AN EXPERIMENTAL COMPARISON OF MESSAGING PROTOCOLS MQTT AND COAP

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF INFORMATICS OF
THE MIDDLE EAST TECHNICAL UNIVERSITY
BY

HASAN FARUK ÇOBAN

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE
OF MASTER OF SCIENCE
IN
THE DEPARTMENT OF INFORMATION SYSTEMS

JUNE 2017

An Experimental Comparison of Messaging Protocols MQTT and COAP

Submitted by HASAN FARUK ÇOBAN in partial fulfillment of the requirements for the degree of **Master of Science in The Department of Information Systems Middle East Technical University** by,

Prof. Dr. Deniz Zeyrek Bozşahin _____
Director, Graduate School of Informatics

Prof. Dr. Yasemin Yardımcı Çetin _____
Head of Department, Information Systems

Assoc. Prof. Dr. Aysu Betin Can _____
Supervisor, Information Systems

Examining Committee Members:

Assoc. Prof. Dr. Altan Koçyiğit _____
Information Systems, Middle East Technical University

Assoc. Prof. Dr. Aysu Betin Can _____
Information Systems, Middle East Technical University

Assist Prof. Dr. Erhan Eren _____
Information Systems, Middle East Technical University

Assist Prof. Dr. Çağdaş Gerede _____
Computer Engineering, TOBB University of Economy and Technology

Assoc. Prof. Dr. Alptekin Temizel _____
Modelling and Simulation, Middle East Technical University

Date: _____

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this wok.

Name, Last name: Hasan Faruk ÇOBAN

Signature : _____

ABSTRACT

An Experimental Comparison of Messaging Protocols MQTT and COAP

Çoban, Hasan Faruk

MSc., Department of Information Systems

Supervisor: Assoc. Dr. Aysu Betin Can

June 2017, 57 pages

As the attention towards to Internet of Things (IoT) increases recently, the need for the infrastructure that carries the communication between nodes, which have limited resources, also increases. The network beneath applications has direct effect on resilience of IoT environments. Due to the advances on mobile devices in terms of more powerful hardware, developers focused on mobile applications. However, solid network structures are needed for these applications. To match these needs several protocols are introduced. MQTT (Message Queue Telemetry Transport) and COAP (Constrained Application Protocol) are the most popular among messaging protocols. Although there are studies comparing these two protocols, they mainly focus on the network perspective. They cover mostly network traffic and load that protocols put on networks. Resource usages, especially on the nodes, are not examined thoroughly. Those studies only cover minimal traits and their test beds are minimalistic environments when it comes to investigate node's resource usages. Many applications need network communications. MQTT and COAP are possible candidates for networking. The amount of resource the protocols use might be the decisive factor.

In this thesis a comparison between two prominent messaging protocols on common hardware and software setup is aimed. MQTT and COAP are compared under the metrics of energy consumption, memory and CPU resource usages, transfer delays and adaptation capabilities. In this study HL7 messages have been used as a data type in order to place a healthcare context in experiments.

Key words: MQTT, COAP, Lightweight Messaging Protocols

ÖZ

MQTT ve COAP Mesajlaşma Protokollerinin Deneysel Bir Karşılaştırması

Çoban, Hasan Faruk
Yüksek Lisans, Bilişim Sistemleri Bölümü
Tez Yöneticisi: Doç. Dr. Aysu Betin Can

Haziran 2017, 57 sayfa

Son yıllarda Nesnelerin İnterneti'ne (IoT) olan ilginin artmasıyla sınırlı kaynaklara sahip düğümler arasındaki iletişimi sağlayan altyapıya duyulan ihtiyaç artmaktadır. Uygulamalar altındaki Ağ, IoT ortamlarının esnekliği üzerinde doğrudan etkiye sahiptir. Mobil cihazlardaki gelişmelere bağlı olarak (örneğin daha güçlü donanım), geliştiriciler mobil uygulamalara odaklandı. Bununla birlikte, dayanıklı ağ yapıları bu uygulamalar için gereklidir. Bu ihtiyaçları karşılamak için çeşitli protokoller getirilmiştir. Mesajlaşma protokolleri arasında MQTT (Message Queue Telemetry transport) ve COAP (Sınırlandırılmış Uygulama Protokolü) en popüler olanlarıdır. Bu iki protokolü karşılaştıran çalışmalar olsa da, bu çalışmalar çoğunlukla ağ trafiğini ve yükünü kapsamaktadır. Kaynak kullanımları, özellikle de nodlardaki kullanımlar yeterince araştırılmamıştır. Bu çalışmalar yalnızca baz seviyede özellikleri kapsamış ve nodların kaynak kullanımlarını araştırma noktasında ise minimalist düzeneklerde çalışılmıştır. Birçok uygulama ağ iletişimi gerektirir. MQTT ve COAP, ağ için olası adaylardır. Protokollerin tercihinde ne kadar kaynak kullandıkları karar verici etken olabilir.

Bu tez çalışmasında; aynı donanım ve yazılım kurulumu üzerindeki iki önde gelen mesajlaşma protokolü arasındaki karşılaştırmanın yapılması amaçlanmaktadır. MQTT ve COAP, enerji tüketimi, bellek ve işlemci kaynak kullanımları, aktarım gecikmeleri ve uyarlama yetenekleri ölçütleriyle karşılaştırılacaktır. Bu çalışmada, deneylerde bir sağlık bağlamı yerleştirmek için HL7 mesajları veri türü olarak kullanılmıştır.

Anahtar Sözcükler: MQTT, COAP, Mesajlaşma Protokolleri

To My Son, Yiğit

ACKNOWLEDGMENTS

First of all, I would like to express my gratitude to my thesis advisor Aysu Betin CAN for her patience and support.

Besides my supervisor, I would like to thank my friends Gürkan SOLMAZ and Onur ÇAM for their support and encouragement

Finally, I would like to express my gratitude to my parents, Ayşe-Mithat ÇOBAN.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF EQUATIONS

LIST OF ABBREVIATIONS

ACK	Acknowledgement
ADT	Admittance-Discharge-Transfer
AMQP	Advanced Message Queuing Protocol
AVG	Average
COAP	Constrained Application Protocol
CON	Confirmable
CoRE	Constrained RESTful Environments
CPU	Central Processing Unit
DFT	Detail-Financial-Transaction
GQM	Goal-Question-Metric
HL7	Health Level Seven
HTTP	Hyper Text Transfer Protocol
IETF	Internet Engineering Task Force
IoT	Internet of Things
MED	Median
MQTT	Message Queue Telemetry Transport
NON	Non-Confirmable
ORM	Order-Messages
ORU	Observation-Result-Update
PC	Personal Computer
QoS	Quality of Service
REST	Representational State Transfer
RQ	Research Question
RTT	Round Trip Time
RTT	Round Trip Time
S.DEV	Standard Deviation
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
UPS	Uninterruptable Power Source
WI-FI	Wireless Fidelis
XMPP	Extensible Messaging and Presence Protocol

CHAPTER 1

INTRODUCTION

Every day, advances on computer science provide new concepts to developers. Networks connect nodes which are capable of conducting processes. Traditionally those nodes were few in numbers and managing them is rather easy. However, processors are constantly getting cheaper and nodes' sizes constantly keep getting smaller. This enables number of nodes boosted and created new need for intercommunication between them. Anything can be a node and everything may try to communicate. This concept evolved into "Internet of Things". IoT's name is first introduced in 1999 by Kevin Ashton (Ashton, 2009). It could be foreseen that in 2020 IOT objects multiples human population at least five times (Evans, 2012).

Smart phones are important member of this newly defined society. They are developed and supported by major corporations. Also, massive numbers of developers are currently working on this environment which depicts a promising future. Development possibilities also bring serious challenges. Traditional network protocols are neither designed nor optimized for supporting pervasive network infrastructures (W. Colitti, 2011). Several lightweight protocols have been developed since then. Two of them are distinguishing themselves. The Message Queuing Telemetry Transport (MQTT) (Locke, 2010), designed by IBM, and the Constrained Application Protocol (COAP) (Z. Shelby C. K., 2013), designed by the Internet Engineering Task Force (IETF). Although they implement different approaches, both serve same need: the lightweight Machine-to-Machine communication.

This thesis focuses on the comparison between MQTT and COAP in terms of energy consumption, resource usage, transfer delays and adaptation capabilities. Providing the same hardware and software test bed their performances are compared on certain metrics in these aspects. To evaluate their respective performance metrics, the identical environment setup is crucial. The aim of this study is after comparison of these two protocols, to discuss their respective advantages over each other and to propose suitable application type of these protocols. The context dictates which protocol would be better choice to implement. Their strengths and weaknesses are only meaningful by the context.

There are a number of research studies on lightweight network protocols. Mostly, these studies tend to focus on network layer metrics such as network trafficking and bandwidth. Other than bandwidth, reliability and energy efficiency metrics are also investigated. The protocols are tested on wired, wireless and cellular networks. Test beds vary from single PC setup to a PC and multiple mobile devices. According to these comparison studies in the literature, in terms of network traffic and bandwidth aspect, COAP is better choice than MQTT (D.Thangavel, 2014) (S. Bandyopadhyay,

2013). Also COAP fared better in message delays (D.Thangavel, 2014) (S. Bandyopadhyay, 2013). When it comes to reliability MQTT is favored by majority of literature (D.Thangavel, 2014) (M. Tucic, 2014) (D. Yi, 2016). COAP is claimed to be more energy efficient than MQTT (S. Bandyopadhyay, 2013), however it is a debatable subject because of deficiencies in test setup (S. Bandyopadhyay, 2013).

In this thesis, we choose to focus on nodes. How protocols affect nodes of network is the primary question. Since IoT is gaining popularity and mobile devices are becoming widespread; MQTT and COAP should be analyzed under wireless networks with mobile devices. While most studies establish common hardware infrastructure, they fail to provide common software infrastructure. In order to reach accurate comparison information middleware should be identical (D.Thangavel, 2014). Mobile devices are resource constraint devices, so resource usages should be investigated between MQTT and COAP. When investigating message delays researchers usually ignore protocols have communication patterns. MQTT uses publish/subscribe and COAP uses request/response. In MQTT delay only matters in message deliver from server to client, however in COAP it matters entire round trip time. Security is not concern of this study. Further literature review analysis is explained in chapter 2. In this thesis, we aim to compare these protocols under common infrastructure in private wireless network on resource usage, delay and power consumption aspects.

To conduct disciplined research, we chose Goal-Question-Metric (GQM) methodology. GQM is a paradigm which proposes measurements should be defined in top-down fashion (V.R.Basili, 1994). GQM approach is suitable for studies which have to deal with quantified information. Because it helps to break down implementation of the study to three levels: (V.R.Basili, 1994)

- Conceptual Level(GOAL)
- Operational Level(Question)
- Quantitive Level (Metric)

These levels force the implementation of study under structural hierarchy and draw outline for tests and development. Since our study depends heavily on experiments and tests, we need to organize our experiments. Goal-Question-Metric methodology is suitable for arranging such study. We decided a goal, generated research questions to reach it and metrics to answer our RQ's. We built this methodology in the context of HL7. HL7 is data transfer standard for medical organizations. It provides message standard for our tests and a base of analysis to evaluate our findings. Another benefit of HL7 context is to propose possible communication system for medical facilities. Detailed explanation about methodology is placed in chapter 3.

After deciding methodology and context, we setup our test bed and developed experiment scenarios. We composed a set of HL7 messages to simulate context. Test scenarios were organized to reflect medical facilities' network load. Detailed explanation about our test scenarios are placed in chapter 4.

Our testbed is comprised of network, hardware and software components. Our network setup is private network provided by Wi-Fi router. The Test bed consists of hardware and software components. The hardware setup is formed with a PC, an Android Device and a router. We developed Server and Client applications for both protocols. Since the aim of this study is to compare MQTT and COAP on common middleware, we setup entire test bed as identical as possible. The same hardware is used for both test runs. Client and server applications are developed on the same frameworks. We developed server applications on .Net Framework and client applications on Android Framework. Other than the protocol implementations, the applications developed are identical. To sum up, we provide common middleware for both protocol experiments. Further information related to middleware is located in chapter 4.

The rest of the paper is organized as follows. Chapter II provides a description of MQTT and COAP. Chapter III discusses on the related work. Chapter IV provides a qualitative comparison of the two protocols. Chapter V reports on the experimental study and discusses. Chapter VI discusses the results and draws the conclusions.

CHAPTER 2

BACKGROUND & LITERATURE REVIEW

This chapter contains brief information about communication protocols, MQTT and COAP, the summary of various related studies and explanation about thesis context.

2.1. Communication Protocols

In this section a brief introduction to MQTT and COAP is given.

2.1.1. MQTT

The Message Queuing Telemetry Transport was developed by IBM. It is designed for devices which have hardware constraints, especially for embedded devices and microcontrollers. The protocol overhead (fixed-length header of 2 bytes) makes the MQTT viable solutions for networks with restricted resources, such as low bandwidth and high-latency (Locke, 2010). This protocol centered on the broker and publish/subscribe pattern. (Edielson P. Frigieri, 2015). A Broker directs telemetry messages between nodes which can be a publisher and a subscriber. A Broker can also support one-to-many communications. Topic mechanism can be used as filters between publishers and subscribers. MQTT is implemented over TCP which allows reliable communication on even unreliable networks. Implementation is centered on a broker which routs messages via topics. Each client receives messages from subscribed topics. This design of architecture enables isolation of clients. They do not need to know each other to get messages. Main difference between HTTP and MQTT is that a client does not have to pull the information it needs, but the broker pushes the information to the client. MQTT provides a lossless stream in an orderly fashion between clients and server (Locke, 2010). The stream can be in both directions.

In MQTT there are three options for message delivery. They provide different level of reliability.

- Quality of Service 0 (QoS0): In this level, messages are sent without any acknowledgement mechanism. It is known as “At Most Once” because of the nature of transfers. In this level messages are only sent once. (Locke, 2010)
- Quality of Service 1 (QoS1): “At Least Once”, this level provides each messages are delivered at least once. Confirmation messages are expected. (Locke, 2010)
- Quality of Service 2 (QoS2): “Exactly Once” is the most reliable service level MQTT has to offer. A Four-Way Handshake mechanism is used. (D.Thangavel, 2014) It guarantees a message is delivered exactly once. It is useful for eliminating duplicate messages.

Since we are comparing the two protocols in the context of healthcare systems, in which reliability is essential. The QoS2 “Exactly Once” feature of MQTT is used in this thesis.

Table 1: MQTT Message Structure (Locke, 2010)

0	4	5	7	8
MQTT Control Packet Type		Dup Flag	QoS level	Retain
Remaining Length				
Payload				

Table 1 shows the format of an MQTT message. An MQTT message starts with a 2 bytes fixed-length header followed by an optional message specific variable length message header and a message payload in this order. The 2 bytes header contains the MQTT control packet type, flags including the flag for quality of service, and remaining length indicating the bytes remaining within the current package. The minimum packet size is 2 bytes which is the fixed length header. MQTT supports maximum package size of 256 MB (Locke, 2010).

2.1.2. *COAP*

The COAP protocol was designed by the Constrained RESTful Environments (CoRE) Working Group of Internet Engineering Task Force (IETF). It is an adaptation of HTTP for devices with limited power and processing capabilities (Z. Shelby K. H., 2014). It runs over UDP and based on REST architecture. Since it was an adaptation of HTTP, COAP is implemented over request/response pattern. The main difference of COAP from HTTP is its reduced overhead in packages and data exchange between clients and servers. Those benefits are the result of UDP usage on request/response pattern. Hence, it does not have congestion control as in TCP (Chen, 2014). Since UDP does not provide reliability, COAP has two mechanisms to overcome it. They are retransmission mechanism and resource discovery mechanism (Chen, 2014).

COAP’s message layer supports four types message: CON (confirmable), NON (non-confirmable), ACK (Acknowledgement), RST (Reset). These messages are used in implementation of transport types: (Z. Shelby C. K., 2013)

- Reliable Message Transport: COAP implements HTTP interface, so it uses HTTP methods. GET is the most used HTTP method. In COAP, GET messages stand for messages dispatch with get method. For each GET message received, a CON message is sent. It keeps retransmissions until ACK message is arrived. In any case of failure of processing messages, receiver sends RST message. Since UDP does not provide such capabilities, it is handled within HTTP implementation (Z. Shelby C. K., 2013). Figure 1 shows a reliable transport.

- Unreliable message transport: Messages are transferred with NON type message. No ACK or RST response is required.

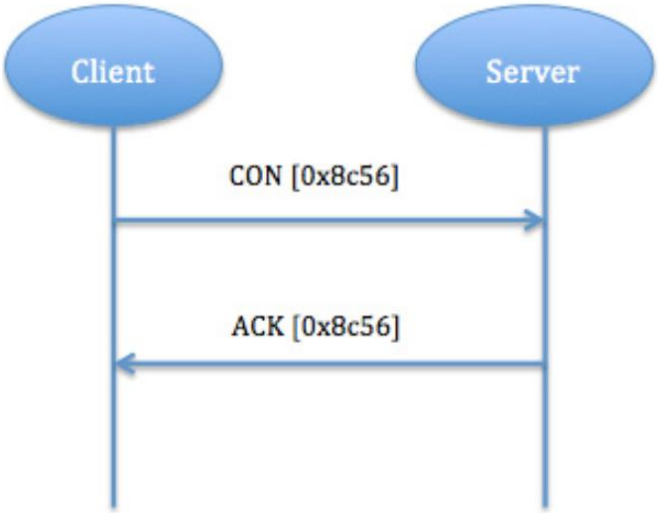

Figure 1: COAP Reliable Transport Implementation (Chen, 2014)

In this thesis, GET messages with CON property are used. COAP is designed as "Stop and Wait" when encounters packet loss. Retransmission mechanism tries to deliver messages. COAP message format, which is defined in the Internet Standards Document RFC 7252 (Z. Shelby K. H., 2014) is shown in Table 2. The messages are encoded in binary format.

Table 2: COAP Message Structure (Z. Shelby C. K., 2013)

0 2	4	8	16	32
Ver	T	OC	Code	MessageID
Token				
Options				
Payload				

A CoAP message starts with a fixed 4 bytes header consisting of version (Ver), type (T), token length (TKL), message code (Code) representing request or a success for request or an error, and message ID used for detecting message duplication and matching messages of type ACK/Reset to Confirmable/Nonconfirmable. The header is followed by a 0-8 byte optional token value. The token field is followed by zero or more COAP options and payload. COAP defines 1152 Bytes as the maximum message size (Z. Shelby K. H., 2014).

2.2. Related Studies

In this section we present the related work in two aspects: the studies comparing MQTT and COAP, and the studies employing these messaging protocols in healthcare domain.

In 2014 study of Thangavel et al. (D.Thangavel, 2014) a comparison of MQTT and COAP is under equal circumstances is presented. They design a common middleware which is supported by common programming interface. The researchers intended to compare two protocols under the same test environment. In their setup, the hardware consists of a laptop, a Beagleboard and a switch. Software used was Mosquitto (Light, 2017) for MQTT, Libcoap (O.Bergmann, 2012) for COAP and Wireshark for network monitoring. In their experiments, it was seen that the two protocols transmitted the messages 100% success. So they simulate packet loss and packet size to compare those two protocols with respect to delay and bandwidth usage. According to findings, on delay aspect MQTT performs better at low packet losses, COAP at higher packet losses. On bandwidth usage aspect, COAP fares better regardless of packet loss rate except for big packet size. As packet size grows UDP, which is used by COAP, loses more message causing retransmission. That situation leads to higher bandwidth usage than MQTT according to their findings. However, their paper does not cover followings. First, they focused on delay and bandwidth usage. Those findings are acquired by wired transmission via a switch. Nowadays wireless communication is rising and this trend will continue for the foreseeable future. Also, they do not take the node's (in this case beagleboard) resource usage such as memory, CPU and energy consumption into consideration. Adding that, there is no indication or measurement for mobile nodes which are integral part of Internet of Things.

The study by Bandyopadhyay et al. (S. Bandyopadhyay, 2013) has quite similar approach to the problem as Ref (D.Thangavel, 2014). Under the same hardware (Laptop and netbook) and software (Mosquitto, Libcoap and Wireshark), they compared these two protocols with respect to energy consumption, bandwidth usage and reliability. According to their findings, MQTT consumes more bandwidth and energy. Both protocols are proven to be reliable under closed protected network conditions. Deficiencies are quite similar with Ref (D.Thangavel, 2014). It does not cover wireless networks and resource usage difference between protocols. Again, there is no investigation about mobile nodes.

Tucic et al. (M. Tucic, 2014) provide us with insight of a network layer comparison between the respective protocols. It is a survey study and it does not present any experimental results; it gives valuable information about how Machine-to-Machine (M2M) communication with multiple remote nodes. It also provides basic information about these protocols' features and capabilities. According to the authors due to its broker mechanism and publish-subscribe pattern MQTT is the most suitable M2M protocols for closed local network layers (M. Tucic, 2014). However, they do not perform any experiment to support their claims.

Dürkop et al. (L.Dürkop, 2015) focus on mainly cellular networks such as Edge, UMTS and LTE and address performance comparison of popular lightweight

networks such as COAP, MQTT and OPC UA. They mainly compare network trafficking of these protocols. They do not conduct any research about nodes, but only focus on network throughput under EDGE, UMTS and LTE environments. The idea of using mobile devices as node is derived from this paper.

Karagiannis et al. (V. Karagiannis, 2015) did a survey study on the application layer protocols that includes MQTT and COAP. They focus on architectural level point of view and provide information about MQTT, COAP, The Extensible Messaging and Presence Protocol (XMPP), Restful Services, The Advanced Message Queuing Protocol (AMQP) and Web Sockets. They conclude with COAP is the most lightweight protocol due to its UDP nature, but MQTT is the most energy efficient one. Also they claim publish/subscribe pattern is more suitable for IoT. Deficiency of their paper is pure qualitative one without any results to support their claims.

De Caro et al. (N. De Caro, 2013) consider MQTT and COAP as sound alternatives for smartphone based sensor networks. They claim HTTP and XMPP are not suitable for pervasive network. Lightweight messaging protocols such as MQTT and COAP are more suitable for transmitting smartphone sensor data. They compare both protocols in qualitative and quantitative aspects. According to their study, MQTT is more appropriate in a qualitative aspect. Data-centric design, congestion control ability and flexibility features put MQTT before COAP. However, in quantitative aspect, COAP fares better than MQTT in bandwidth usage and round trip time (RTT). The authors fail to include mobile devices' resource constraints into their qualitive and quantitive aspects.

Sutaria et al. (R. Sutaria, 2013) address standardization issues on IoT and emphasize importance interoperability between different protocols. Since HTTP is largely used in current network applications and web, they think COAP would be attractive alternative for IoT applications. COAP has similar implementation with HTTP and suits better with constraint devices. Interoperability between these two protocols is easy task according to Sutaria et al. They also consider application level interoperability between MQTT and COAP. Although there is no provided experimental result, they claim the best way to interoperate between MQTT and COAP is common gateways which can operate in protocol layer.

In their study Kim et al. (W. Kim, 2015) try to achieve complete IoT service assisted by smart phones. They consider COAP, MQTT and XMPP for possible candidates to their proposed architecture. They develop infrastructure to provide services to HTTP, COAP and MQTT based application platforms. Although, their paper only provides for a thermostat system over HTTP based communication, the idea of interoperable IoT system is noteworthy.

Cohn (Cohn, 2011) compares AMQP and MQTT in his white paper. According to him, AMQP has wider range of use and more complete messaging protocol than MQTT. He states that, AMQP and MQTT has divergent intended of uses. MQTT targeted for small devices and small messages on low-bandwidth networks, but AMQP is designed for rather bigger and comprehended systems. Also, AMQP provides extra functionalities especially on security aspect. However, he fails to see these two protocols are diverged with respect to their usage targets. Although,

AMQP is more powerful, it also required powerful clients. MQTT on the other hand, requires far less resources and bandwidth (S. Hamida, 2015).

In this study, a comparison of MQTT and COAP through common infrastructure idea is influenced by Ref (D.Thangavel, 2014). Using mobile devices idea is derived from Ref (L.Dürkop, 2015), Ref (N. De Caro, 2013) and Ref (W. Kim, 2015). The metrics used in this thesis study are determined by combined analysis of these papers. Since network trafficking and bandwidth usage of protocols are largely studied, in this thesis we focus on usage of nodes' hardware resources. After deciding mobile devices as nodes of network which is used to compare COAP and MQTT; metrics are decided by weaknesses of mobile devices which are battery, CPU usage and memory usage. Delay is also added to these metrics because of their difference of design. The atomic transaction of MQTT begins with server publishing packet and ends with client acquire it. However, in COAP transaction begins with client request packet, server sends it and finally client acquires it. This difference between data flows is also investigated.

2.2.1 *Use of MQTT and COAP in Healthcare*

Healthcare systems have to be reliable by its nature of the domain; hence many researchers automatically assume TCP is indispensable part of such systems. However, COAP brings reliability features over UDP. Khattak et al. (H. Khattak, 2014) make a survey study based on COAP-based healthcare implementation. According to their study, COAP can be easily integrated to Internet, because of its HTTP-based RESTful architecture. They claim COAP might be viable option for healthcare systems. There are several deficiencies of this paper. First, they do not conduct any experiment on this issue and provide any result. Second, there is no comprehensive analysis with respect to reliability. This study only focuses on COAP might be practical implementation for devices which are used in patient monitoring.

Another study related COAP-based healthcare application is conducted by Cha et al. (M. Cha, 2017). In this study, Cha et al. design a healthcare monitoring system. In their implementation, data acquired from sensors are transmitted to android based mobile devices via COAP. Linux-based gateways are responsible for transmission between sensor and mobile devices. Three elements (sensor, gateway, mobile device) of their setup communicate with each other request/response based COAP implementation. However, they do not provide further information other than architectural design.

Hamida et al. (S. Hamida, 2015) focus on designing a mobile health framework. They plan to use Wireless Body Area Networks (WBANs). They consider COAP, MQTT, MQTT-SN and AMQP as suitable candidates and compare these protocols for designing a framework in their study. Message overhead and delay are their performance metrics. According to their study AMQP has the biggest overhead and the highest delay. COAP has smaller overhead and faster than MQTT. Finally, they conclude MQTT is better choice but COAP is attractive alternative with its low-overhead and low-latency.

Yi et al (D. Yi, Design and Implementation of Mobile Health Monitoring System based on MQTT Protocol, 2016) suggest architecture for healthcare services. The

proposed architecture uses MQTT for local communications. They choose MQTT for its low bandwidth, reliability, limited resource needs. Although, their primary goal was to develop a complete system design for healthcare monitoring applications, MQTT related test results could be a reference to our study. Also, their local area network setup is similar to our setup: Android client operates on Wi-Fi network where communication is handled by MQTT protocol. They only use MQTT but test all QoS levels for their monitoring data. Results are in expected value where delay is sorted in descending order from Qos2, Qos1 and QoS0. Since QoS2 has more control mechanism in its design, it has bigger delay than other QoS levels.

2.3. HL7

There are many possible apply areas of IoT. Healthcare applications, manufacturing management, automotive industry, traffic automation systems, and media and entertainment sectors are potential candidates (D.Soldani, 2015). For this thesis, the most applicable context is Healthcare. Manufacturing management, automotive industry, traffic infrastructure, media and entertainment sectors are heavily commercialized areas where all big corporations position themselves dominantly, hence all developments are hidden because they are thought as commercial secrets. In traffic infrastructure there is no substantial progress or accepted standards. On the other hand, healthcare has public and established standards. Therefore, we have chosen healthcare as a context. To simulate healthcare system, we transport Health Level Seven (HL7) messages over MQTT and COAP.

HL7 is international data transfer standard for medical organizations. It is developed by the Health Level Seven International. Since medicine is a collaborative work across different areas of expertise, the communication between them is an important aspect of health and efficiency of knowledge sharing. Interoperability between different facilities also has equal importance. These requirements cause the need for standards and it produces HL7. HL7 provides communication platform above hardware and software infrastructure for hospitals and other healthcare organizations. These organizations can easily share clinical information between them regardless of their difference of healthcare, billing, and patient tracking systems. Modern medical information management is a knowledge intensive activity requiring a high degree of interoperability across various health management entities (B. Orguna, 2006). As Orguna (B. Orguna, 2006) suggests the aim is interoperability and HL7 provides that. Health Level 7 standards define and provide common workspace for data definitions, data exchange, diagnosis support, personal health records, documentations and labels. HL7 has several standards, methodologies and guidelines. In this thesis we have used Version 2.5.1 standard which is the most widely known standard. HL7 V2 messages are used for compare respective protocols. There four main message types (HL7, 2000):

Patient Administration (ADT): Admittance (A), Discharge (D) and Transfer (T) messages are, as their name suggest, the patient information related information carrier messages. Patient personal information (name, age, insurance etc.) and status of patient (admit, transfer, registration etc.) are relayed via ADT messages.

Orders (ORM): Order (OR) messages (M) are used for transmitting information about an order. ORM messages also relay orders statuses such as new orders, cancellations, information updates, and discontinuation.

Results (ORU): Observation (O) Result (R) and Update (U) messages transmits observation and results from various treatment and analysis instrument such as EKG, Clinical lab results, Imaging study reports, Patient condition or other data (i.e. vital signs, symptoms, allergies, notes, etc.). It may also be used to transmit result data from the producing system to a medical record archival system or to another system not part of the original order process. ORU messages are sometimes used to register or link to clinical trials or for medical reporting purposes for drugs and devices as well.

Charges (DFT): Detail (D) financial (F) transaction (T) messages contain information about patients' billing and accounting data. DFT includes charges, deposits and carries them between clinical and billing systems.

CHAPTER 3

METHODOLOGY

We follow Goal-Question-Metric (GQM) methodology for our thesis investigation. It provides us traceable link and action steps between aim of the study and experiments. In other words, it helps to sort our experiments in reasonable structural integrity. Also, it provides contextual order for examination of the experiment results.

3.1. Goal

The goal of this thesis is to determine strengths and weaknesses of MQTT and COAP protocols on application level. After stating these strengths and weaknesses, possible usage areas are suggested. To achieve this goal several research questions (RQs) are developed. Measurable information is the key for answering research questions. To acquire the measurable information, we need metrics. Hence, first we decided RQ's; then the metrics. We have planned to run series of experiments to acquire metric results. After acquiring the results which are measurable information to us; we sought to answer RQs.

Research questions are listed below:

- RQ1: Which protocol is resource efficient for mobile hardware aspect?
- RQ2: Which protocol performs better for transfer time aspect?

For each RQ, we determined a set of metrics to provide a measurable data to answer the question.

3.2. Research Questions

We choose three aspects and develop three research questions (RQ) related to them. Those are data, resource and performance aspects.

3.2.1.RQ1:Resource Aspect
In this thesis we focus protocols on application level. Specifically, we focused on applications running on a node. In the test environment, this node is a mobile device. The application, which contains the implementation of respective protocols, uses the resources of the node. The usages of these resources are classified as metrics. Three metrics are chosen as shown in Figure 2:

Central Processing Unit (CPU) Time: Every application has to use CPU Allocation for execution. In other words, every application uses CPU resource while it is running. So application's usage of CPU Allocation is important metric for this thesis.

Memory Usage: Mobile devices share its memory between applications. Hence, memory usage is distinctive quality between applications.

Battery Usage: Battery is the biggest bottleneck for mobile devices. Great number of research studies is conducted to increase battery life of these devices. Application's energy usage is a significant criterion.

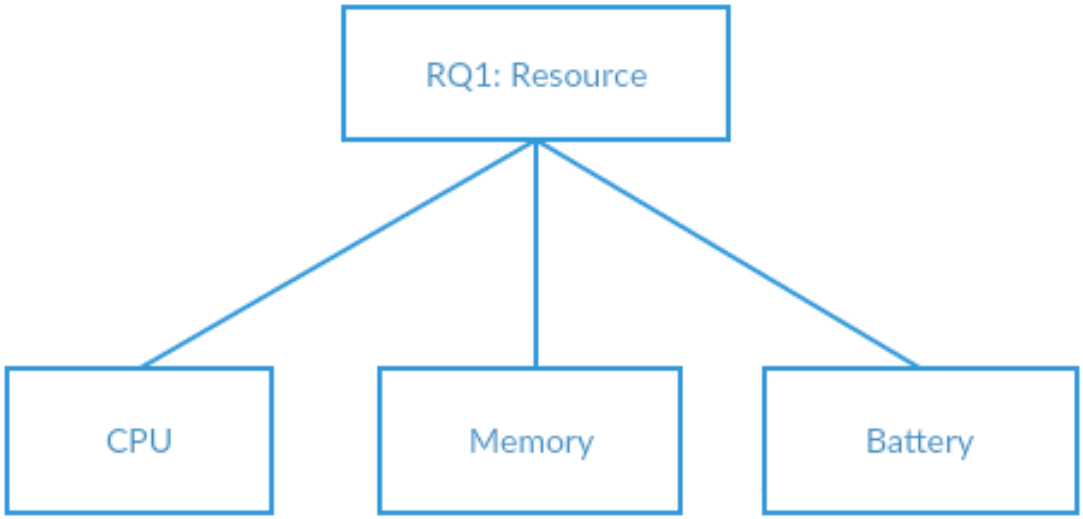

Figure 2: RQ1 and Metrics

3.2.2.RQ2:Performance Aspect

When two protocols are compared, it is inevitable to measure their packet delivery performances. This aspect is analyzed in the literature, as discussed in Chapter 2, as well. In this thesis we measured **delay** metrics on application level. Other papers monitored network via Wireshark software (G.Combs, 2007). In this study, delay metric is considered as atomic packet delivery time. We define our delay metric as the time passed between a message become available at server and its arrival to client. COAP and MQTT have different approach on communication style. COAP polls server for message availability. MQTT publishes messages when it is ready.

3.3 Metrics

There are several metrics are determined and linked with RQs. Their definitions are given in the previous section. In this section, further information is provided. Adding that, how these metrics are gathered is explained.

Central Processing Unit (CPU) Usage: CPU usage time and idle time are acquired from the android system file named "proc" and then CPU usage percentage is calculated.

Memory Usage: Unshared memory usage is acquired from the Android operating system.

Battery Usage: Android also provides battery status for applications. It can be acquired programmatically via the native Android libraries.

Delay: It is measured as a time elapse for single packet delivery. Due to their different nature, for each protocols delay is calculated differently. In COAP entire request/response time is qualified for delay. However, in MQTT delay is transmission from publisher to subscriber.

The whole picture of Goal-Question-Metric tree is depicted in Figure 3.

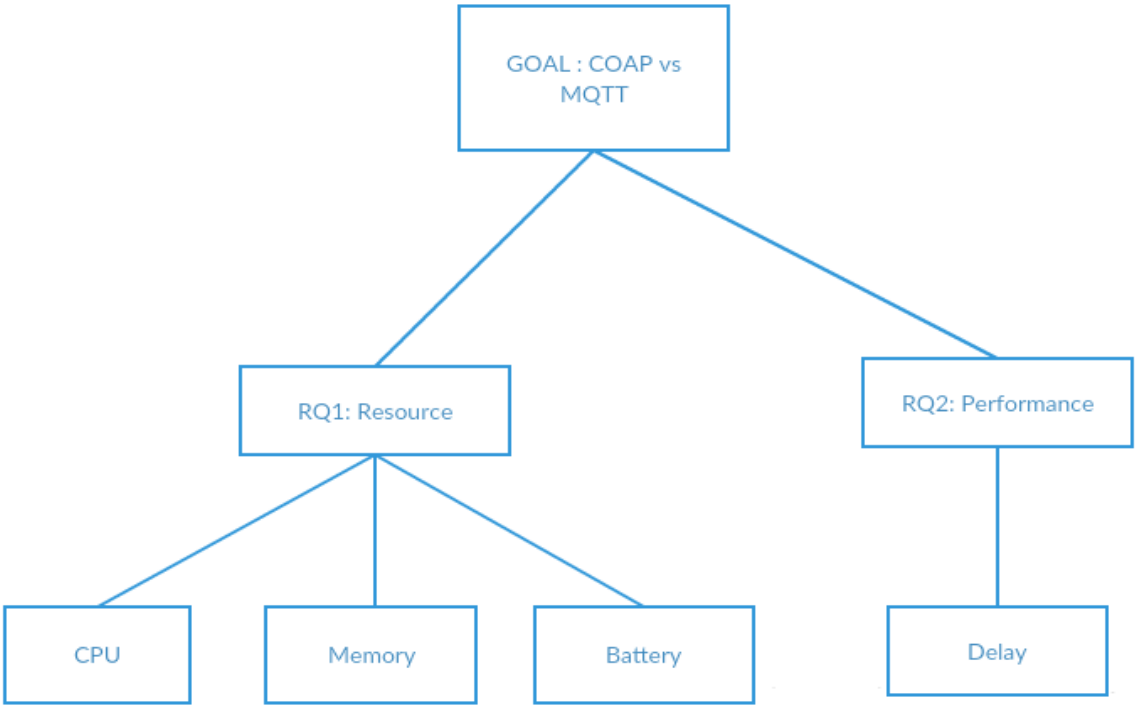

Figure 3: GQM Structure

CHAPTER 4

EXPERIMENTS/ EXPERIMENT SETUP

This chapter contains experiment related information. Experiment setup is explained in section 4.1 and measurement of tests is located section 4.2. Experiment results and protocols comparison are discussed in chapter 5.

4.1. Setup

For our experiments, we setup a closed private network and develop context as a healthcare system which is driven by HL7 messages. We compare MQTT and COAP not on network layer but on application layer. Since this thesis focus on IoT, we try to understand protocols' behavior on network nodes. In our case, the clients are nodes of network. For both protocols, we developed both server and client applications. Server applications are responsible for providing HL7 messages for the experiment. The results are collected from clients. Client applications are developed as light as possible to acquire protocols' resource usages accurately. In both applications metrics are acquired by identical software subcomponents. The only difference between Client applications is protocol implementation. In MQTT, we use M2MQTT.jar published by IBM that implements MQTT protocol. In COAP, Californium (M. Kovatsch, 2014) implementation of COAP is used. The network setup is explained below and context is detailed at "Test Case Scenarios" section.

4.1.1. Network Setup

Network setup consists of a single router with Wi-Fi capabilities. The Network is configured as closed, private network to eliminate any outside effects. There is no direct outside influence on clients and servers. Since it is closed to the world, any bandwidth configuration can be provided for test cases.

4.1.2. MQTT Setup

Setup for MQTT tests are divided into two categories:

Hardware

Hardware setup of MQTT is consists of three components (Figure 4):

- PC working as a server
- Router
- Android device working as client

A PC is configured to be the host of broker and server. The Broker is a third party software which is responsible for publish/subscribe mechanism of MQTT. Mosquitto is used for the broker in this setup. We have developed the Server

component for distribution of data. An Android device simulates the client. Android application implementing the MQTT client runs on this device. Figure 4 illustrates hardware setup for MQTT.

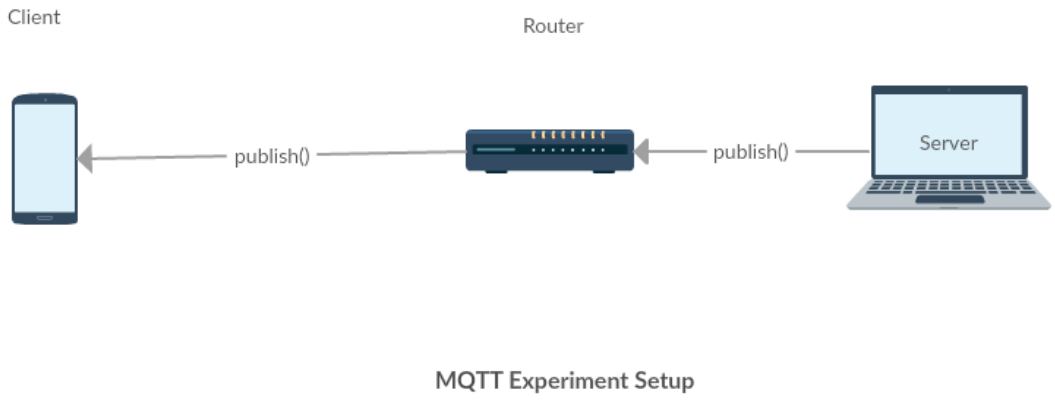

Figure 4: Hardware Setup for MQTT

Software

Two programs were developed for this setup: MqttClient and MqttServer. On the server side, we implemented an MQTT server, named MqttServer, that is responsible for publishing data. It has simple user interface and it is single process application. This server uses a broker, called Mosquito, to publish data on the network via a router. On the client side, we implemented a MqttClient application that handles published messages. In order to accomplish this task, MqttClient subscribes to the broker before publishing commences. MQTTClient also has simple user interface, and all functionality is delegated to an asynchronous task (thread) in order to prevent screen locking. In order to lighten our client applications, we only implement messaging and data gathering functionality at our client application. Our aim is to reach only resources that are used on messaging phase. We apply same constraints on COAPClient as well.

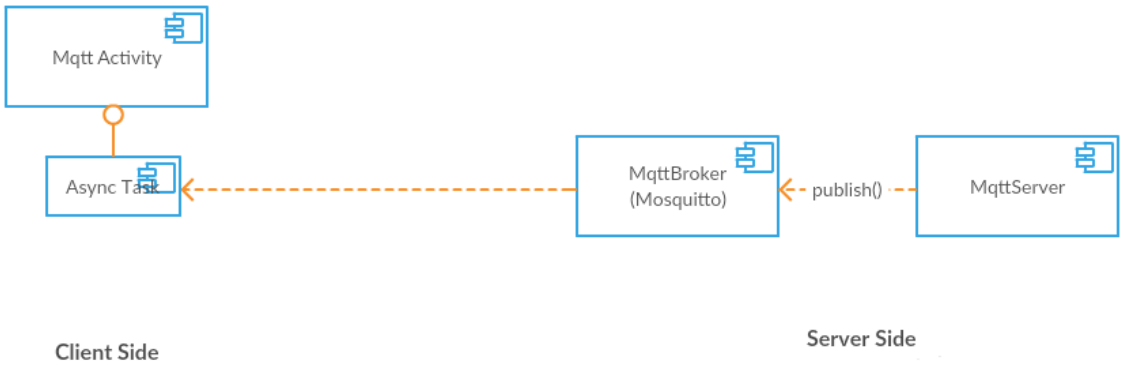

Figure 5: Software Components of MQTT

Figure 5 shows the software components used in the experiment set up. MQTTServer is the component that runs on PC. It is responsible for publishing messages. It implements M2MQTT.dll that IBM published for .NET environment. MQTTServer publishes the messages; Mosquitto (Light, 2017) brokers the messages into the network. This concludes Server Side's operations. MQTTClient application receives the messages. MQTTClient implements M2MQTT.jar also, published by IBM for Java environment[i]. That is our client application. For each arrival of message, metric data are stored. At the end of the test run metrics are dumped into text files for analysis.

4.1.3. COAP Setup

Similar to the MQTT setup, the COAP setup is classified into two parts.

Hardware

The hardware setup of COAP is the same as MQTT (Figure 6):

- PC working as a server
- Router
- Android device working as client

CoapServer operates on a PC. The Router is responsible for network transfers. Android device simulates the client. Android application of COAP runs on this device.

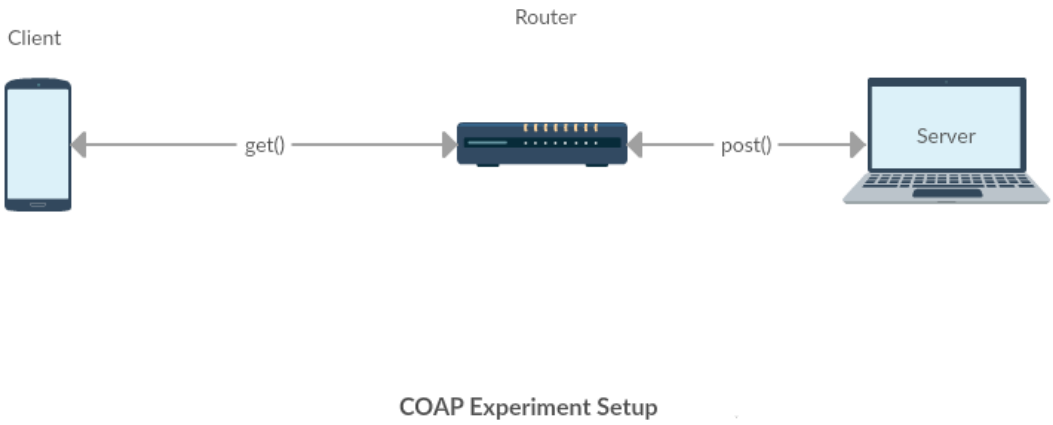

Figure 6: Hardware Setup for COAP

Software

Similar to MQTT, we have developed two programs: CoapServer and CoapClient. CoapServer posts prepared resources to CoapClients' requests. Unlike MQTT, COAP applications are driven by clients.

Figure 7 shows the software components we have implemented and used in the experiment set up. COAPServer is the component that runs on PC. It is responsible for responding to COAPClient's requests. It is a COAP.Net[ii] framework adaptation. It is a single process application with simple interface. COAP.Net is developed by ETH Zurich for .Net environment. COAPClient application requests the messages and acquires them from COAPServer's responses. Californium (M. Kovatsch, 2014) implementation is used in COAPClient. It is a SpitFireFox[iii] adaptation. It also has an asynchronous task (thread) which handles communications. For each message acquired, metric data are stored. At the end of the test run, metrics are dumped into text files as same way in MQTTCLient for analyze.

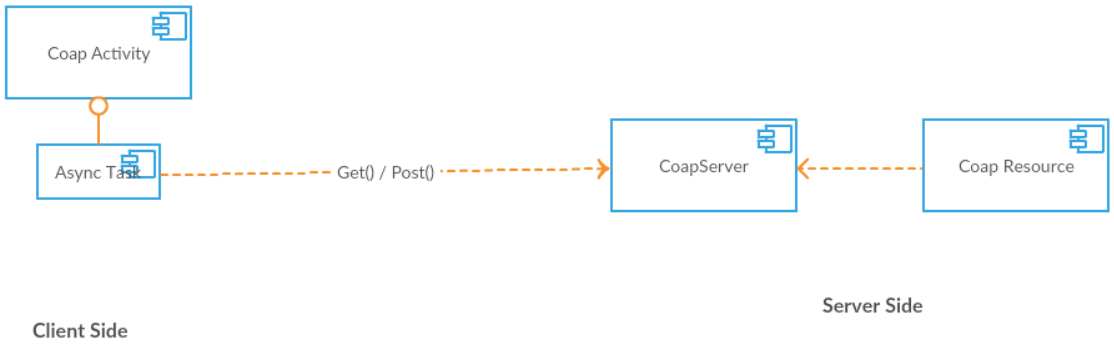

Figure 7: Software Components of COAP

4.1.4. Test Case Scenarios

We design our tests under the context of medical applications. We have planned the experiments in several scenarios aiming to simulate different environments in order to compare MQTT and COAP under possible circumstances. In E-Health scenarios, departments of medical facilities are simulated. Hospitals can roughly be divided into 3 large departments. These departments are emergency, out-patient polyclinic and inpatient service. Therefore, we developed one scenario for each of these departments. Scenarios are built on HL7 messages which are transferred via network where MQTT or COAP running. Each scenario has different ratio of HL7 messages. Our scenario message distribution is depicted in Table 3:

Table 3: Test Case Scenarios

Scenario	**ADT%**	**ORM%**	**ORU%**	**DFT%**
Emergency	60	20	10	10
Polyclinic	20	30	30	20
Inpatient	10	35	35	20

The reasoning behind these ratios is basic operational habits of hospitals. In emergency, a large number of patients are admitted and after basic treatment they are either discharged or transferred. Hence ADT ratio is high at emergencies. In out-

patient polyclinics where treatments are applied balanced message pattern can be seen. Because for nearly all patients who are admitted there is medical analysis orders and observations are applied. Obviously there will be some pharmacy-related actions. Inpatient department messages are comprised mostly order and observation messages.

We created emergency, polyclinic and inpatient test cases consisting of 300 messages and set up a message repository. During the experiments, the server applications (MQTTServer and COAPServer) get the messages from this repository and send them in the order they appear. On each test run they publish (MQTT) or response (COAP) from server, depending on which protocol is being tested. Server applications have presented one message per 100 ms out of 300 message pool for 15 minutes. When all the messages are sent, the server continues to sending the messages starting from the beginning of the repository. Message package sizes are between 386-534 Bytes for MQTT and 370-520 Bytes for COAP.

4.2 Qualitive Analysis on Reliability and Integrity

In this section, we provide a qualitative analysis on reliability and integrity of the two messaging protocols. This analysis is a qualitative one using the literature and the standards specification since our experimental setup is on a closed network which is not suitable to evaluate these aspects empirically.

Since COAP and MQTT are lightweight network messaging protocols, their primary aim is to deliver data from source to destination. We analyzed this delivery in terms of two aspects: Reliability and Integrity. Reliability refers that data will *definitely* reach from source to destination. Integrity refers that data will *correctly* reach from source to destination.

MQTT provide reliability and integrity at the transport layer due to the use of TCP. MQTT runs over TCP (Locke, 2010). TCP specification (Postel, 1981) states that TCP provides reliable communication. TCP applies sequence number and acknowledgements to make transmission reliable (Postel, 1981). Adding that, MQTT specification (Locke, 2010) states that QoS2: “Exactly Once” assures there is no loss or duplication of messages. Thangavel et al (D.Thangavel, 2014) experimentally support this statement. Although, they do not provide experimental result Tucic et al (M. Tucic, 2014) and Yi et al (D. Yi, 2016) are also supports reliability of MQTT.

COAP runs over UDP (Z. Shelby C. K., 2013) which has a reputation for unreliability unlike TCP (Postel J. , 1980). COAP handles reliability issues at application layer. A retransmission mechanism is added to provide reliability. Also, COAP message packets contain 16-bit *Message ID* section to detect duplicates (V. Karagiannis, 2015). COAP retransmit messages until it acquires an acknowledgement (Chen, 2014). By applying HTTP features, COAP manages acknowledgements and eliminate UDP’s deficiencies (Z. Shelby C. K., 2013). Thangavel et al (D.Thangavel, 2014) and Bandyopadhyay et al (S. Bandyopadhyay, 2013) support this statement.

The standards and the literature show that both protocols have reliability and integrity properties. Therefore, before running our experiments, we checked whether the protocol implementations satisfy these conditions. I.e. the protocol and mobile implementations transmit messages in a reliable fashion with preserved integrity. This sanity test on our MQTT and COAP implementation showed that the applications developed for the experiment setup satisfy the reliability and integrity conditions.

4.3. Measurement

Before proceeding measurement and the analysis section, we explain how data are gathered and evaluated. The metrics are collected during test runs. When a client receives a message a set of metric data are stored. However, these data are not meaningful yet. They are raw information acquired from Android system files. To lighten the applications, analyze phase is separated from client applications. In other words, we divide measurement phase into two groups: Collection Phase and Evaluation Phase. In collection phase, the aim was to gather raw data accurate and valid. To achieve dependable results, we ran the test scenario 10 times for both protocols.

We ran the test scenario for 10 times with the time interval of 15 minutes. After each test, Delay, CPU Allocation, Memory Usage, Battery data were gathered from the application log.

After gathering data phase, evaluation phase initiated. In this phase raw data were subject to calculation of average and median means. The reasons behind these calculations are to eliminate spikes or inaccurate results which were caused by unexpected or irrelevant outside effects.

The following sections present the experiment results.

4.3.1. MQTT

In this section the test results for MQTT are presented. When message is ready, server application publishes the messages and client application gathers results.

Delay: MQTT has around 46 milliseconds delay between server and client. The numeric results are depicted in Table 4:

Table 4: Message Delay Results for MQTT

Run Count	**MEDIAN(ms)**	**AVG(ms)**	**S.DEV(ms)**
Run 1	46	39.470	8.944
Run 2	46	39.660	8.888
Run 3	46	39.755	8.852

Run 4	46	40.305	8.512
Run 5	46	39.820	8.710
Run 6	46	39.720	8.907
Run 7	46	40.100	8.732
Run 8	46	39.320	8.721
Run 9	46	39.665	8.837
Run 10	46	40.270	8.661

Graphical representation in Figure 8 also shows stable condition for delay metric. It transfers messages within expected range of delays. Notable derivation from Figure 8 is the difference between median and average values. This shows us that MQTT fluctuates within range of 8-9 ms in terms of delay. It sometimes transmits messages below 35ms. However, it is quite stable around 46ms. Our results are very close to the literature's Wi-Fi latency results (S. Hamida, 2015) with respect to MQTT.

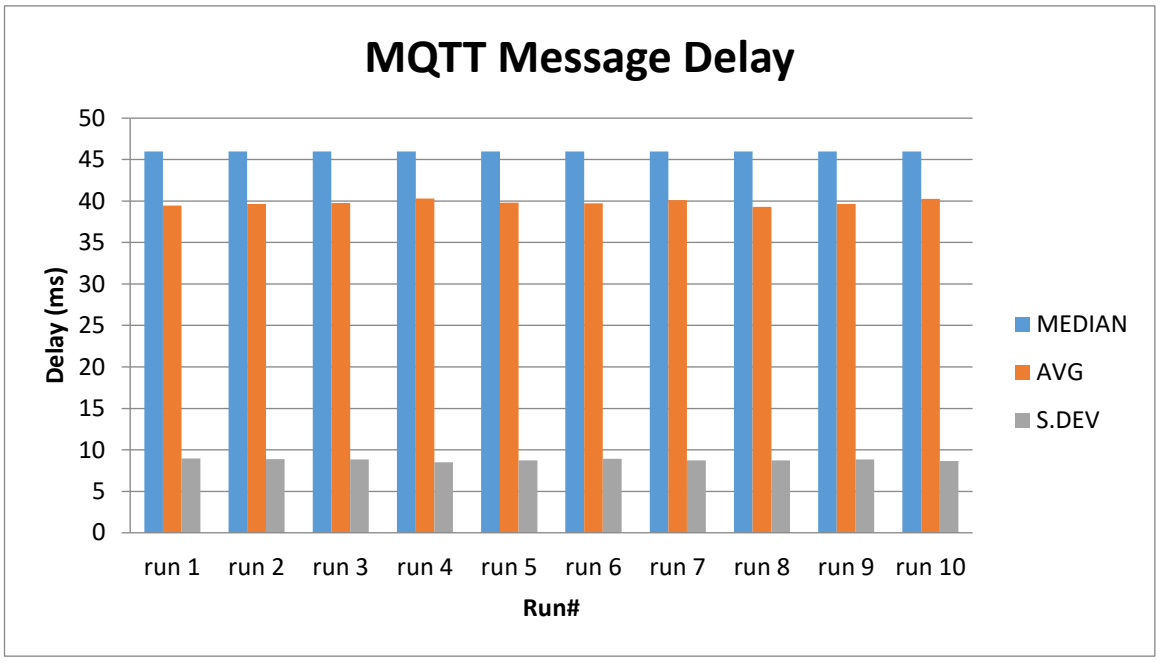

Figure 8: Message Delay Representation for MQTT

CPU Usage: The CPU resource is divided between applications running on the operating system. Amount of time that the processor allocated to the application is considered as CPU resource usage of application.

In android CPU usage can be acquired any given time from "*proc*" file. CPU usage data are collected via formula of:

$$CPU = ((cpu2 - cpu1) / ((cpu2 + idle2) - (cpu1 + idle1))/1000$$

Equation 1

Equation 1[iv] calculates application percentage of CPU usage while running. Running applications share CPU. Operating system is responsible for allocating CPU between running applications. Hence every application uses processor in turns. CPU Allocation denotes the time that processor actually processes designated tasks from application when the application takes its turn of processor usage. Cpu2 is the duration after the application used processor and cpu1 is the duration before. Idle denotes processor sits idle when application is using processor. Cpu2 and cpu1 denote processor usages before and after application's turn. Idle2 and idle1 denote how much time processor sits idly while application is running.

MQTTClient application allocates around 26% of CPU. Results are depicted in Table 5:

Table 5: CPU Allocation Percentage Results for MQTT

Run Count	**MEDIAN (%)**	**AVG (%)**	**S.DEV (%)**
Run 1	24.786	24.763	0.199
Run 2	27.815	27.805	1.115
Run 3	25.623	25.645	0.346
Run 4	25.556	25.568	0.198
Run 5	26.298	26.165	0.405
Run 6	26.360	26.328	0.269
Run 7	26.669	26.723	0.433
Run 8	27.069	27.094	0.190
Run 9	26.972	26.514	1.028
Run 10	24.741	24.722	0.207

Figure 9 shows MQTT results for CPU metrics. Horizontal axis shows the test run identifier. The vertical axis shows CPU Allocation percentage of MQTTClient. The columns show the CPU Allocation of each test runs. The average and the median of the measurements show that MQTTClient allocates 25% of CPU with little fluctuation.

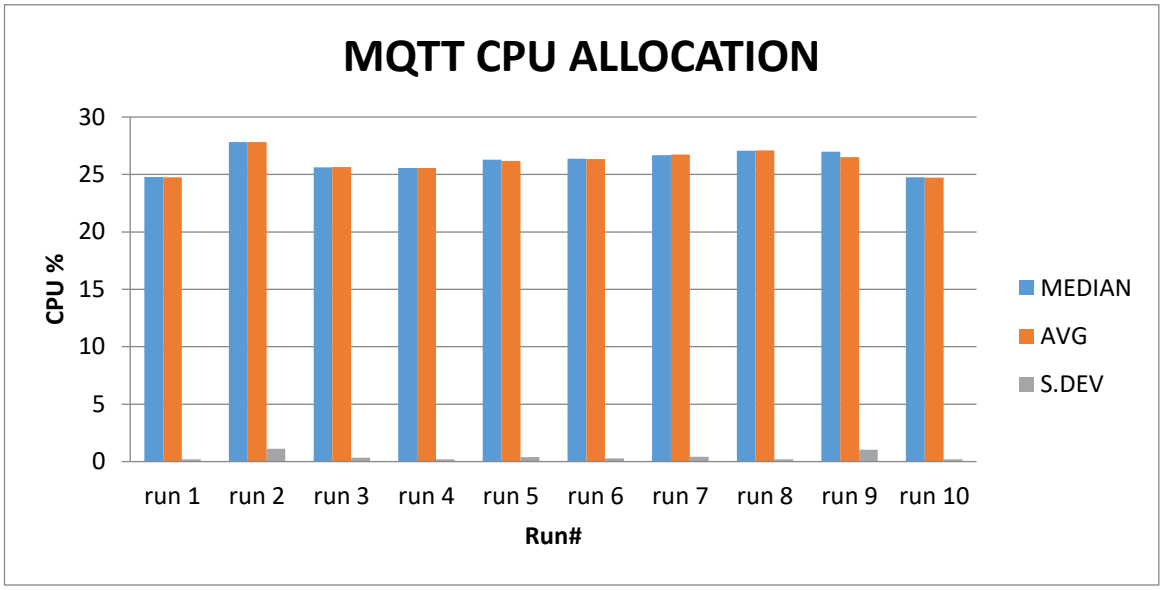

Figure 9: CPU Allocation Representation for MQTT

Memory: MQTTClient uses around 12 MB of unshared memory. Android has divided device's memory between processes. Processes might be sharing the same segment of memory. However, to determine accurate memory usage in this thesis only unshared memory is gathered. Results are depicted in Table 6:

Table 6: Memory Usage Results for MQTT

Run Count	**MEDIAN(MB)**	**AVG(MB)**	**S.DEV(MB)**
Run 1	11.718	11.625	0.341
Run 2	12.556	12.499	0.230
Run 3	12.184	12.152	0.148
Run 4	12.176	12.144	0.156
Run 5	12.532	12.384	0.227
Run 6	12.552	12.467	0.195
Run 7	12.512	12.458	0.183
Run 8	12.508	12.474	0.150
Run 9	12.496	12.242	0.440
Run 10	11.658	11.581	0.415

The graphic in Figure 10 shows memory usage of MQTTClient while it is running. The vertical axis represents memory usage in MB.

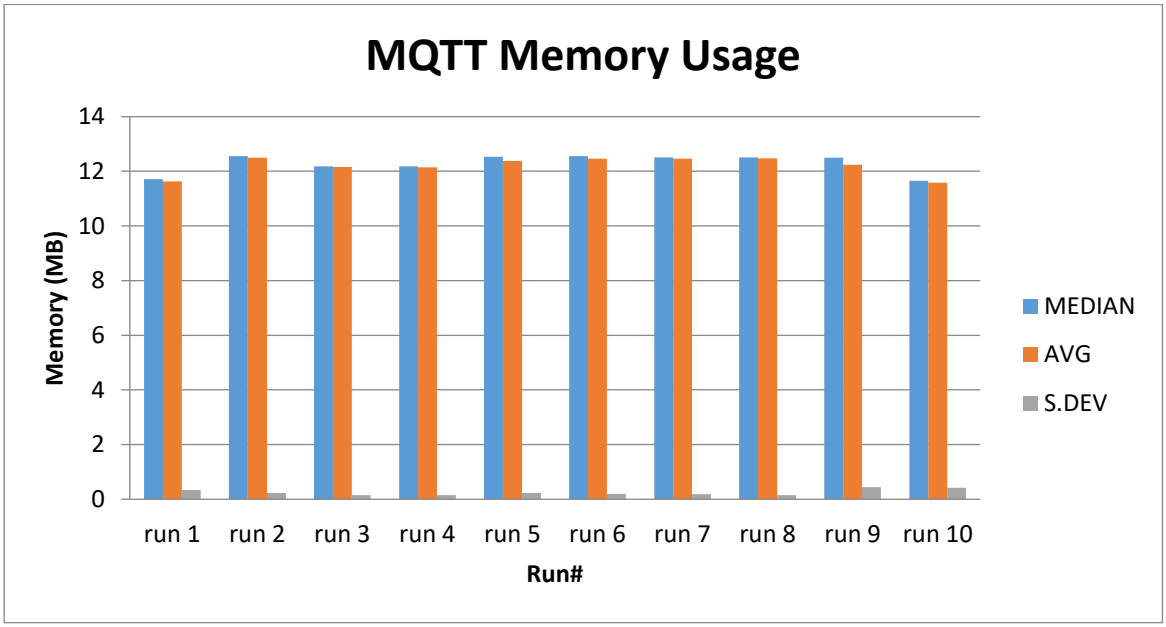

Figure 10: Memory Usage Representation for MQTT

As the Figure 10 depicts MQTTClient uses 12MB of memory. Similar to CPU Allocation, memory usage of MQTTClient is stable.

Battery Usage: To determine accurate power consumption the battery was fully charged before running tests. Approximately, MQTTClient consumed %10 of battery during the test sessions:

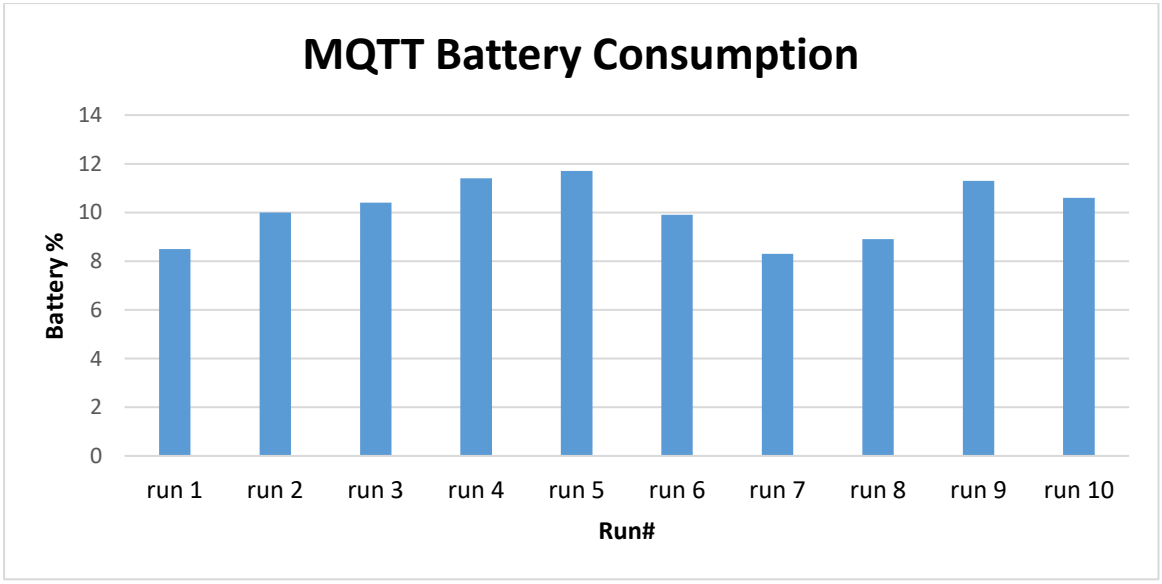

Figure 11: Battery Consumption Results for MQTT

Since battery is bottleneck for mobile devices energy consumption is important metric for our study. Figure 11 shows the battery consumption results for MQTTClient results. Extensive explanation and comparison is located in Chapter 5.

4.3.2 COAP

In this section the test results for COAP are presented. Since COAP is based on request/ response pattern we conduct our experiments by polling for messages with different intervals. We choose 50ms, 100ms, 150ms, and 200ms for polling. After each poll we gather metric data. For delay data we collect latency between message generation and client's message acquisition. CPU and memory data are collected from Android *proc* file system. We get battery data from Android inbuilt PowerManager[v] library. Experiment results are depicted in this section. Further analysis and discussions are placed in Chapter 5.

50ms Polling Results:

In this run COAPClient polls server per 50ms.

Delay: COAP has around 7 milliseconds delay at 50ms polling. Its numeric results are shown at Table 7:

Table 7: Message Delay Results for COAP(50ms)

Run Count	MEDIAN(ms)	AVG(ms)	S.DEV(ms)
Run 1	6,5	9,725	20,758
Run 2	6	11,21	28,620
Run 3	7	9,235	24,820
Run 4	6	9,935	24,802
Run 5	6	8,970	21,0245
Run 6	7	10,935	30,703
Run 7	7	12,050	27,540
Run 8	6	11,840	36,380
Run 9	6	10,685	32,180
Run 10	7	7,780	6,011

Graphical representation in Figure 12 shows that delay results vary but COAP performs better than MQTT with respect to message latency in 50ms polling case. Recall that the delay is measured as the time difference between a message is generated and its client acquisition.

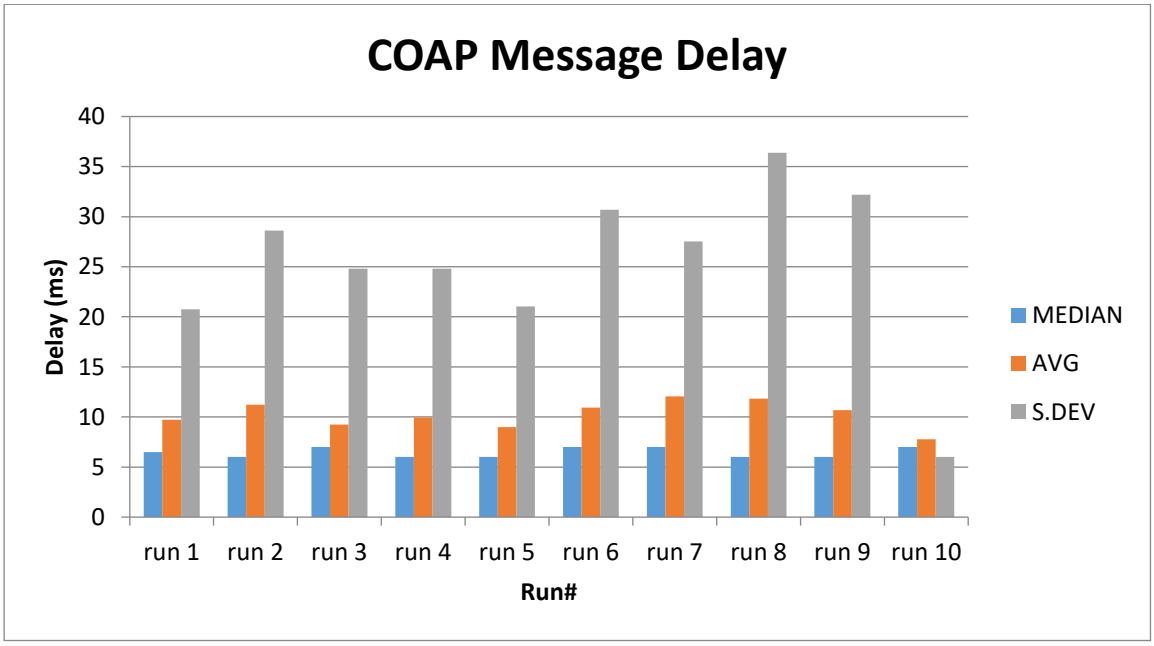

Figure 12: Message Delay Representation for COAP(50ms)

CPU Usage: In android CPU usage can be acquired any given time from "*proc*" file. CPU usage data are collected via equation 1. COAP Client application allocates around %53 of CPU. Results are depicted in Table 8.

Table 8: CPU Allocation Percentage Results for COAP(50ms)

Run Count	**MEDIAN (%)**	**AVG (%)**	**S.DEV (%)**
Run 1	54,916	63,932	24,367
Run 2	51,962	62,129	25,185
Run 3	53,290	61,360	22,256
Run 4	52,593	59,914	23,387
Run 5	52,203	62,959	25,573
Run 6	53,916	61,451	24,073
Run 7	52,311	59,863	22,831
Run 8	50,641	58,919	23,521
Run 9	51,041	55,982	20,869
Run 10	53,277	62,552	24,943

Figure 13 shows CPU Allocation of COAPClient for 50ms polling:

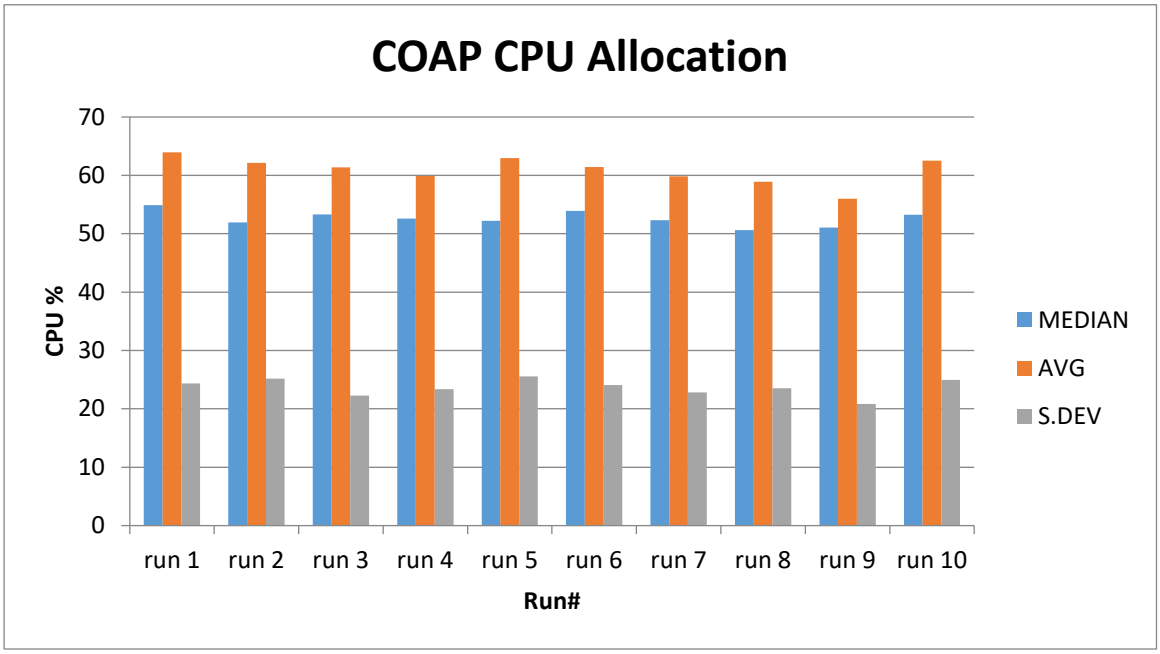

Figure 13: CPU Allocation Representation for COAP(50ms)

Memory: COAP Client uses around 37 MB of unshared memory. Android has divided device's memory between processes. Processes might be sharing same segment of memory. However, to determine accurate memory usage in this thesis only unshared memory is gathered. Table 9 shows results.

Table 9: Memory Usage Results for COAP(50ms)

Run Count	**MEDIAN(MB)**	**AVG(MB)**	**S.DEV(MB)**
Run 1	37,180	45,111	19,252
Run 2	36,902	41,827	16,448
Run 3	36,956	41,804	15,925
Run 4	37,080	42,700	17,334
Run 5	36,974	39,441	15,159
Run 6	36,992	39,988	16,691
Run 7	37,080	41,539	15,292
Run 8	36,930	40,964	14,381
Run 9	36,524	38,017	11,258

Run 10	36,712	41,759	18,784

The graphic in Figure 14 shows memory usage of COAPClient for 50ms polling:

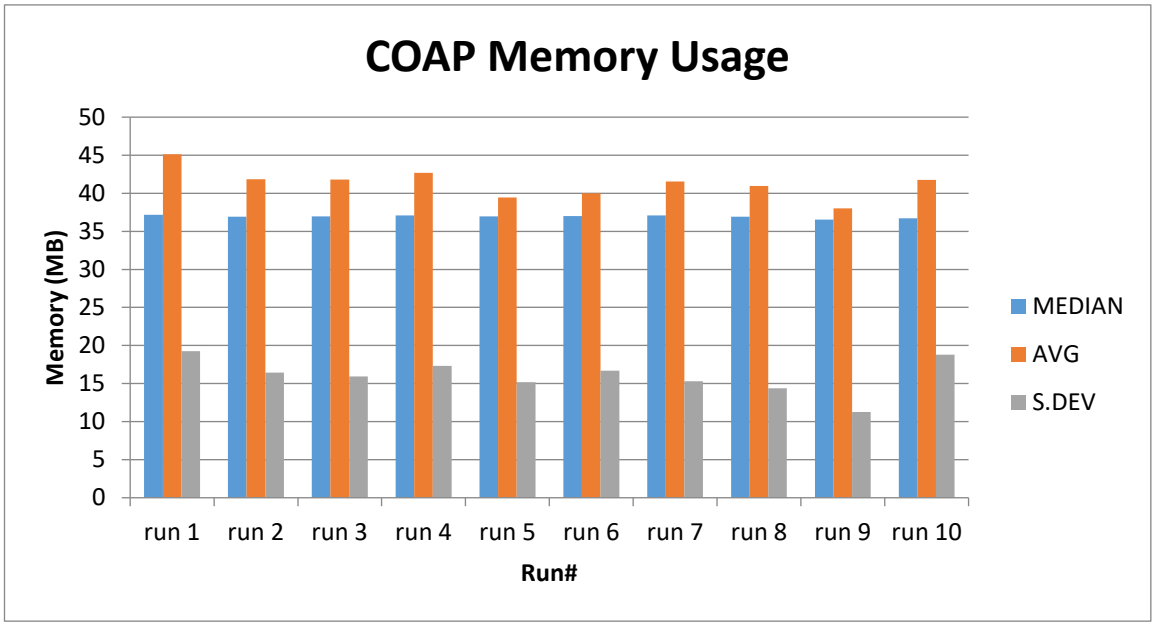

Figure 14: Memory Usage Representation for COAP(50ms)

Battery: To determine accurate power consumption the battery is fully charged before running tests.

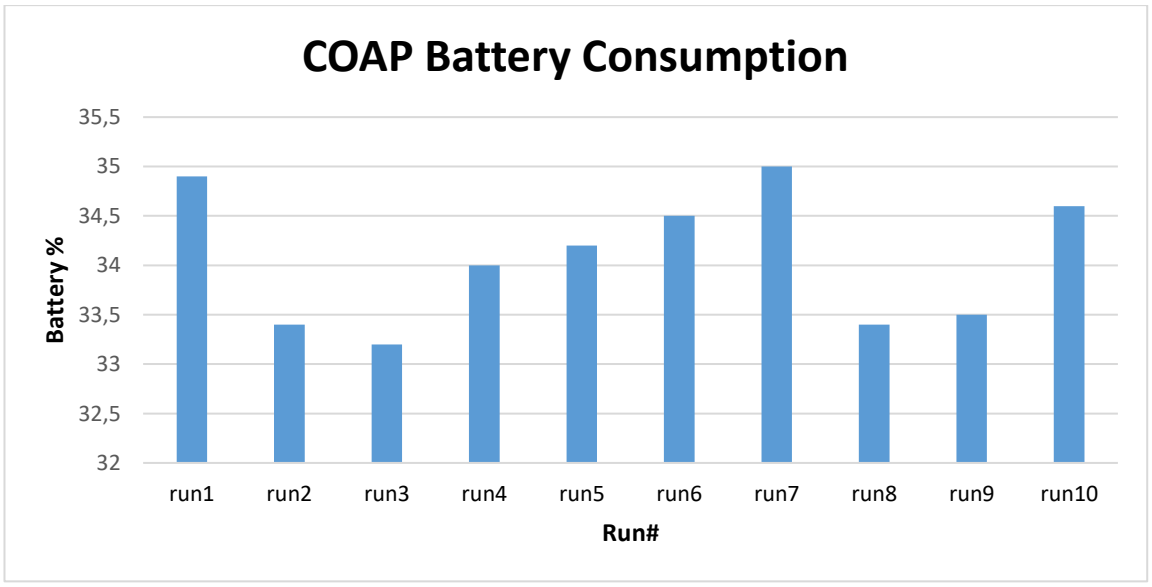

Figure 15: Battery Consumption Results for COAP(50ms)

As the Figure 15 shows COAPClient has serious energy consumption. High CPU Allocation and memory usage affect battery consumption. Also, retransmission is

costly operation which might increase the consumption. Besides, in 50ms polling case the client tries two times to get the message, since a message is generated per 100ms. This situation also affects battery. Extensive explanation and comparison is located in chapter 5.

100ms Polling Results:

In this run COAPClient polls server per 100ms.

Delay: COAP has around 7 milliseconds delay at 100ms polling. Its numeric results are depicted in Table 10:

Table 10: Message Delay Results for COAP(100ms)

Run Count	**MEDIAN(ms)**	**AVG(ms)**	**S.DEV(ms)**
Run 1	7	8,735	5,610
Run 2	7	8,345	6,436
Run 3	7	9,275	7,542
Run 4	6	8,290	5,017
Run 5	7	9,095	6,457
Run 6	7	8,715	7,391
Run 7	7	8,705	6,705
Run 8	6	8,490	5,589
Run 9	7	8,445	4,839
Run 10	6	7,725	5,611

Graphical representation in Figure 16 shows that there is a big variation on delay values on COAP, but COAP is still faster than MQTT. This fluctuation is probably caused by retransmission cases. Also, COAP's latency results are supported by Hamida et al (S. Hamida, 2015).

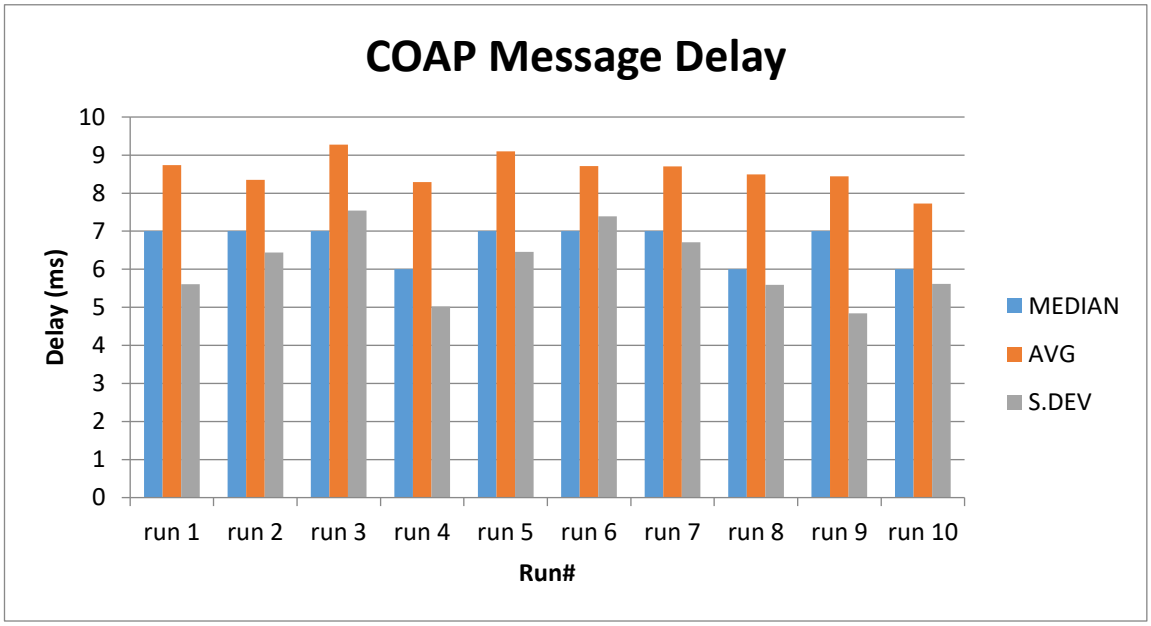

Figure 16: Message Delay Representation for COAP(100ms)

CPU Usage: COAP Client application allocates around %50 of CPU. Results are depicted in Table 11. This table shows the median, mean and standard deviation of CPU allocation percentage for each test run.

Table 11: CPU Allocation Percentage Results for COAP(100ms)

Run Count	**MEDIAN (%)**	**AVG (%)**	**S.DEV (%)**
Run 1	48,381	49,169	5,901
Run 2	49,159	49,241	5,738
Run 3	48,734	49,048	5,731
Run 4	48,739	48,712	5,468
Run 5	48,655	49,058	5,496
Run 6	48,255	48,888	5,269
Run 7	49,064	49,318	5,550
Run 8	49,927	49,690	5,294
Run 9	47,017	48,742	5,474
Run 10	49,468	49,143	6,179

Table 12: CPU Allocation Results for COAP

Figure 17 shows CPU Allocation of COAPClient while it is running:

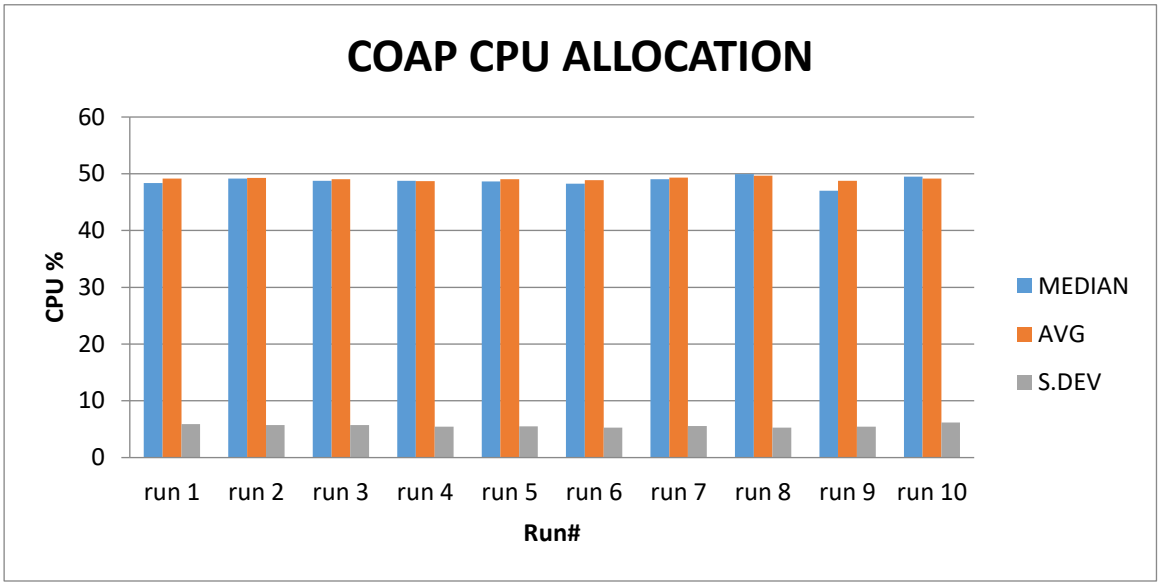

Figure 17: CPU Allocation Representation for COAP(100ms)

In Figure 17, we notice that deviation between results is clear. Together with volatile nature of delay, we concluded that in any case of retransmission also affects CPU Allocation metric.

Memory: COAP Client uses around 20 MB of unshared memory. Results are depicted in Table 13.

Table 13: Memory Usage Results for COAP(100ms)

Run Count	**MEDIAN(MB)**	**AVG(MB)**	**S.DEV(MB)**
Run 1	20.492	20.637	1.835
Run 2	20.392	20.540	1.784
Run 3	20.484	20.471	1.745
Run 4	20.382	20.433	1.633
Run 5	20.536	20.544	1.717
Run 6	20.396	20.487	1.582
Run 7	20.396	20.594	1.650
Run 8	20.492	20.680	1.516
Run 9	20.376	20.403	1.614
Run 10	20.352	20.481	1.953

The graphic in Figure 18 shows memory usage of COAPClient while it is running:

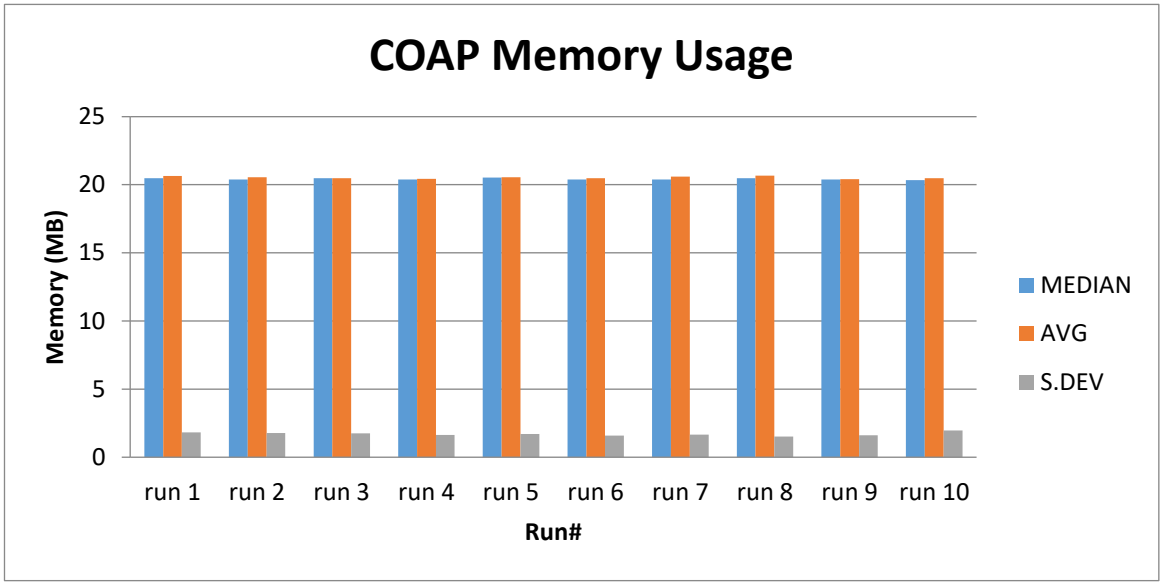

Figure 18: Memory Usage Representation for COAP(100ms)

COAPClient's memory usage is stable around 20MB with the deviation of 1.5MB at 100ms polling.

Battery Usage: Approximately, COAPClient consumes %30 of energy during the tests sessions. Figure 19 shows results:

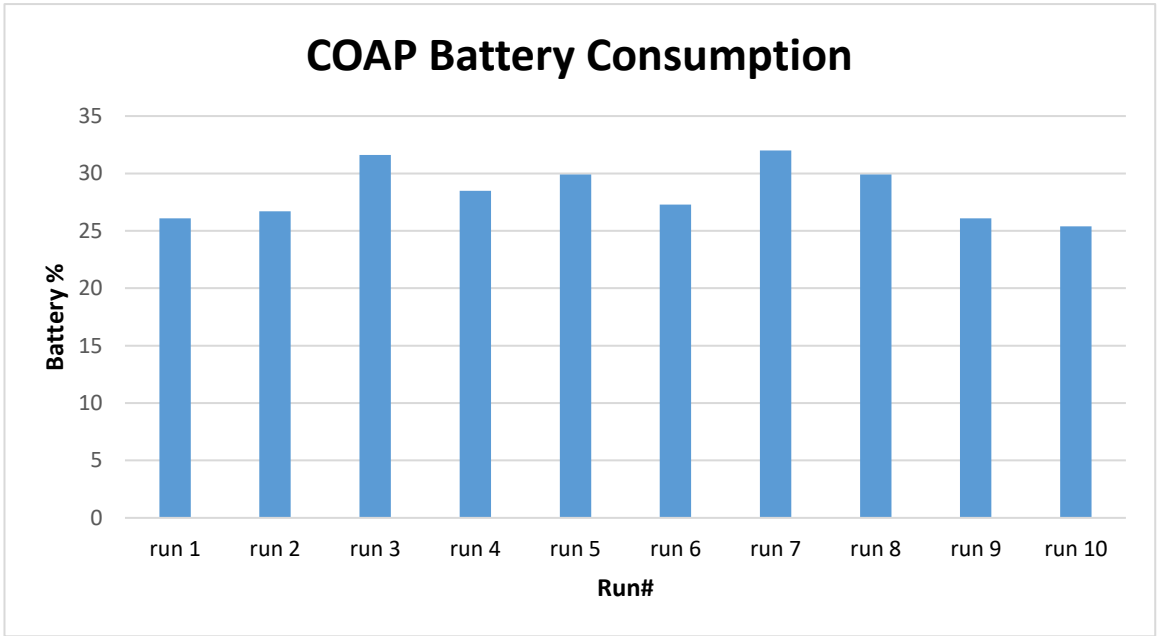

Figure 19: Battery Consumption Results for COAP(100ms)

150ms Polling Results:

In this run COAPClient polls server per 150ms.

Delay: COAP has around 56 milliseconds delay at 150ms polling. Its numeric results are shown at Table 14:

Table 14: Message Delay Results for COAP(150ms)

Run Count	**MEDIAN(ms)**	**AVG(ms)**	**S.DEV(ms)**
Run 1	56	64,525	67,830
Run 2	56	62,635	156,366
Run 3	56	86,315	215,889
Run 4	57	76,615	196,299
Run 5	56	59,660	38,119
Run 6	56,5	76,220	206,375
Run 7	57	68,475	72,152
Run 8	57	67,670	57,361
Run 9	56	59,29	43,515
Run 10	56,5	83,25	229,531

Visual depiction of latency is presented at Figure 20. Recall that in this setup, the client is polling in 150 ms intervals while messages are prepared in the server in 100ms intervals. Therefore, messages are not always ready to consume at each polling. This situation is causing the standard deviation to be high.

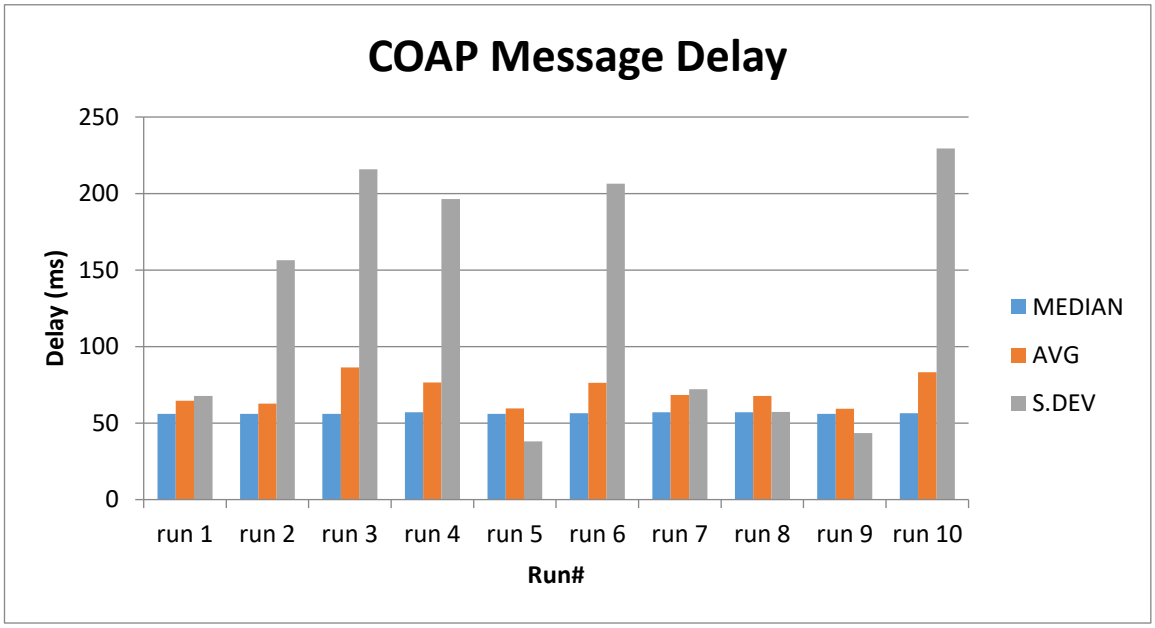

Figure 20: Message Delay Representation for COAP(150ms)

CPU Usage: COAP Client application allocates around %22 of CPU at 150ms polling. Results are depicted in Table 15.

Table 15: CPU Allocation Percentage Results for COAP(150ms)

Run Count	MEDIAN (%)	AVG (%)	S.DEV (%)
Run 1	22,425	22,476	1,1702
Run 2	22,27	22,468	1,201
Run 3	22,176	22,282	1,119
Run 4	22,414	22,556	1,143
Run 5	22,309	22,416	1,193
Run 6	22,370	22,388	1,077
Run 7	22,380	22,423	1,157
Run 8	22,397	22,427	1,185
Run 9	22,469	22,445	1,190
Run 10	22,47715	22,478	1,127

Figure 21 shows visual representation of Table 15.

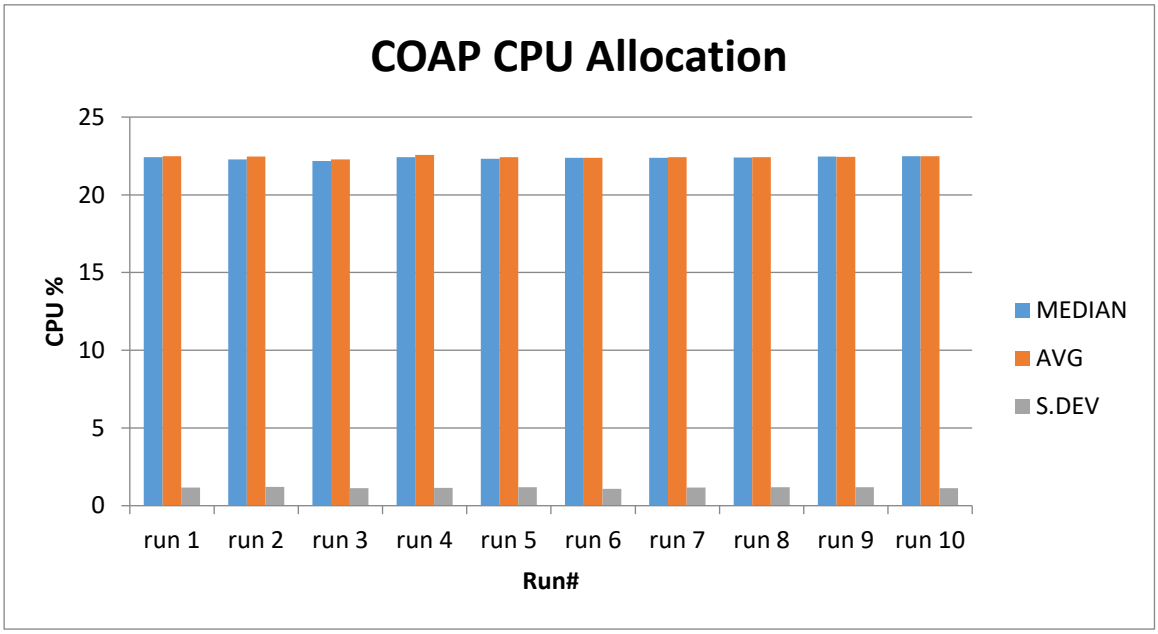

Figure 21: CPU Allocation Representation for COAP(150ms)

CPU and polling intervals are in negative correlation as expected. Smaller intervals result in higher CPU allocation.

Memory: COAP Client uses around 23 MB of unshared memory at 150ms polling. Table 16 lists experiment results.

Table 16: Memory Usage Results for COAP(150ms)

Run Count	MEDIAN(MB)	AVG(MB)	S.DEV(MB)
Run 1	23,670	26,541	8,469
Run 2	23,672	26,507	8,0480
Run 3	23,656	26,418	8,158
Run 4	23,676	26,726	8,278
Run 5	23,672	26,830	8,051
Run 6	23,750	26,137	7,172
Run 7	23,676	26,976	8,209
Run 8	23,674	26,187	7,944
Run 9	23,658	26,546	8,190
Run 10	23,664	26,281	7,751

Figure 22 shows COAPClient's memory usage at 150ms polling.

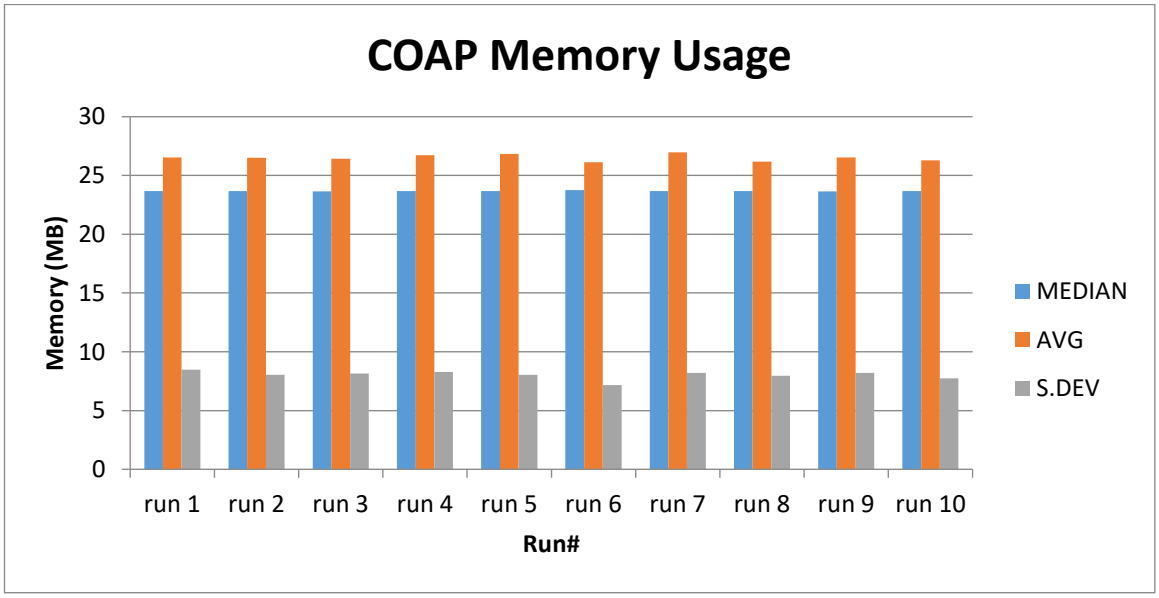

Figure 22: Memory Usage Representation for COAP(150ms)

Battery: As CPU and memory resource usages decrease, power consumption of COAPClient reduces accordingly. Numeric results are shown at Figure 23.

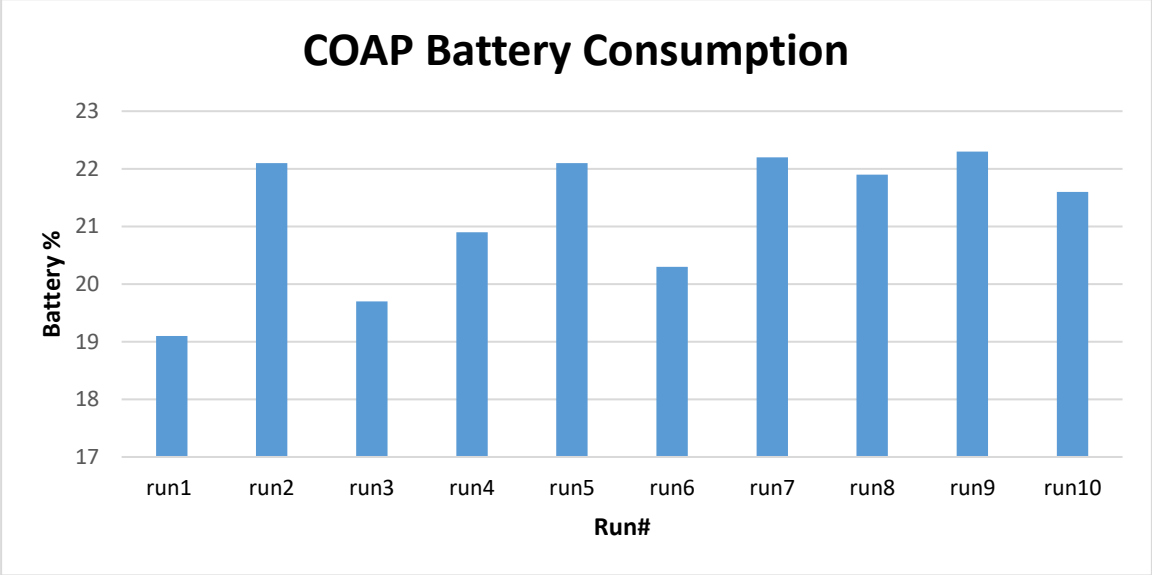

Figure 23: Battery Consumption Results for COAP(150ms)

200ms Polling Results:

In this run COAPClient polls server per 200ms.

Delay: COAP has around 60 milliseconds delay at 200ms polling. Its numeric results are shown at Table 17:

Table 17: Message Delay Results for COAP(200ms)

Run Count	**MEDIAN(ms)**	**AVG(ms)**	**S.DEV(ms)**
Run 1	65,5	64,490	66,882
Run 2	59	59,720	50,704
Run 3	59	59,865	49,881
Run 4	58	57,645	48,683
Run 5	64	66,300	63,005
Run 6	60	60,415	49,756
Run 7	67	67,230	76,006
Run 8	57	65,235	69,746

Run 9	66	65,865	63,677
Run 10	58,5	57,940	49,837

While polling interval increases COAP's latency advantage diminishes. Figure 24 depicts this result.

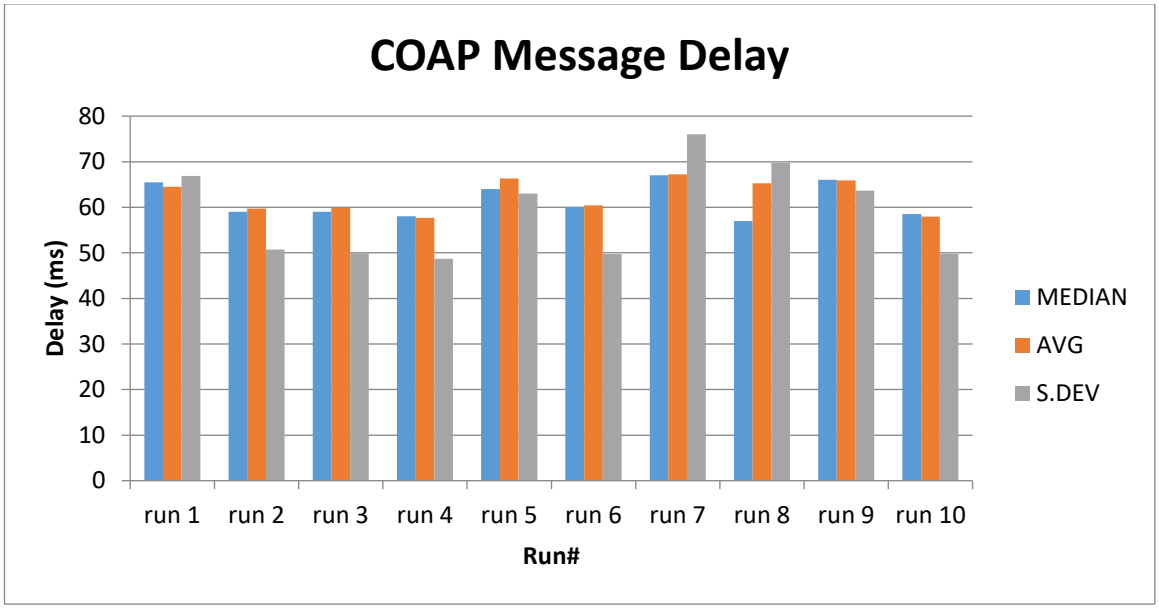

Figure 24: Message Delay Representation for COAP(200ms)

CPU usage: COAP Client application allocates around %17 of CPU. Results are depicted in Table 18.

Table 18: CPU Allocation Percentage Results for COAP(200ms)

Run Count	**MEDIAN (%)**	**AVG (%)**	**S.DEV (%)**
Run 1	17,616	17,872	1,263
Run 2	17,540	17,831	1,240
Run 3	17,780	17,930	1,195
Run 4	17,790	17,938	1,235
Run 5	17,806	17,916	1,225
Run 6	17,877	17,946	1,173
Run 7	17,761	17,931	1,246

Run 8	17,785	17,898	1,170
Run 9	17,701	17,931	1,203
Run 10	17,810	17,893	1,172

At 200ms polling, COAPClient actually allocate lower CPU than MQTTClient. COAPClients CPU allocation is shown at Figure 25.

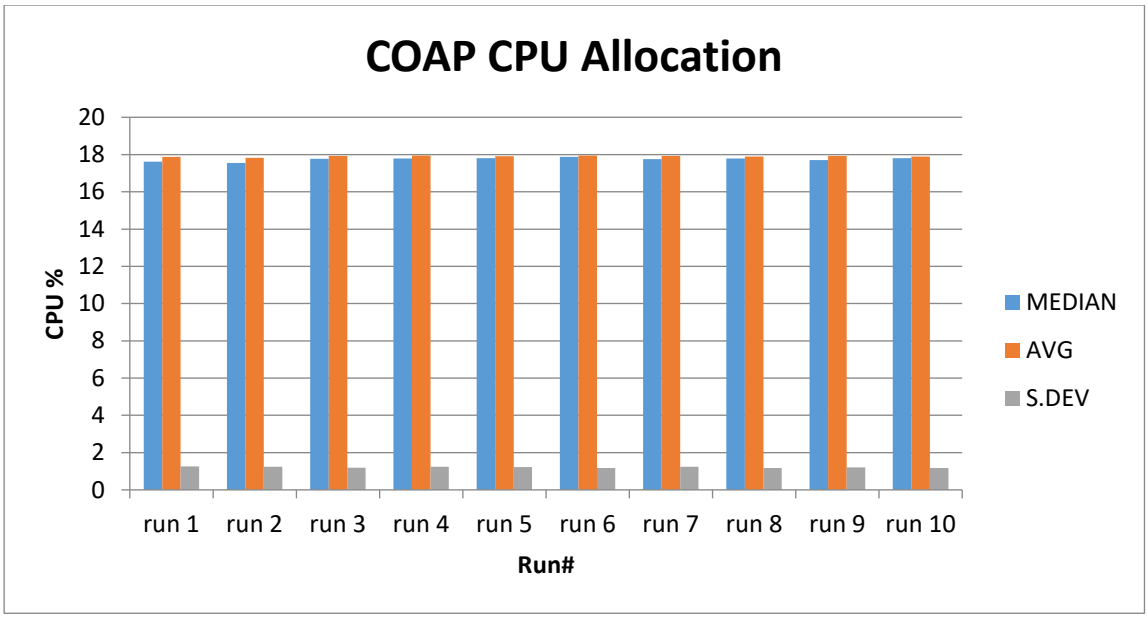

Figure 25: CPU Allocation Representation for COAP(200ms)

Memory: COAP Client uses around 24 MB of unshared memory. Experiment results are listed in Table19.

Table 19: Memory Usage Results for COAP(200ms)

Run Count	**MEDIAN(MB)**	**AVG(MB)**	**S.DEV(MB)**
Run 1	24,156	27,154	8,798
Run 2	25,016	28,192	9,406
Run 3	23,524	27,049	9,469
Run 4	25,946	27,409	8,466
Run 5	25,478	28,065	9,438
Run 6	24,268	26,642	8,731

Run 7	24,826	28,217	9,401
Run 8	25,052	28,020	9,222
Run 9	25,486	28,660	9,492
Run 10	25,532	28,382	9,229

Memory usage experiment results are placed in Figure 26.

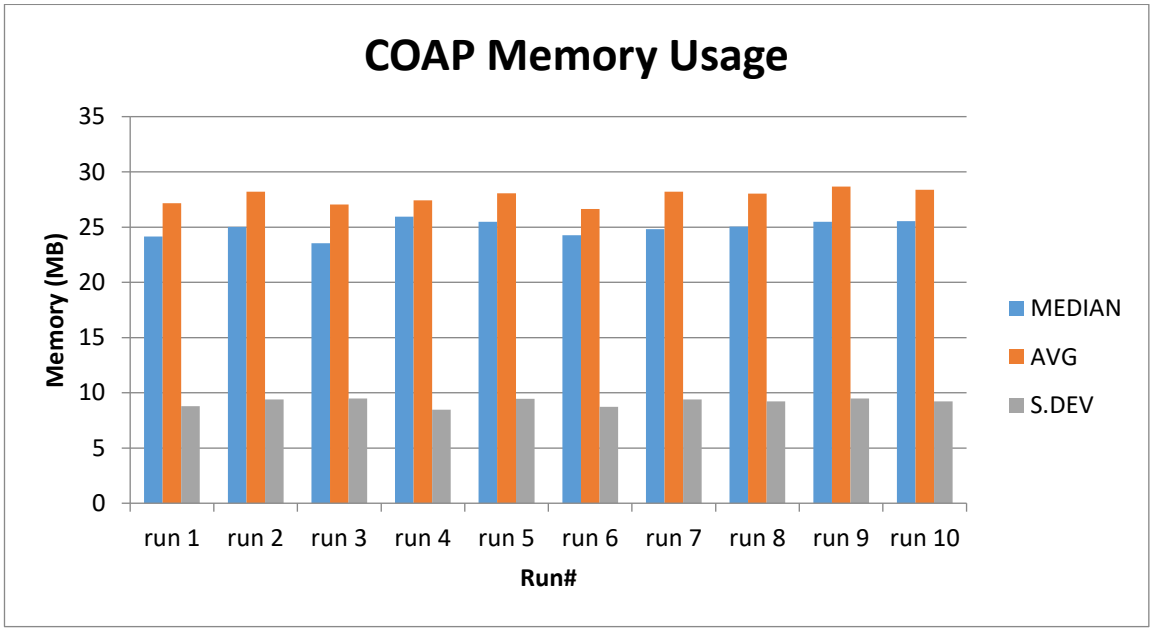

Figure 26: Memory Usage Representation for COAP(200ms)

Battery: As resource usages are drawn lower, battery consumption of COAPClient scale down to around 11 percent. Results are shown at Figure 27.

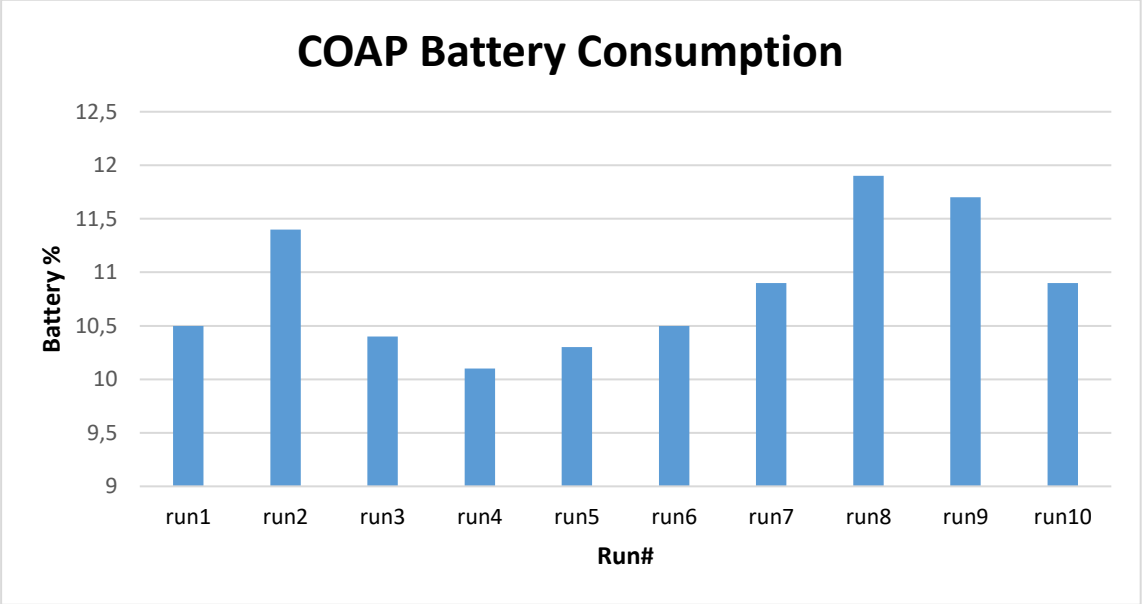

Figure 27: Battery Consumption Results for COAP(200ms)

Polling at different rates directly affects COAP experiment results. Hence, to provide a better understanding we prepare several figures. These figures compare polling interval's impacts on our metrics.

Figure 28 shows latency results under different polling intervals. Latency increases as polling interval grows. Since messages are generated per 100ms, the latency is greater in polling at 150 and 200 ms intervals. The larger intervals cause longer latencies.

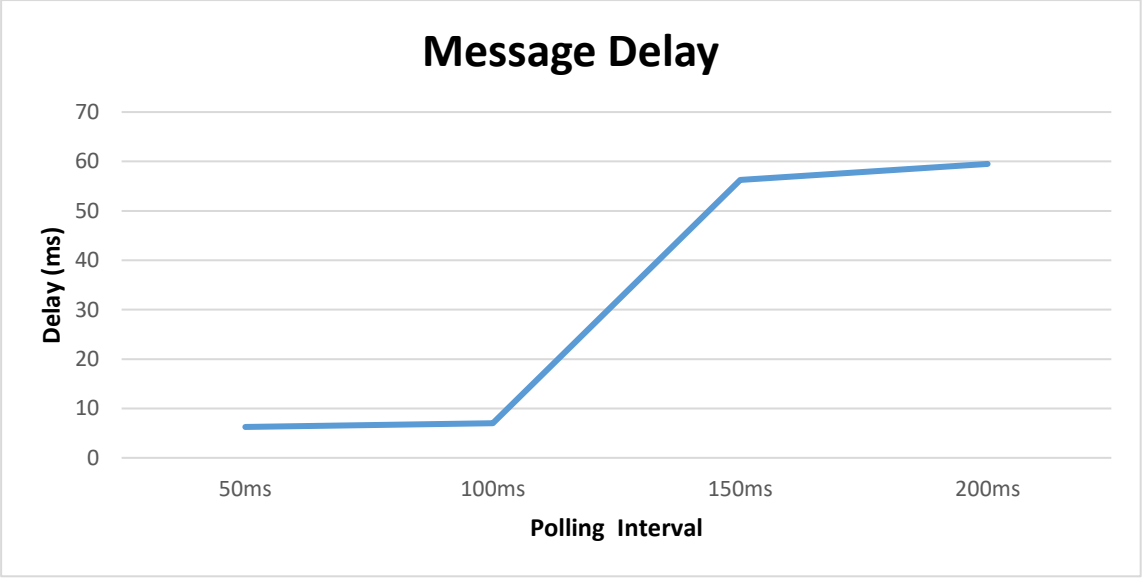

Figure 28: COAP Delay Comparison by Polling

CPU allocation has negative correlation with polling intervals. As polling interval grows, the CPU allocation decreases. Figure 29 depicts experiment results.

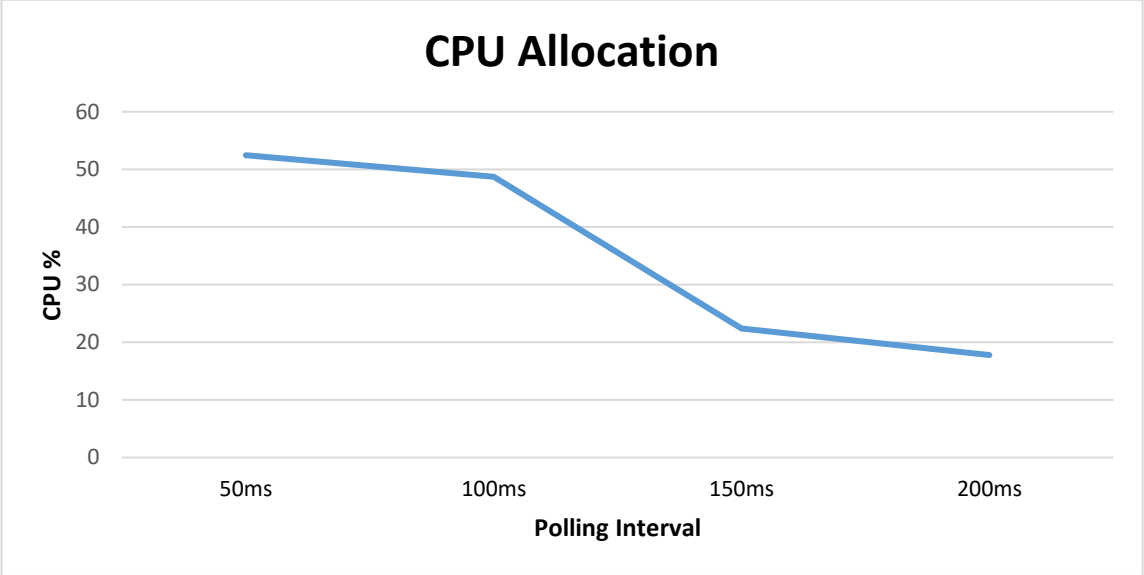

Figure 29 : COAP CPU Allocation Comparison by Polling

Polling intervals has no direct effect on memory usage. Our findings are presented at Figure 30.

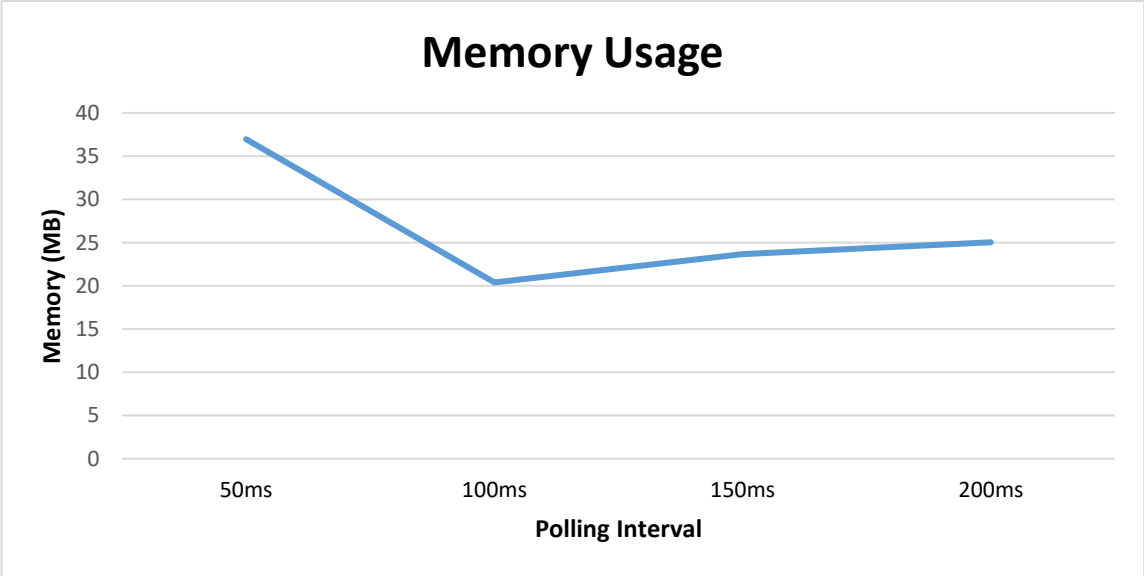

Figure 30: COAP Memory Usage Comparison by Polling

Power consumption has positive correlation with CPU usage. So, results of these two metrics are similar. As CPU usage grows, battery drainage also increases. Figure 31 shows our results.

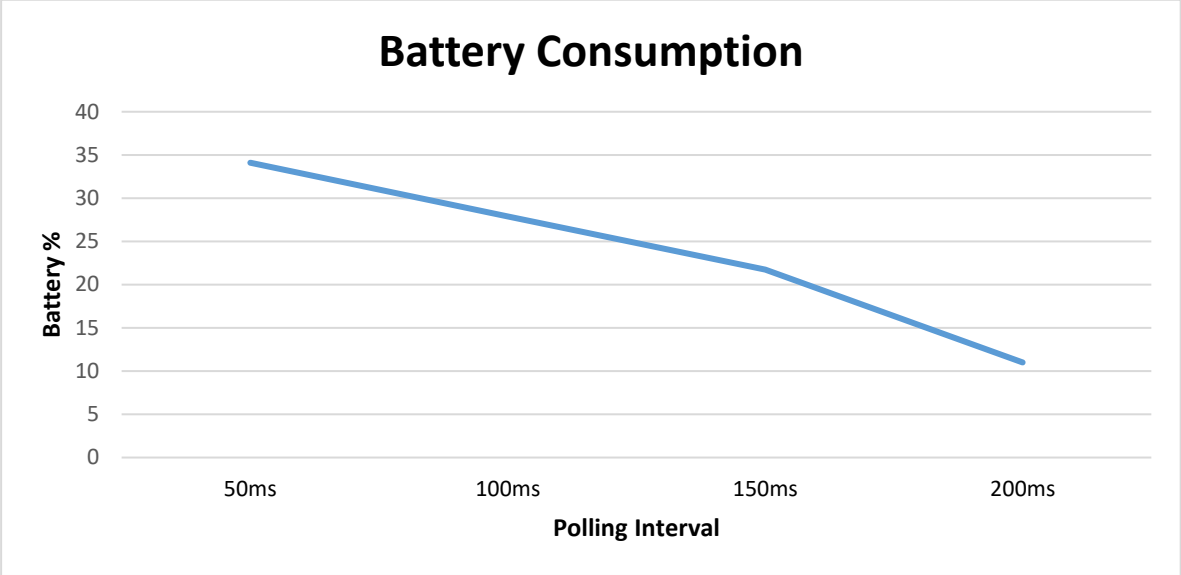

Figure 31: COAP Battery Consumption Comparison by Polling

Figure 32 provides combined experiment results on COAP.

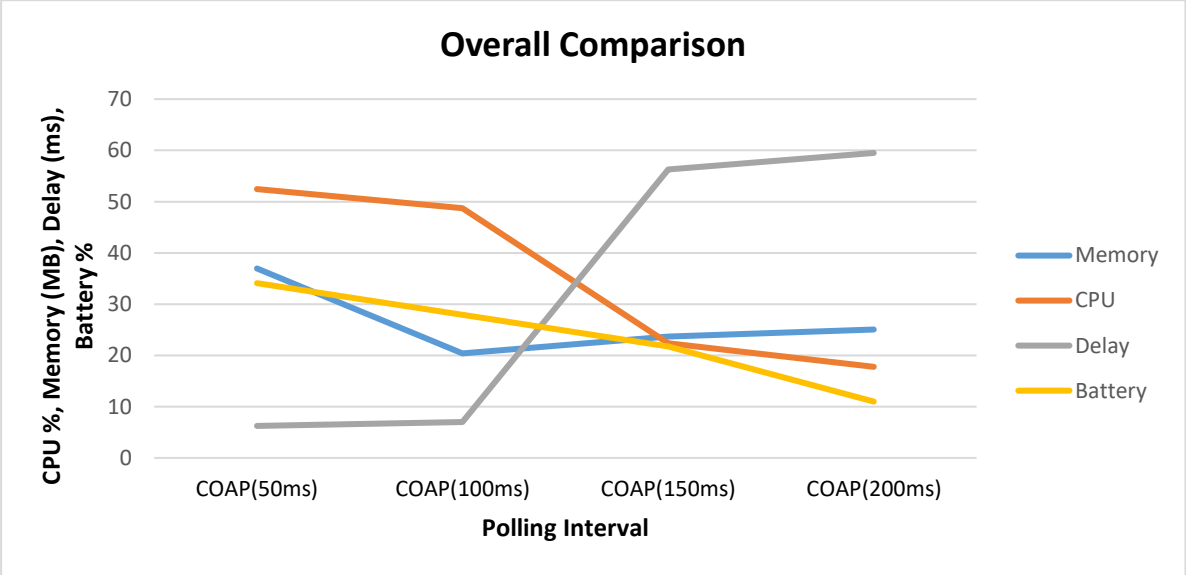

Figure 32: COAP Overall Comparison by Polling

CHAPTER 5

RESULTS AND DISCUSSION

This chapter presents our discussions on the results of the experiments presented in the previous chapter. In our experiments, the numerical results are dependent to hardware and software specifications. In order to overcome this threat, the exact same hardware and software infrastructure is used for testbed. To accomplish exact settings for hardware is easy, but it is hard to achieve for software. To provide accurate comparison environment, in software components are designed as layers and the only difference between applications (MQTTServer, MQTTClient, COAPServer and COAPClient) is protocol implementations. We also should note that, our metrics are collected from clients (nodes); since our primary focus is protocols effects on clients.

In the experiments, the servers have produced a message for each 100ms. MQTT has publish/subscribe pattern, so MQTTServer publishes message as soon as message ready. At MQTTClient we collect the metric data. However, COAP implements request/response pattern. COAPClient needs to poll server for messages. To see the effect of polling interval, we have collected the data for 50ms, 100ms, 150ms and 200ms polling intervals. Analyses of the results are listed below.

Delay: As seen in Figure 33, MQTT has higher delay than COAP, unless polling interval is bigger or equal to message generating interval. At 50ms and 100ms polling COAP fares better than MQTT; however, at 150ms and 200ms polling experiments MQTT has lower latency. At the same message acquisition timing, because of its UDP implementation and smaller packet overhead (L.Dürkop, 2015) COAP is faster than MQTT. Another interesting observation is that MQTT performs in more stable fashion. COAP has some spikes through the test sessions. Those spikes are values belong to retransmitted packets. As the COAP specification (Z. Shelby C. K., 2013) states, in the case of a packet loss the protocol triggers its retransmission mechanism. The spikes in the experiments are caused by retransmission of packets. Since UDP does not have any reliability feature, HTTP enables COAP to provide this feature. Although, COAP is significantly faster than MQTT, retransmission is extremely expensive. Retransmission nearly doubles message delay. Our testbed is private network; hence number of retransmission cases is not high. Our results are backed by Hamida et al (S. Hamida, 2015). Comparison on delay aspect between COAP with different polling intervals and MQTT is shown Figure 33.

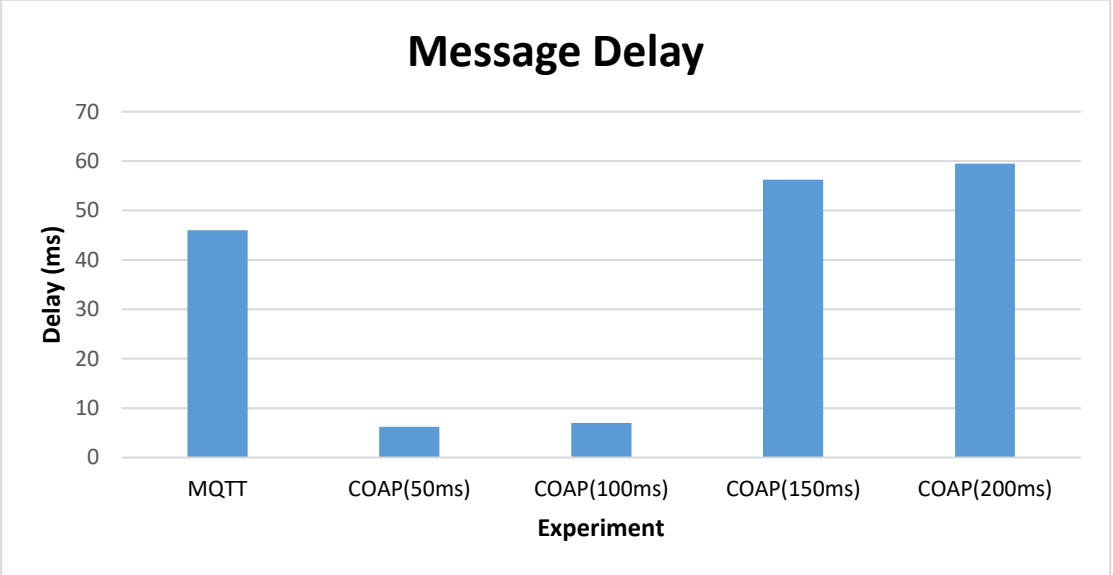

Figure 33: Delay Comparison between COAP and MQTT

CPU Allocation: Figure 34 depicts the CPU allocation for MQTTClient and for COAPClient with different polling intervals. As seen in the figure, the comparison of the two protocols in terms of this metric highly depends on the polling intervals in the COAP. In the COAP case, we have to mark that there is negative correlation between polling interval and CPU usage. CPU allocation steadily regressed through from 50ms polling to 200ms polling. MQTTClient needs less CPU resource than COAP when the 100ms and 50ms polling intervals are used. However, when the polling interval is bigger than the message preparing time, then COAP outperforms the MQTT client.

Here we compare the two protocols when the polling interval is the same as the message preparation interval. MQTTClient uses less CPU time than COAPClient with 100ms polling interval. One possible reason is as follows. COAP has client-driven protocol. Due to its HTTP elements, its client has more responsibility than a MQTT client. The COAP client contains both a request sender and a response receiver which increases computation on client. MQTT has publish/subscribe pattern, a client only listens the incoming messages. Karagiannis et al. (V. Karagiannis, 2015) also supports this finding. According to them, additional message processing required by request/response pattern puts extra execution for CPU. Another possible reason of the CPU allocation difference is the packet loss cases. COAP handles it at application layer because of UDP's deficiency (Z. Shelby C. K., 2013). Retransmission mechanism is implemented within HTTP adaptation (Z. Shelby C. K., 2013). MQTT handles it at network layer. TCP protocol handles packet loss cases.

In conclusion, when the message preparation time is known, using a COAP client with longer polling intervals would require less CPU resource than a MQTT client.

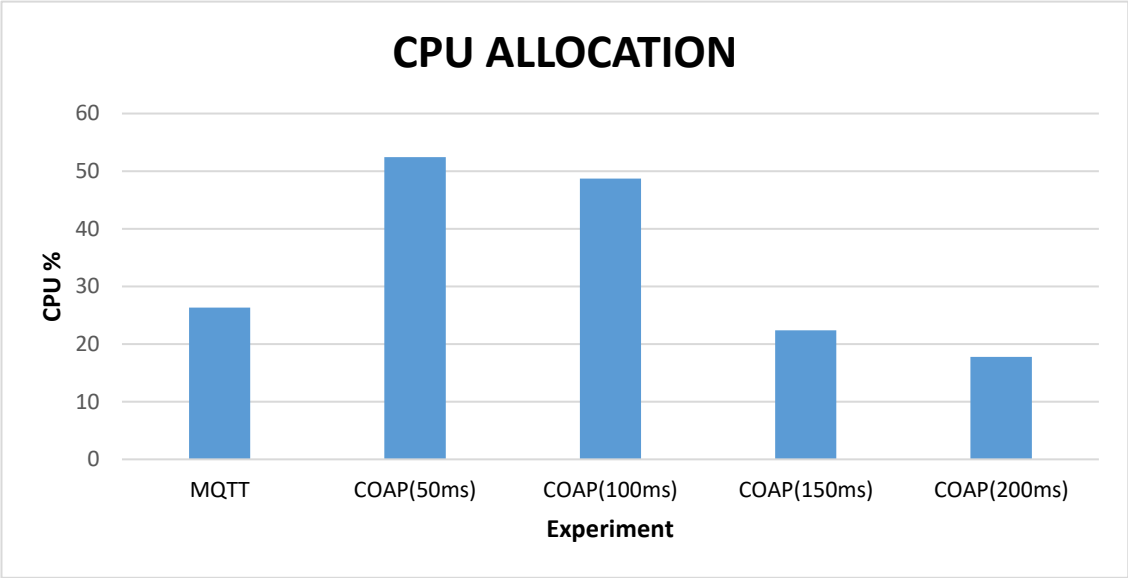

Figure 34: CPU Allocation Comparison between COAP and MQTT

Memory: As Figure 35 shows, COAP uses more memory than MQTT. COAP Client has more software component than MQTT due to its implementation. This situation is the result of design choices of protocol designers. In COAP, the client has the initiative to start the communication which causes heavier application with respect to resource usage. COAPClient has to send request message to the server and process response message. Since COAPClient application size does not depend on the polling intervals, memory usage stays steady across our experiments with the exception of 50ms polling.

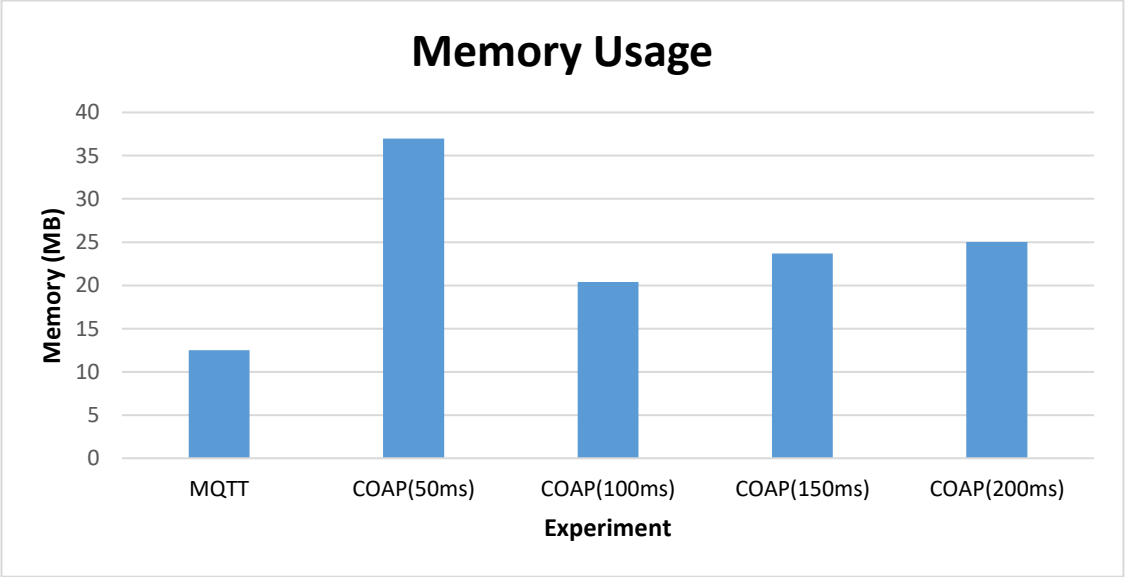

Figure 35: Memory Usage Comparison between COAP and MQTT

Battery Usage: MQTTClient is more energy efficient than its COAP counterpart as shown in Figure 36. This result is predictable due to COAPCLient's memory resource usage is higher than MQTTCLient. Also, since at the lower polling intervals COAP requires more CPU time, the battery usage is higher in these intervals. Our findings related to battery usage are supported by Karagiannis et al. (V. Karagiannis, 2015). It is also affected by retransmission and resource discovery cases. Since UDP does not provide any reliability or integrity features, these features are provided in application level in the COAP implementation. Specifically, HTTP is part of its implementation. Any packet loss or reconnection causes more resource usage which is end up more energy consumption. Battery usage has positive correlation with CPU allocation. As CPU allocation gets high, power consumption also increases. As their combined (CPU and Memory) resource usages get closer, MQTT and COAP protocols' power consumptions also get close to each other.

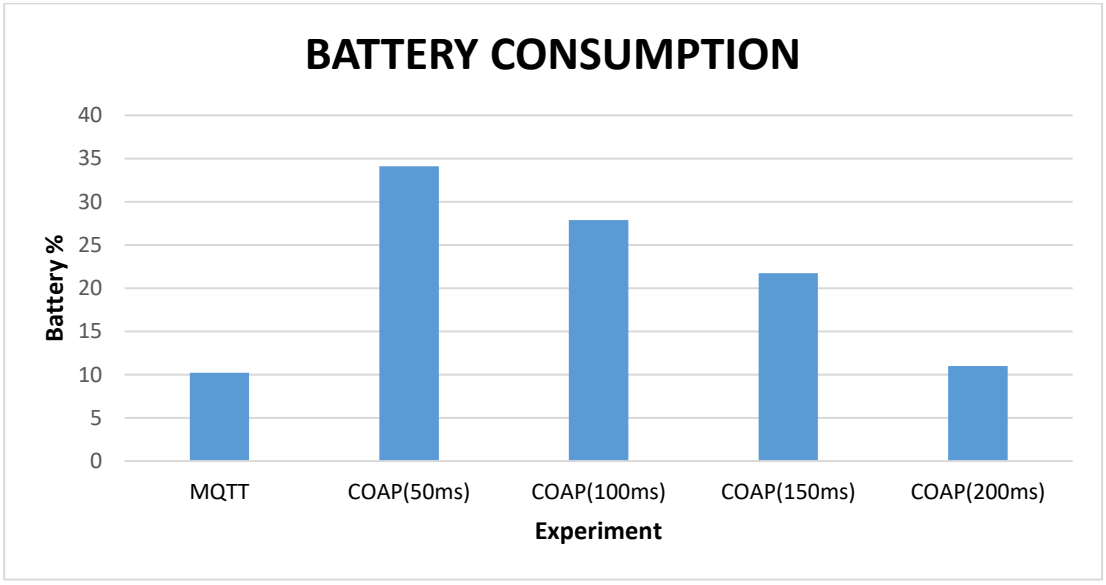

Figure 36: Battery Consumption Comparison between COAP and MQTT

CHAPTER 6

CONCLUSION

We aim to compare two prominent lightweight network protocols; MQTT and COAP. In order to achieve this goal, we designed and provided a common hardware setup and software infrastructure. A PC, a mobile device and a router formed our hardware setup. To avoid constructional threat, we have developed the software implementing both protocols. Our software applications had dependencies to two common frameworks. They are .NET and Android frameworks. We developed our server applications on .NET, client applications on Android framework.

We studied according to GQM approach. The goal is to determine strengths and weaknesses of MQTT and COAP protocols on application level. The following research questions are chosen:

- RQ1: Which protocol is resource efficient for mobile hardware aspect?
- RQ2: Which protocol performs better for transfer time aspect?

Our context in the experiments is medical facilities. We simulated them with HL7 messages. We analyzed our findings in aspect of clients. In other words, we examined the effect of the protocols on the nodes.

For RQ1, we measured CPU allocation, memory usage and battery consumption. According to our findings, MQTT consumes lower hardware resource than COAP, unless polling interval is bigger than message generating interval. At higher intervals memory usage still favors MQTT; however CPU allocation results are reversed. COAP fares better than MQTT at higher polling interval with respect to CPU allocation. Power consumption of COAP reduces with increase of interval and its 200ms polling results are similar with MQTT. CPU allocation and memory usages of MQTT and COAP have not been investigated before in the literature, to the best of our knowledge the power consumption results conflicts with the literature (S. Bandyopadhyay, 2013). It is understandable because Bandyopadhyay et al. (S. Bandyopadhyay, 2013) considers entire systems power consumption. On the other hand, Karagiannis et al. (V. Karagiannis, 2015) claims by using publish/subscribe pattern MQTT nodes require less message processing which results extending battery life. We only focused on battery consumption at the client side. Since server machines are provided with UPS, power is not constraint resource for them. On the other hand, our client consists of mobile device for which power is the greatest constraint. In conclusion, under similar delay or polling conditions MQTT requires less hardware resources than COAP. For RQ2, we measured the message delay; COAP is clearly faster than MQTT. Although there are some fluctuations at COAP's results, it still faster than MQTT unless polling rate is higher than data generate rate. These results are compatible with the literature (D.Thangavel, 2014) (S. Hamida, 2015) (N. De Caro, 2013). Figure 37 shows overall comparison conclusion of our experiments.

The areas of use of the protocols are determined by protocols strengths. In general, publish/subscribe pattern suits better for IoT than request/response pattern (S. Hamida, 2015). When message delay is the most important factor and overweights hardware constraints, COAP is apparent solution. As for the CPU metrics, on larger polling intervals, the resource usage of COAPClient is regressed. However, it costs COAP's latency advantage. There is a tradeoff between CPU allocation and latency for COAP. By increasing polling interval COAP we can achieve lower CPU allocation than MQTT. But if intended solution requires balance between latency and CPU MQTT should be used. Also, MQTT would be better choice when it comes to reliability (D. Yi, 2016) because of its TCP features (Cohn, 2011). COAP also has reliability mechanism; but TCP is found more dependable (N. De Caro, 2013). Although there are several initiatives COAP-based patient monitoring (M. Cha, 2017) and sensor network (H. Khattak, 2014) applications; MQTT still stands more preferable solution. This conclusion is supported by several studies (M. Tucic, 2014) (D. Yi, 2016) (V. Karagiannis, 2015) (S. Hamida, 2015).

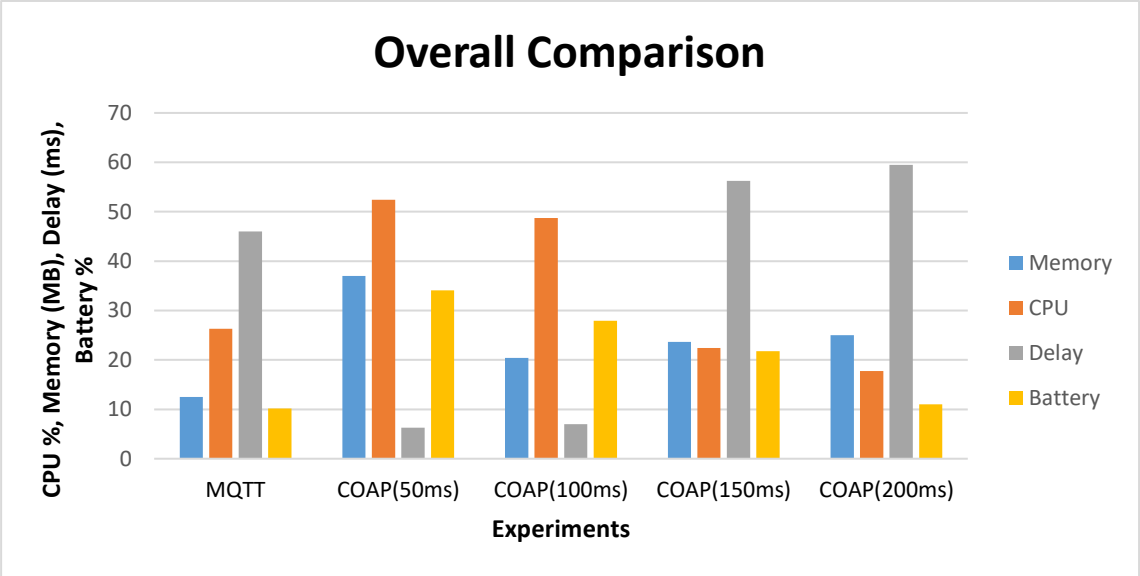

Figure 37: Overall Comparison of MQTT and COAP

Patient related information messages would be carried on MQTT. If total hardware resource usages are considered as a whole, mobile applications which are used by medical staff should be implemented with MQTT due to its lower combined resource usages. Although at higher intervals COAP uses less CPU time, combined CPU, memory and battery requirements are higher than MQTT. Ambulance and medical supply movement information related communications should be carried over COAP since time is essence. Nearly all studies confirm low-latency advantage of COAP over MQTT when COAP sends requests at the same time MQTT publishes. In other words, if the message is ready and both protocols try to get it at the same time; COAP performs better than MQTT. However, there are several limitations in hybrid approach. Since each protocol has strong sides we suggest hybrid approach for medical facilities. Currently MQTT and COAP are not interoperable. In other words, they cannot communicate each other. Each node is able to process both protocols and they communicate via middleware which is capable of processing messages of both

protocols (R. Sutaria, 2013). Former approach is not practical because nodes are constraint devices and this solution negates all advantages of lightweight messaging protocols. Latter approach is more suitable; however, it may add additional latency and increase overall system complexity by an adding extra layer.

There are limitations for our thesis. They could be classified into two categories, Hardware and software limitations. First, hardware limitations consist of closed network, PC-mobile device specifications and router capabilities. We ran the tests on closed private network. On public network, security should be included as a metric, however security is not a concern of this study, and it does not cause any threat to validity. Adding that, we used router that has data rate of 150 Megabit per second (Mbit/s)[vi]. Any differences at data rate can cause changes in delay experiments. Also our client applications run on mobile device that has Quad Core 1200 megahertz (MHz) and 4 gigabyte (GB) memory. These specifications directly effect on CPU allocation and memory usage. Numerical results of CPU allocation and memory usage have cordial relationship with hardware specifications. Power consumption has direct relations with mobile device's battery. In this thesis we use 1800 miliampere-hour (mAH) battery on mobile device. Numeric results depend heavily on hardware specifications. They may change under different hardware setup. For example, if client application runs on device that has more powerful processor will allocate lower CPU. Likewise, more capable battery will lose lower percentage of power during test runs. Because of these facts we focus on comparison rather than numeric values. Second, software limitations comprise of fixed message size, .NET and Android framework. In our tests, the server publishes/ responses HL7 messages. Those messages have fixed sizes. This situation is dictated by HL7 standards. Due to our context decision, we did not cover protocols' responses for different message sizes. We develop our server applications on .Net framework v4.5 with C# and client applications on Android framework v4.4.2 with Java. Since covering all available frameworks is impractical, we provide common software infrastructure. Optimization of framework directly impacts our experiments. Applications resource usages depend on framework's optimization. It affects CPU allocation and memory usage. Resource usage also affects power consumption, because if resource usage increases device will definitely consume more power and vice versa. Those threats do not affect our results because at any rate, both protocols are tested under same software infrastructure. Although it is subjective matter, we think MQTT has easier interface for application development.

In the future work, we plan to design for complete communication system for medical facilities with hybrid use of both protocols. Also, MQTT and COAP can be viable options to vehicle-to-vehicle (V2V) communication.

REFERENCE

1. Ashton, K. (2009). That ‘internet of things’ thing. RFiD Journal, 22(7), 97-114.
2. Evans, D. (2012). The Internet of Things How the Next Evolution of the Internet is Changing Everything (April 2011). White Paper by Cisco Internet Business Solutions Group (IBSG).
3. Colitti, W., Steenhaut, K., & De Caro, N. (2011). Integrating wireless sensor networks with the web. Extending the Internet to Low power and Lossy Networks (IP+ SN 2011).
4. Locke, D. (2010). Mq telemetry transport (mqtt) v3. 1 protocol specification. IBM developerWorks Technical Library.
5. Shelby, Z., Hartke, K., Bormann, C., & Frank, B. Constrained application protocol (CoAP), draft-ietf-core-coap-18 sl: IETF 2013.
6. Bergmann, O. (2012). libcoap: C-Implementation of CoAP. *URL: http://libcoap. sourceforge. net, Date of access 13.09*.
7. Thangavel, D., Ma, X., Valera, A., Tan, H. X., & Tan, C. K. Y. (2014, April). Performance evaluation of MQTT and CoAP via a common middleware. In Intelligent Sensors, Sensor Networks and Information Processing (ISSNIP), 2014 IEEE Ninth International Conference on (pp. 1-6). IEEE.
8. Bandyopadhyay, S., & Bhattacharyya, A. (2013, January). Lightweight Internet protocols for web enablement of sensors using constrained gateway devices. In Computing, Networking and Communications (ICNC), 2013 International Conference on (pp. 334-340). IEEE.
9. Tucic, M., Pavlovic, R., Papp, I., & Saric, D. (2014, November). Networking layer for unifying distributed smart home entities. In Telecommunications Forum Telfor (TELFOR), 2014 22nd (pp. 368-371). IEEE.
10. Yi, D., Binwen, F., Xiaoming, K., & Qianqian, M. (2016, October). Design and implementation of mobile health monitoring system based on MQTT protocol. In Advanced Information Management, Communicates, Electronic and Automation Control Conference (IMCEC), 2016 IEEE (pp. 1679-1682). IEEE.
11. Caldiera, V. R. B. G., & Rombach, H. D. (1994). Goal question metric paradigm. Encyclopedia of Software Engineering, 1, 528-532.
12. Frigieri, E. P., Mazzer, D., & Parreira, L. F. M2M Protocols for Constrained Environments in the Context of IoT: A Comparison of Approaches. In International Telecommunications Symposium.
13. Shelby, Z., Hartke, K., & Bormann, C. (2014). RFC 7252—The Constrained Application Protocol (CoAP). Internet Engineering T ask Force (IETF).
14. Chen, X. (2014). Constrained Application Protocol for Internet of Things. URL: http://www1. cse. wustl. edu/~ jain/cse574-14/ftp/coap.

15. Durkop, L., Czybik, B., & Jasperneite, J. (2015, February). Performance evaluation of M2M protocols over cellular networks in a lab environment. In Intelligence in Next Generation Networks (ICIN), 2015 18th International Conference on (pp. 70-75). IEEE.
16. Karagiannis, V., Chatzimisios, P., Vazquez-Gallego, F., & Alonso-Zarate, J. (2015). A survey on application layer protocols for the internet of things. Transaction on IoT and Cloud Computing, 3(1), 11-17.
17. De Caro, N., Colitti, W., Steenhaut, K., Mangino, G., & Reali, G. (2013, November). Comparison of two lightweight protocols for smartphone-based sensing. In Communications and Vehicular Technology in the Benelux (SCVT), 2013 IEEE 20th Symposium on (pp. 1-6). IEEE.
18. Sutaria, R., & Govindachari, R. (2013). Making sense of interoperability: Protocols and Standardization initiatives in IOT. In 2nd International Workshop on Computing and Networking for Internet of Things.
19. Kim, W., Shin, Y., & Seol, S. (2015). Smart phone assisted personal IoT service. Advanced Science and Technology Letters, 110, 61-66.
20. Cohn, R. S. (2011). A Comparison of AMQP and MQTT. Available: www.stormmq.com.
21. Hamida, S. T. B., Hamida, E. B., & Ahmed, B. (2015). A new mHealth communication framework for use in wearable WBANs and mobile technologies. Sensors, 15(2), 3379-3408.
22. Khattak, H. A., Ruta, M., & Di Sciascio, E. (2014, January). CoAP-based healthcare sensor networks: A survey. In Applied Sciences and Technology (IBCAST), 2014 11th International Bhurban Conference on (pp. 499-503). IEEE.
23. M. Cha, J. Kwon,E. Kim (2017), Implementation of Healthcare Monitoring System based on CoAP Group Communication. Advanced Science and Technology Letters vol.143 (AST 2017), pp.98-101
24. Soldani, D., & Manzalini, A. (2015). Horizon 2020 and beyond: on the 5G operating system for a true digital society. IEEE Vehicular Technology Magazine, 10(1), 32-42.
25. Kovatsch, M., Lanter, M., Shelby, M. (2014). Californium: Scalable Cloud Services for the Internet of Things with CoAP. Proceedings of the 4th International Conference on the Internet of Things (IoT 2014). Cambridge, MA, USA, October 2014
26. Orguna, B., & Vu, J. (2006). HL7 ontology and mobile agents for interoperability in heterogeneous medical information systems. Computers in biology and medicine, 36(7), 817-836.
27. HL7 v2.X Message Profiling Specification Version 2.2 (2000). Retrieved from https://www.hl7.org/documentcenter/public/standards/v22/HL7_Profile_V2r2_final.doc

28. R. A. Light, “Mosquitto: server and client implementation of the MQTT protocol,” The Journal of Open Source Software, vol. 2, no. 13, May 2017, DOI: 10.21105/joss.00265
29. Postel, J. (1981). RFC 793: Transmission control protocol, September 1981. Status: Standard, 88.
30. Postel, J. (1980). RFC 768: User datagram protocol, August 1980. Status: Standard.

[i] http://www.eclipse.org/paho/downloads.php 2017
[ii] https://github.com/smeshlink/CoAP.NET 2016
[iii] https://github.com/okleine/spitfirefox 2016
[iv] https://stackoverflow.com/questions/3118234/get-memory-usage-in-android/5562634# 5560634
[v] https://developer.android.com/reference/android/os/PowerManager.html 2017
[vi] http://airties.com.tr/datasheets/AIR5340EN_DS.pdf