Reliability Side-Effects in Internet of Things Application Layer Protocols

Bardia Safaei*, Amir Mahdi Hosseini Monazzah†, Milad Barzegar Bafroei‡ and Alireza Ejlali§
Department of Computer Engineering
Sharif University of Technology
Email: {*bsafaei, †ahosseini, ‡mbarzegar} @ce.sharif.edu, §ejlali@sharif.edu

***Abstract*—With the widespread use of IoT devices in safety-critical applications, new constraints should be addressed in designing IoT infrastructures. Reliability is one of the most important characteristics of an IoT system which should be satisfied with high consideration. The way how IoT devices communicate with each other in different layers of architecture, plays an important role in building a reliable IoT infrastructure. Maintaining the desired level of reliability in IoT applications through application layer protocols, imposes a noticeable amount of overhead to different characteristics of IoT systems. In this paper, we are going to investigate these overheads through practical evaluations on a conventional IoT infrastructure. To this end, we will compare two well-known application layer protocols, i.e., MQTT with many reliable features and CoAP with fewer reliability mechanisms, from different aspects. Conducted experiments in Arduino test-beds illustrate while MQTT provides more reliable and predictable infrastructure for IoT devices, on average, it imposes 364uW of more power consumption than CoAP for delivering the same amount of data. Furthermore, utilizing MQTT as a reliable application layer protocol, imposes 4.99x of more latency in comparison with CoAP.**

Keywords*—*Internet of Things, Reliability, Power Consumption, Performance, Protocol, Application, CoAP, MQTT.

I. Introduction

Internet of Things (IoT) is a system comprising a communicative infrastructure which connects an enormous amounts of identified, low-power embedded devices through exploitation of Internet and communication technologies without human intervention. The potential applications of IoT are widespread, starting from smart homes to smart cities, many of them require real-time and low power communications, e.g., health care applications.

McKinsey Global Institute® has estimated that the financial impact of the IoT market on the global economy may reach as much as $11.1 trillion by 2025 [1]. On the other hand the total number of smart connected devices around the world is rising rapidly, that many institutions such as IHS® reported 20.3 Billion connected "Things" will Be in Use in 2017 which would rise up to more than 75 billion at the end of 2025. As it has been illustrated in Fig. 1, based on the informations on population and total number of connected devices [2], [3], we have anticipated that there will be more than 9 smart devices per person at the end of 2025. This amount of interconnected devices would raise many challenges.

Considering the rapid growth in number of connected IoT devices, different international standardization bodies such as Internet Engineering Task Force (IETF), IEEE, The 3rd Generation Partnership Project (3GPP) standardized new protocols to meet the requirements and emerging applications of IoT. Reliability is one of the most important requirements in IoT communication protocols. In other words, data transmission in harsh environments via dynamic and lossy links is inherently unreliable, leading to excessive re-transmissions, large amount of energy consumption and long latency in the shared wired and wireless medium. Therefore, reliability plays an important role not only in delivering data in IoT infrastructures, but also due to its side-effects on performance and energy consumption of IoT devices. To this end, reliable and energy-efficient data delivery has to be considered as two important aspects of IoT applications.

The primary goal of an IoT infrastructure is to ensure effective communication between objects and build a reliable bond among them using different types of applications. Interconnecting billions of smart devices throughout the internet, necessitates a flexible layered architecture for IoT infrastructures [4]. Nevertheless, a vast number of provided architectures has not yet converged to a standard model. However, three layers i.e., Objects (also known as perception), Network, and Application Layers are in common among all of the provided architectures. Indeed, more complicated architectures has just expanded each of the mentioned three layers in to more layers to provide more accuracy.

Among these three layers, the application layer is responsible for providing services and determines a set of protocols for message passing at the application level. These protocols could reliably transfer data between nodes through exchanging acknowledgement packets (Ack) or simply refusing Acks to save other resources, e.g., power and bandwidth with the cost

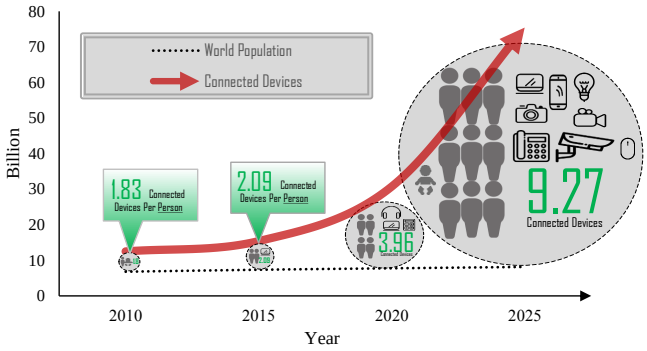


Fig. 1: Estimated Number of Connected Devices Per Person By 2025

of lower reliability.

Message Queue Telemetry Transport (MQTT) [5] and Constrained Application Protocol (CoAP) [6] are two major IoT application layer protocols which recently gained attention. These open source protocols have been designed for lightweight devices to specifically address the difficult requirements of real-world IoT deployment scenarios. These protocols employ "publish-subscribe" and "client-server" architectures respectively to transfer data between nodes.

In this paper we are going to investigate the side-effects of reliability on IoT communication protocols. To this end, we consider MQTT as our reliable protocol, since it utilizes acknowledge policy in its transmissions. As our baseline, we consider CoAP communication protocol which utilizes a poor level of reliability approaches in its transmissions. We implement both of these communication protocols in real test-bed, i.e., Arduino [7]. To evaluate the side effects of providing reliability on communication protocols, we measure power consumption and performance of our test-beds during the transmissions.

In this study, we are going to clarify the existing trade-offs between reliability and resource constraints in a real IoT infrastructure. The evaluation results of this study would pave the way for selecting an appropriate application layer protocol for a specified IoT application. For example, although MQTT is a reliable protocol, our evaluation results show that it may not be an appropriate choice for IoT applications where timing and power constraints are more important than reliability. Indeed, for these kinds of applications it has been shown that CoAP would be a much better choice by providing less power consumption and node to node latency.

The rest of this paper is organized as follows: Section II represents a background on the application layer protocols and their structure in IoT, concentrating on MQTT and CoAP. In Section III, the system setup and experiment methodology has been discussed. Experimental results are stated in Section IV. Finally, this paper is concluded in Section V.

II. IoT Application Protocols

Delivered services by IoT infrastructures cover a broad range of applications from non-critical applications like informing the level of humidity or brightness in a smart greenhouse to critical applications like monitoring and analyzing biological data from patients in real-time health-care services. In each of these application, there are a number of customers which require desired services. The way how IoT devices connect to each other, plays an important role in satisfying the services. Basically, the IoT architecture consists of three layers, i.e., object layer, network layer, and application layer. Among them, the last one is responsible for managing the IoT connections in level of application via applied protocols. As it is illustrated in Table I, in a typical IoT architecture, application protocols act as an interface between network layer and the services in the application layer. The most widespread application protocols used in IoT infrastructures are MQTT, CoAP, Extensible Messaging and Presence Protocol (XMPP) [8], Advanced Message Queuing Protocol (AMQP) [9] and Data Distribution Service (DDS) [10].

According to the estimated market share of IoT applications in 2025, health care, manufacturing and electricity will dominate other applications with 41%, 33% and 7% of contributions, respectively [4]. In these applications, the required application layer connectivity which is delivered via intended protocols, should be accompanied with reliability, low energy consumption and low delay to prolong the lifetime of the sensors and to minimize the probability of disconnection. Among the mentioned conventional application layer protocols, AMQP with 8 byte header size imposes high network overhead and power consumption and XMPP does not provide any Quality of Service (QoS). On the other hand, although DDS provides many desirable concepts such as reliability, real-time features and providing 23 kinds of QoS [4], its complexity of design could make it a low priority for low cost IoT infrastructures. Therefore MQTT and CoAP could be preferred against the other protocols. These two protocols are somehow the two sides of a same coin; both are an application layer protocol but with different characteristics. In the following, we will focus on introducing these two protocols more deeply.

A. CoAP

Due to the improperness of HyperText Transfer Protocol (HTTP) in resource constrained systems such as embedded devices and IoT applications, the IETF Constraint RESTful Environment (CoRE) Working Group designed CoAP for application layer communications [6]. CoAP is a client-server, web transfer protocol optimized for connecting distributed nodes in constrained environments. As HTTP, CoAP has the responsibility to transfer documents but with a lower overhead and taking into account the resource, power and energy constraints. This is due to the employment of UDP protocol instead of TCP protocol in the structure of CoAP and mapping existing string fields of HTTP into fields with dimensions as small as a bit. Consequently, CoAP packets are far smaller than HTTP packets. While CoAP benefits from small packet sizes, since it is based on UDP protocols, it suffers from reliability challenges. Indeed, UDP based protocols such as CoAP, provide less reliable transactions because the loss or failure of a packet is not important for the transmitter. To overcome this challenge, CoAP exploits a two layer approach in its architecture: *1) Transaction Layer* and *2) Request/Response Layer*. The Transaction layer is responsible for single message exchange between end nodes and the Request/Response layer's duty is to handle request/response transmissions and also the

TABLE I: Operating layers in an IoT infrastructure

Application Layer	Operating Systems	Contiki	Tiny OS	Riot OS	Lite OS	Android			
	Services	Smart City: Building, Transport, Traffic	Environment: Natural Resources, Disaster	Law	Health Care	Industrial	Power Grid		
	Protocols	MQTT	CoAP	XMPP	AMQP	DDS			
Network Layer	Routing Protocol	Routing Protocol for Low Power and Lossy Networks (RPL)							
	Network Protocol	6LowPAN	IPV6						
	Link Related Technology	IEEE 802.15.4							
Objects Layer (Perception)	Sensors and Actuators	Arduino	Raspberry Pi	ESP8266	Beaglebone Black	Intel Edison	Telos B		
	Physical Layer Protocols	Zigbee	EPCglobal	IEEE 802.15.4	Z-Wave	BLE			
	Device Communication Technologies	RFID	GSM	Infrared	UMTS	LTE-A	5G		

resource management in the system [11].

To improve the reliability challenges for critical applications which exploit CoAP as their application layer protocol, CoAP delivers a user controllable reliability flag called, "Confirmable". Indeed, a poor level of reliability in CoAP could be provided through labeling a message as Confirmable. In this case, The transmitter tries to send the Confirmable message up to a maximum number of times, according to a predefined back-off algorithm for re-transmissions, until the receiver sends back an Ack packet upon receiving the Confirmable message. This confirms the message has been received, but still lacks confirmation on whether its contents has been decoded correctly or not. On the other hand, there are "Non-Confirmable" messages which have the same applicability as the "Fire and Forget" approach in MQTT (which is going to be discussed later). In this approach, the reception of the Ack packets does not matter for the transmitter.

Furthermore, CoAP supports multicast features of UDP to provide node addressing. It exploits a client-server based architecture in which all the nodes could play as a client, server or even both. Usually clients send their requests as *GET*, *POST*, *PUT* and *DELETE* messages to the server nodes and wait for them to respond. In conclusion, some of the most promising features provided by the CoAP protocol include: 1) Resource Observation, 2) Block-wise resource transport, 3) Resource Discovery, 4) Integration with HTTP, and 5) Security. Considering the security in CoAP, it should be noticed that CoAP employs Datagram Transport Layer Security (DTLS) protocol [12] to secure its communications through message encryption. DTLS is a protocol based on TLS [13] which secures UDP.

B. MQTT

MQTT is an open source, application layer, publish-subscribe messaging protocol, designed for lightweight Machine to Machine (M2M) and IoT communications. Initially, MQTT was developed by IBM® in 1999 and implemented across a variety of industries. Two examples of well-known projects which exploited MQTT as an application layer protocol are Smart Lab [14] and Flood Net [15]. The architecture of MQTT is based on a publish-subscribe model and its purpose is to exchange messages among clients through a common broker with an inherent reliability.

In a publish-subscribe architecture, there is a UTF-8 string which is used by the broker to filter messages for each connected client, known as a "topic". A client (subscriber) sets a request with a desired topic of interest to the server (known as the broker); Immediately after reception of this request, the node which produces data (publisher) sends the data (related to the specified topic) towards the broker and afterwards the broker forwards the gathered information to the subscriber. In this context, each message is published under a defined topic (an address) which could be accessed by all other nodes, subscribed to that topic. It should be noted, there isn't any constraints on the number of subscriptions in defined topics for a single node in the system. All of the nodes which are subscribed to a topic by the broker, would be able to receive the related messages. This connection is being established through transmitting a *CONNECT* message. *CONNECT* messages require an acknowledgment through a *CONNACK* message [11]. In publish-subscription, publishers do not need to have any knowledge about the identity and even the existence of subscribers and vise versa, so it's called a "loosely-coupled" architecture. This decoupling provides the opportunity for better scalability than traditional clientserver models.

Providing a reliable transaction is one of the core goals in MQTT. MQTT provides reliable communications by employing TCP as its under layer transport protocol. This makes MQTT a top priority for devices which want to communicate in harsh operating environment, e.g., IoT and WSNs. Depending on how reliably messages should be delivered to their destinations, MQTT provides three levels of QoS:

1) *QoS(0)*: This is the simplest form of QoS provided in MQTT. In This level of QoS, which is also called a Fire and Forget mechanism, once the transmitter sends the message, it will not wait to check the delivery of the message. Also, in addition to the message delivery, the integrity of the delivered message does not matter for the transmitter.
2) *QoS(1)*: This level is suggesting a more reliable communication. In this case, the sender will (re-)transmit the packet as long as it receives the Ack packet. In this level, it is probable for a receiver to receive duplicates of packets. Accordingly, this QoS is delivering the message for *at least once*.
3) *QoS(2)*: The highest degree of reliability is provided in QoS(2), which ensures not only the reception of the packets (by receiving Ack packets from the receiver) but also that they are delivered only once to the destinations [16]. Delivering only one copy of the message to the destination will reduce the bit-error-rate (BER) over the wired or wireless media and improve the packet delivery probability. Thus, QoS(2) provides a more reliable communication than QoS(1).

As CoAP's Non-Confirmable messages, the QoS(0) does not guarantee any reliability, while MQTT's QoS(1) could operate in the same level as CoAP's Confirmable messages and guarantee an Ack-based reliability. It should be noted that MQTT's QoS(2), which guarantees the non reception of duplicated packets by the receiver, does not correspond to any similar QoS levels in the CoAP protocol [11]. The publish-subscribe architecture in MQTT protocol enables the support for different types of traffics mainly one-to-one, one-to-many and many-to-one.

Besides the promising features of MQTT, a major issue in MQTT is the concept of topic matching. Topic matching will indicate its importance as a node subscribes to a defined topic. In MQTT, topics are designed as the same as a file system. In other words, they exploit a hierarchy, e.g., "kitchen/oven/temperature". In subscription, wild cards are allowed to determine the name of the desired topics. In Table II, a list of allowed wildcards for the topic matching has been explained briefly with their use-cases. Topic matching in MQTT, imposes extra processing overhead to the nodes and thus intensifies the pace of wearing-out of nodes and consequently reduces the reliability of the entire system.

In addition to the provided levels of reliability (QoS) for

TABLE II: Topic Matching in MQTT

Wildcard	Description	Example		
		Topic	Match	Not Matched
+	Employed for single-level topic matching	/CE/+/Temperature	/CE/Lab1/Temperature	/CE/Lab2/Section3/Temperature
#	Employed for multi-level topic matching	/CE/GroundFloor/#	/CE/GroundFloor/Lobby/Temperature	/CE/1st-Floor/Room1/Temperature

transmitting messages in MQTT, this protocol also considers some protective approaches in initializing a connection or disconnecting the currently connected nods which makes it a proper protocol for reliable IoT infrastructures. Indeed, at the beginning of a communication, each of the clients which want to be connected to the broker, specify a message known as "Last Will and Testament (LWT)" message and send it to the broker. This message is being employed to inform all the other clients about an abrupt disconnection of a client, which was subscribed to a shared topic with all of them. The LWT includes the intended topic and is stored by the broker. This message would not be multicasted until a sudden disconnection of a node. On the other hand, if a client tries to disconnect from the network in a graceful manner through transmitting a "DISCONNECT" message, LWT will be discarded. This feature would provide a more reliable transaction, because when a node is lost in the network and other nodes are not informed about it, they may send data packets to a lost node while there isn't any receiver at the opposite side. Furthermore, security is also another feature provided by the MQTT protocol. Infact, as TCP is the under layer protocol for MQTT, brokers and clients can benefit from message encryption via SSL and TLS protocols.

III. System Setup

In this section, first we will provide a detailed information about the employed hardwares for setting up the test-bed which is later used for evaluating the efficiency of the most promising IoT communication protocols in this study. Then, we will introduce our experiment methodologies.

A. Test-bed Introduction

Fig. 2 includes the modules that have been used in this study. We utilize Arduino as our main platform for constructing a simple IoT infrastructure. Arduino is an open-source platform with widespread applications as a development board. There are many variations of Arduino in the market. Among them, Arduino Uno has the most popularity for IoT applications due to its specifications. Fig. 2 (a) represents the Arduino Uno platform which has been exploited in this study. The detailed specifications of this platform could be found in Table III. For programming the Arduino platform, the provided Integrated Development Environment (IDE) which comes with the device has been used. As it is illustrated in Table III, the promising features delivered by Arduino Uno and the cheep price of this platform, makes it an appropriate candidate for developing many IoT applications. One of the interesting characteristics of this platform is its ability to work with peripheral devices through its USB port.

Wireless communications in the free space is one of the aspects of IoT infrastructures. Hence, we are going to setup the test-bed through the WiFi technology. To this end, the Arduino WiFi module, i.e., ESP8266 has been employed. This module

TABLE III: Technical Specifications of Arduino Uno

Parameter	Value
Microcontroller	ATmega328P
Operating Voltage	5 v
Digital I/O Pins	14 (of which 6 provide PWM output)
PWM Digital I/O Pins	6
Analog Input Pins	6
DC Current per I/O Pin	20 mA
DC Current for 3.3V Pin	50 mA
Flash Memory	32 KB (ATmega328P) of which 0.5 KB used by bootloader
SRAM	2 KB (ATmega328P)
EEPROM	1 KB (ATmega328P)
Clock Speed	16 MHz
Length	68.6 mm
Width	53.4 mm
Weight	25 g
Price	22 $

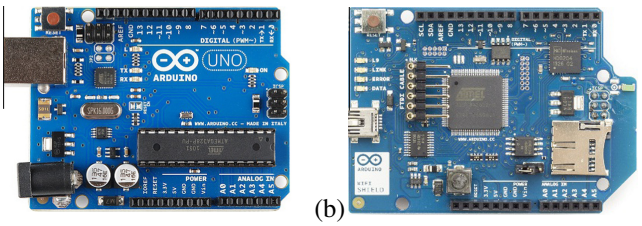
TABLE IV: Technical Specifications of ESP8266

Parameter	Value
Input Voltage	5 v
Frequency	2.4 GHz
Range	100 m
Communication Protocols	IEEE 802.11b/g
Wake Up and Transmit Packet	<2 ms
Leakage Power	<1.0 mW

can be connected to the Arduino platform through its USB port. Fig. 2 (b) shows the ESP8266 module. This module is equipped with integrated TCP/IP protocol stack which easily enables the connection of existing Arduino micro-controllers to the WiFi network. ESP8266 in addition to providing the ability to communicate with other processors which run intended applications, could host an application itself. Table IV shows the technical specifications of ESP8266.

B. Experiment Methodologies

To investigate the side-effects of CoAP and MQTT protocols, we have established a simple scenario using the platforms introduced in the previous sub-section. Indeed, we have employed two Arduino platforms with their WiFi module, acting as the two end-nodes of the network, i.e., A and B. These nodes are located at a distance of 1m from each other. We conduct our experiments in standard room conditions. Note that the mentioned scenario could be extended to accommodate multiple nodes. In case of MQTT, A is generating data packets as a publisher and simultaneously is playing as a subscriber to



(a) (b)

Fig. 2: Test-bed modules, (a) Arduino Uno, and (b) ESP8266 WiFi module

receive required data packets according to the intended topic. On the other hand, B is acting as a broker which receives the produced data from the publisher and forwards it to the subscriber.

In case of CoAP, A is acting as a client which is setting a request or transmitting data packets, and B is serving the node A through accomplishing the required task. In this paper, in order to fairly compare MQTT and CoAP, we have limited the MQTT evaluations to the communication between the Broker and the subscriber. More over, in both cases of evaluations (MQTT and CoAP), the size of the payloads in the packets has been set to 14 bytes of data.

IV. Results and Discussion

In this section, we will discuss the observations which have been made through the experiments and compare the performance of MQTT and CoAP in terms of communication power consumption, latency and the level of predictability provided by them to determine the costs of their deployment in real-time IoT applications. Their efficiency has been also studied. In the following subsections, we have investigated the side-effects of utilizing a reliable application layer protocol on power consumption, latency, efficiency and predictability.

A. Power Consumption

Due to the nature of IoT systems, power consumption plays an important role for the node and in general, for the system's life time. Many activities in an IoT system would lead to power and energy consumption. These activities could be related to different operating layers, e.g., transceiver modules, CPU processes, routing functionalities and communication protocols. In this part, the amount of power consumed in a communication by the MQTT as a reliable application protocol and CoAP as a less reliable protocol, has been compared. As it has been illustrated in Fig. 3, on average, MQTT consumes nearly 364uW of more power than CoAP. This issue comes from the reliable nature of MQTT protocol which imposes extra overhead to the network. Indeed, utilizing TCP as an under-layer protocol in MQTT, adds flow control mechanism to the communication. Flow control provides reliable and ordered delivery of data packets as well as congestion control. Flow control enables the receiver to advertise its available buffer size to the transmitter. To this end, in the initialization phase of a socket connection and before sending any data, TCP requires three packets to set up the connection. These flow control transactions impose longer strings of data and consequently more power consumption to the communication. On the other hand, CoAP has been built on top of UDP and does not support any flow control mechanism. In addition, the inherent acknowledgement and handshaking mechanisms utilized in MQTT will lead in to more power consumption in MQTT.

B. Latency

According to the observations been made, as it has been depicted in Fig.4, it takes about 4.99x more time for a client to receive the contents through MQTT protocol. However CoAP is serving the requests more quickly. Experiments show that on average, CoAP is serving the request after 70.4ms while it takes 351.8ms for MQTT to accomplish the same

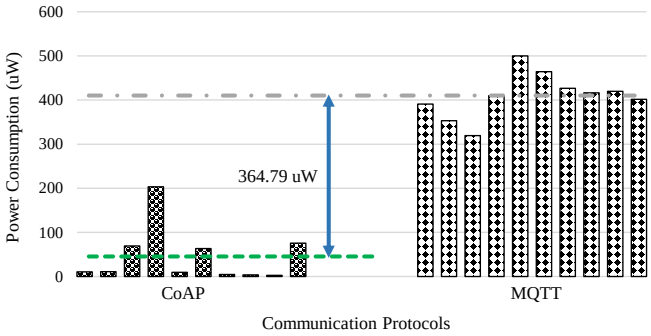


Fig. 3: Evaluated power consumption in different experiments for MQTT and CoAP

mission. One of the major reasons for this difference in the latency of packet delivery in MQTT and CoAP, is the sliding window mechanism employed by the TCP flow control and also the three-way handshaking required for initializing the communication in MQTT. According to the sliding window characteristics, the transmitting side of a TCP based connection, must stop sending further packets, until all the packets in the current sending window are acknowledged by the receiving system. After reception of data, the receiver should send an acknowledge accompanied by a new receiving window which informs the available number of bytes in its buffer. This mechanism would prevent congestion on the receiving side and accordingly avoids dropping packets by the receiver. This will lead to more reliability but with a cost of consuming more power, and longer response time.

C. Predictability

In IoT systems, the quality of the infrastructures behaviour not only depends on the functionality of them, but also depends on how quickly they react to the environment alterations. Therefore timing and predictability could play an essential role specially for the IoT applications with timing constraints. Indeed, in addition to reliability, there are a quite number of IoT applications which need to be predictable. A suitable example for such applications is Health-care and avionic services. The term predictable refers to systems which guarantee to provide the output values with the certain amount of delay. While to the best of our knowledge, communication protocol's predictability has not been considered in the previous literature, we were motivated to compare MQTT and CoAP protocols from predictability point of view. To this end, the information

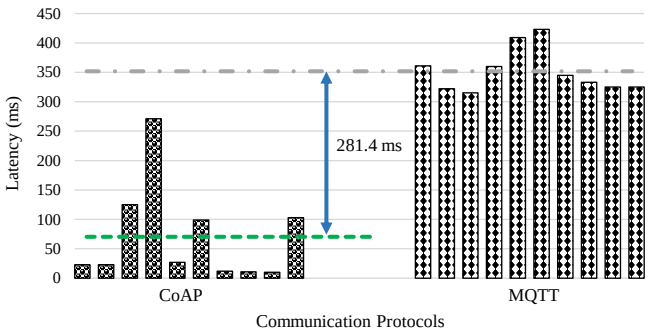


Fig. 4: Evaluated latency in different experiments for MQTT and CoAP

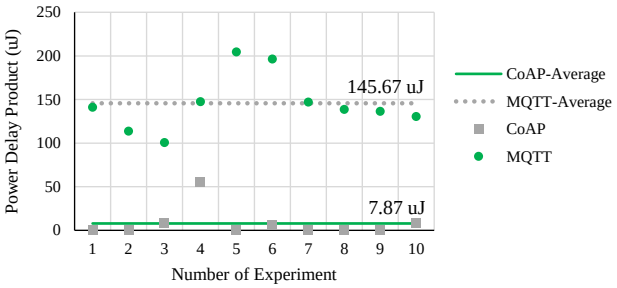


Fig. 5: Evaluated PDP in different experiments for MQTT and CoAP

gathered by the latency analysis has been exploited to measure the variance of response times for all the experiments. According to the outcomes, MQTT outperforms CoAP by 80% improvement in variance of latencies.

D. Efficiency

In addition to the existing trade-off among reliability and power consumption, there is also a trade-off between latency and power consumption. How quickly the system acts and the consumed power, are two building blocks of a systems efficiency. Hence, in IoT and embedded systems with high efficiency requirements, both features are equally important and have to be considered simultaneously. To distinguish such systems and technologies from their efficiency point of view, a single parameter known as the power-delay product (PDP), impressed by both, the power and the latency is being used for many decades [17]. The goal of minimizing the PDP is to reach an optimal energy utilization for a system. To this end, we have conducted experiments to compare the communication efficiency of the intended application layer protocols via PDP. In this context, observations illustrated in Fig.5 unveiled that, on average, CoAP is more efficient with 7.87uJ while MQTT requires 145.67uJ for its communications.

V. Conclusion and Future Works

The impact of IoT applications on different aspects of human's daily life is getting bolder. According to estimations, there will be more than 9 devices per person at the end of 2025. Collaboration of these devices enables IoT to serve the clients in different applications, including safety-critical applications. Accordingly, new constraints such as reliability should be considered in such systems. Providing reliability in IoT systems could be achieved by employing reliable mechanisms at different layers of its architecture, including application layer protocols. Maintaining the desired level of reliability in IoT applications via application layer protocols, imposes a noticeable amount of overheads. In this paper, we have investigated the side-effects of providing such a reliable application layer connection, by evaluating two well-known application layer protocols, i.e., MQTT with many reliable features and CoAP with less reliability mechanisms. We have conducted our evaluations on a real-world test-bed composing of reputed Arduino platforms. Our observations illustrated while MQTT provides more reliable and predictable infrastructure for IoT devices, on average, it imposes 364uW of more power consumption than CoAP for delivering the same amount of data. Furthermore, utilizing MQTT as a reliable application layer protocol, imposes 4.99x of more latency in comparison with CoAP. Overall, MQTT protocol is suitable for IoT systems with substantial reliability requirements were power and performance issues are not a consideration while CoAP protocol is an appropriate choice for real-time IoT systems with high power and performance constraints.

As a future work, due to the nature of IoT which constraint devices mainly operate on a wireless medium with high loss rates and low reliability, the concept of Low Power and Lossy Networks (LLN) is getting more attraction among academic societies. In this context, the Routing Protocol for Low power and Lossy Networks (RPL) has emerged as an appropriate routing mechanism for LLNs. Our goal is to implement RPL on real-world test beds and evaluate its performance in future studies.

References

[1] J. Manyika, M. Chui, P. Bisson, J. Woetzel, R. Dobbs, J. Bughin, and D. Aharon, "The internet of things: Mapping the value beyond the hype," in *McKinsey Global Institute*. McKinsey, 2015, p. 3.

[2] IHS®, "Internet of things (iot) connected devices installed base worldwide from 2015 to 2025," 2016.

[3] United Nations®, "World population prospects: The 2015 revision population database," 2016.

[4] A. Al-Fuqaha, M. Guizani, M. Mohammadi, M. Aledhari, and M. Ayyash, "Internet of things: A survey on enabling technologies, protocols, and applications," *IEEE Communications Surveys & Tutorials*, vol. 17, no. 4, pp. 2347–2376, 2015.

[5] D. Locke, "Mq telemetry transport (mqtt) v3. 1 protocol specification," in *IBM developerWorks*. IBM, [Online]. Available: Http://Www.Ibm.Com/Developerworks/ Webservices/Library/Ws-Mqtt/Index.Html, 2010, pp. 1–42.

[6] Z. Shelby, K. Hartke, C. Bormann, and B. Frank, "Constrained application protocol (coap).draft-ietf-core-coap-18," in *Internet Eng. Task Force (IETF)*. IETF, 2013, pp. 1–118.

[7] Arduino®, "Arduino uno development boards," 2017.

[8] P. Saint-Andre, "Extensible messaging and presence protocol (xmpp): Core, request for comments: 6120," in *Internet Eng. Task Force (IETF)*. IETF, 2011, p. 6120.

[9] Adv. Open Std. Inf. Soc. (OASIS), "Oasis advanced message queuing protocol (amqp) version 1.0," 2012.

[10] Object Manage. Group (OMG), Available: http://www.omg.org/spec/DDS/1.2/, "Data distribution services specification, v1.2," 2015.

[11] N. De Caro, W. Colitti, K. Steenhaut, G. Mangino, and G. Reali, "Comparison of two lightweight protocols for smartphone-based sensing," in *Communications and Vehicular Technology in the Benelux (SCVT), 2013 IEEE 20th Symposium on*. IEEE, 2013, pp. 1–6.

[12] E. Rescorla and N. Modadugu, "Datagram transport layer security version 1.2," 2012.

[13] T. Dierks, "The transport layer security (tls) protocol version 1.2," 2008.

[14] J. M. Robinson, J. G. Frey, A. J. Stanford-Clark, A. D. Reynolds, and B. V. Bedi, "Sensor networks and grid middleware for laboratory monitoring," in *e-Science and Grid Computing, 2005. First International Conference on*. IEEE, 2005, pp. 8–pp.

[15] D. DeRoure, "Improving flood warning times using pervasive and grid computing," *submitted to quarterly of Royal Academy of Engineering, UK*, p. 1, 2005.

[16] U. Hunkeler, H. L. Truong, and A. Stanford-Clark, "Mqtt-sa publish/subscribe protocol for wireless sensor networks," in *Communication systems software and middleware and workshops, 2008. comsware 2008. 3rd international conference on*. IEEE, 2008, pp. 791–798.

[17] A. P. Chandrakasan, S. Sheng, and R. W. Brodersen, "Low-power cmos digital design," *IEICE Transactions on Electronics*, vol. 75, no. 4, pp. 371–382, 1992.