
  //printf("=== CLOCKSECONDS - %lu ===\n", CLOCK_SECOND);
}

void init_broker(void *ptr){
  sprintf(device_id,"%02X%02X%02X%02X%02X%02X%02X%02X",
          rimeaddr_node_addr.u8[0],rimeaddr_node_addr.u8[1],
          rimeaddr_node_addr.u8[2],rimeaddr_node_addr.u8[3],
          rimeaddr_node_addr.u8[4],rimeaddr_node_addr.u8[5],
          rimeaddr_node_addr.u8[6],rimeaddr_node_addr.u8[7]);

  mqtt_sn_connection.client_id     = device_id;
  mqtt_sn_connection.udp_port      = udp_port;
  mqtt_sn_connection.ipv6_broker   = broker_address;
  mqtt_sn_connection.keep_alive    = keep_alive;
  //mqtt_sn_connection.will_topic    = will_topic;   // Configure as
0x00 if you don't want to use
  //mqtt_sn_connection.will_message  = will_message; // Configure as
0x00 if you don't want to use
  mqtt_sn_connection.will_topic    = 0x00;
  mqtt_sn_connection.will_message  = 0x00;

  mqtt_sn_init();


mqtt_sn_create_sck(mqtt_sn_connection,topics_mqtt,ss(topics_mqtt),mqt
t _sn_callback);

  ctimer_stop(&timer_init);
  ctimer_set(&timer_subscribe, SUBSCRIBE_DELAY, subscribe_callback,
NULL);
}


/*------------------------------------------------------------------
-------*/
PROCESS(init_system_process, "[Contiki-OS] Initializing OS");
AUTOSTART_PROCESSES(&init_system_process);
/*------------------------------------------------------------------
-------*/

PROCESS_THREAD(init_system_process, ev, data) {
  // setup_coap_resources(METHOD_GET, "temp", temp_handler);
  // setup_coap_resources(METHOD_GET, "humid", humid_handler);

  static unsigned long
rx_start_time,lpm_start_time,cpu_start_time,tx_start_time = 0;
  static unsigned long
rx_new_time,lpm_new_time,cpu_new_time,tx_new_time = 0;

  PROCESS_BEGIN();

  debug_os("\nInitializing the MQTT_COAP_SN\n");

  uint32_t random_delay = CLOCK_SECOND * rimeaddr_node_addr.u8[7];

  init_mqtt_coap(random_delay);

  // Check sensor data at some interval
  // ctimer_set(&timer_sensor, SENSOR_CHECK_DELAY,
check_temp_humid_callback, NULL);
  energest_flush();

  rx_stime = energest_type_time(ENERGEST_TYPE_LISTEN);
  lpm_stime = energest_type_time(ENERGEST_TYPE_LPM);
  cpu_stime = energest_type_time(ENERGEST_TYPE_CPU);
  tx_stime = energest_type_time(ENERGEST_TYPE_TRANSMIT);

  etimer_set(&time_poll, 10*CLOCK_SECOND);

  while(1) {
      PROCESS_WAIT_EVENT();
      energest_flush();
      if (etimer_expired(&time_poll)){
        etimer_reset(&time_poll);
     //if(mqtt_sn_check_status() >= MQTTSN_CONNECTED){
       rx_new_time=energest_type_time(ENERGEST_TYPE_LISTEN);
       lpm_new_time=energest_type_time(ENERGEST_TYPE_LPM);
       cpu_new_time=energest_type_time(ENERGEST_TYPE_CPU);
       tx_new_time=energest_type_time(ENERGEST_TYPE_TRANSMIT);

       printf("Time spent (usec) rx: %lu, tx: %lu, cpu: %lu, lpm:
%lu\n",
               (unsigned long)(1e6*(rx_new_time - rx_stime)
/ RTIMER_SECOND),
               (unsigned long)(1e6*(tx_new_time - tx_stime)
/ RTIMER_SECOND),
               (unsigned long)(1e6*(cpu_new_time - cpu_stime)
/ RTIMER_SECOND),
               (unsigned long)(1e6*(lpm_new_time - lpm_stime)
/ RTIMER_SECOND));


       printf("Total Energy: %lu uJ\n",
                 (unsigned long)( (21800*3*(rx_new_time -
rx_start_time) / RTIMER_SECOND) +
                 (19500*3*(tx_new_time - tx_start_time) /
RTIMER_SECOND) +
                 (1800*3 *(cpu_new_time - cpu_start_time) /
RTIMER_SECOND) +
                 (2.6*3 *(lpm_new_time - lpm_start_time)
/ RTIMER_SECOND) ) );

       rx_stime = energest_type_time(ENERGEST_TYPE_LISTEN);
       lpm_stime = energest_type_time(ENERGEST_TYPE_LPM);
       cpu_stime = energest_type_time(ENERGEST_TYPE_CPU);
       tx_stime = energest_type_time(ENERGEST_TYPE_TRANSMIT);
     //}
      }
  }
  PROCESS_END();
}
```