Performance evaluation of MQTT and CoAP via a common middleware

Conference Paper · April 2014

DOI: 10.1109/ISSNIP.2014.6827678

5 authors, including:

Xiaoping Ma
National University of Singapore
3 PUBLICATIONS **459** CITATIONS

SEE PROFILE

Hwee-Xian Tan
Singapore Management University
43 PUBLICATIONS **1,161** CITATIONS

SEE PROFILE

Performance Evaluation of MQTT and CoAP via a Common Middleware

Dinesh Thangavel
Faculty of Engineering
National University of Singapore
Email: a0088592@nus.edu.sg

Xiaoping Ma, Alvin Valera and Hwee-Xian Tan
Sense and Sense-Abilities
Institute for Infocomm Research
Email: xma, acvalera, tanhx@i2r.a-star.edu.sg

Colin Keng-Yan TAN
School of Computing
National University of Singapore
Email: colintan@nus.edu.sg

***Abstract*—Wireless sensor networks (WSNs) typically consist of sensor nodes and gateways that operate on devices with limited resources. As a result, WSNs require bandwidth-efficient and energy-efficient application protocols for data transmission. Message Queue Telemetry Transport (MQTT) and Constrained Application Protocol (CoAP) are two such protocols proposed for resource-constrained devices. In this paper, we design and implement a common middleware that supports MQTT and CoAP and provides a common programming interface. We design the middleware to be extensible to support future protocols. Using the common middleware, we conducted experiments to study the performance of MQTT and CoAP in terms of end-to-end delay and bandwidth consumption. Experimental results reveal that MQTT messages have lower delay than CoAP messages at lower packet loss rates and higher delay than CoAP messages at higher loss rates. Moreover, when the message size is small and the loss rate is equal to or less than 25%, CoAP generates lower additional traffic than MQTT to ensure message reliability.**

I. Introduction

Wireless sensor networks are being deployed for large-scale sensing of environmental parameters such as temperature, humidity and air quality [1] in cities and hostile environments [2]. A typical WSN deployment consists of sensor nodes and gateways. The sensor nodes measure the physical environment and send the data to a gateway node. The gateway aggregates the data from various sensor nodes and then sends the data to a server/broker.

From an end-to-end perspective, a WSN can be viewed as comprising of two subnets: (*i*) a subnet connecting sensor nodes and one or more gateway nodes in which sensor nodes route data until it reaches one of the gateways using WSN protocols (*e.g.*, Collection Tree Protocol [3]), and (*ii*) another subnet connecting the gateway and a back-end server or broker. Sensor data generated by sensor nodes are delivered to the server through the gateway. Meanwhile, clients that are interested to receive sensor data connect to the server to obtain the data. Fig. 1 shows one example of how sensor data flow from sensor nodes to the gateway, then to the server and finally to the clients.

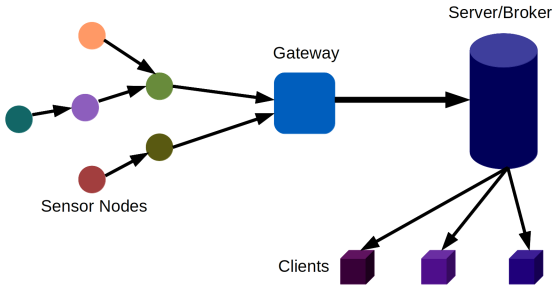

Fig. 1: End-to-end wireless sensor network perspective: from a WSN to the data consumers. The focus of this paper is the process of data transmission for a gateway to the server or broker.

To transfer all the sensor data collected by a gateway node to a server, the former requires a protocol that is bandwidth-efficient, energy-efficient and capable of working with limited hardware resources (*i.e.*, main memory and power supply). As a result, protocols such as Message Queue Telemetry Transport (MQTT) [4] and Constrained Application Protocol (CoAP) [5] have been proposed to specifically address the difficult requirements of real-world WSN deployment scenarios.

One way for wireless sensor networks to transfer data from a gateway to clients is the "publish-subscribe" architecture [6]. In this architecture, a client needing data (known as subscriber) registers its interests with a server (also known as broker). The client producing data (known as publisher) sends the data to a server and this server forwards the fresh data to the subscriber. One of the major advantages of this architecture is the decoupling of the clients needing data and the clients

sending data, *i.e.*, sensor nodes need not know the identities of clients that are interested in their data and conversely, clients need not know the identities of sensor nodes generating the sensor data. This decoupling enables the architecture to be highly scalable [6]. The "publish-subscribe" architecture is supported by machine to machine (M2M) protocols such as MQTT and CoAP.

This paper focuses on the transport of the aggregated sensor data at the gateway node to the back-end server or broker. This paper has two novel contributions: (*i*) We developed a *common middleware* that can accommodate different application protocols based on the "publish-subscribe" architecture for a gateway; and (*ii*) We experimentally evaluated the performance of CoAP and MQTT at different network conditions using the common middleware. One of the major advantages of the common middleware is that it enables the adaptive selection of the most suitable protocol based on network conditions.

The rest of the paper is organized as follows: Section II provides an overview of CoAP and MQTT and introduces the common middleware design. Section III presents the experimental setup used to evaluate the common middleware while section IV provides a discussion of the experimental results. Section V finally concludes the paper.

II. Application Layer Protocols and Common Middleware

A. Message Queue Telemetry Transport (MQTT)

Message Queue Telemetry Transport (MQTT) protocol is an application layer protocol designed for resource-constrained devices [4]. It uses a *topic-based publish-subscribe architecture*. This means that when a client publishes a message M to a particular topic T, then all the clients subscribed to the topic T will receive the message M. Like Hypertext Transfer Protocol (HTTP), MQTT relies on Transmission Control Protocol (TCP) and IP as its underlying layers. However, compared to HTTP, MQTT is designed to have a lower protocol overhead.

The reliability of messages in MQTT is taken care by three Quality of Service (QoS) levels. QoS level 0 means that a message is delivered at most once and no acknowledgement of reception is required. QoS level 1 means that every message is delivered at least once and confirmation of message reception is required. In QoS level 2, a four-way handshake mechanism is used for the delivery of a message exactly once.

B. Constrained Application Protocol (CoAP)

Constrained Application Protocol (CoAP) is a recently developed application layer protocol intended to be used in the communication of resource-constrained devices. This protocol is based on Representational State Transfer (REST) architecture and supports request-response model like HTTP. In addition to request-response model, CoAP also supports publish-subscribe architecture using an extended GET method. Unlike MQTT, the publish-subscribe model of CoAP uses Universal Resource Identifier (URI) instead of topics [5]. This means that subscribers will subscribe to a particular resource indicated by the URI U. When a publisher publishes data D to the URI U, then all the subscribers are notified about the new value as indicated in D.

	MQTT	CoAP
Application Layer	Single Layered completely	Single Layered with 2 conceptual sub layers (Messages Layer and Request Response Layer)
Transport Layer	Runs on TCP	Runs on UDP
Reliability Mechanism	3 Quality of Service levels	Confirmable messages, Non-confirmable messages, Acknowledgements and retransmissions
Supported Architectures	Publish-Subscribe	Request-Response, Resource observe/Publish-Subscribe

Fig. 2: Major differences between MQTT and CoAP

The major difference between CoAP and MQTT is that the former runs on top of the User Datagram Protocol (UDP) while the latter runs on top of TCP. As UDP is inherently not reliable, CoAP provides its own reliability mechanism. This is accomplished with the use of "confirmable messages" and "non-confirmable messages". Confirmable messages require an acknowledgement while non-confirmable messages do not need an acknowledgement. Another difference between CoAP and MQTT is the availability of different QoS levels. MQTT defines 3 QoS levels while CoAP does not provide differentiated QoS. The main differences between CoAP and MQTT have been summarized in the Fig. 2.

C. Common Middleware

CoAP and MQTT are just two of the protocols that are being proposed for gateway-to-server communication and indeed, numerous protocols have been proposed for this purpose. For a gateway node to be highly flexible, it must support the various protocols but at the same time, it must expose a common API to ease the development of gateway applications. We are

therefore proposing a common middleware that has the following features:

- *Extensible*: Support for existing and future gateway-to-server data transport protocols;
- *Common API*: Provides a common programming interface to access the different functionalities of the underlying protocol;
- *Adaptive*: In the future, the middleware can intelligently decide on the protocol to employ given certain constraints.

The current design and implementation of the common middleware is illustrated in Fig. 3. A common interface was developed to enable the gateway to forward data aggregated from sensor nodes to a server using any of the protocols. As mentioned, the middleware is open to extensions such that new protocols can be easily incorporated into the middleware.

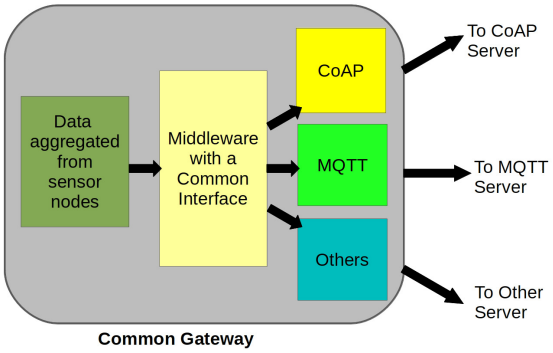

Fig. 3: Design of the Middleware. The middleware provides a common interface so that the processes above the middleware can easily select which protocol to use.

The common interface of the middleware provides API calls such as $publishMessage()$ and $isPublishSuccess()$ to publish messages and to check if the published message has been successfully received and accepted by the broker. Though the individual protocol implementations of MQTT and CoAP are different, the features of the publish-subscribe architecture are used to create the common API calls. For example, the API call $publishMessage()$ takes in arguments such as *message*, *destination* and *topic*. In the case of MQTT, the message is published to the topic by connecting to the broker with IP address given as destination. In the case of CoAP, a URI is first generated using the destination and topic and the message is then published to the URI.

This common middleware implementation has been used to compare the performance of the MQTT and CoAP protocols. The common gateway provides an uniform environment for the comparison of the protocols and simplifies the process of comparison. The exact method adopted for comparing the protocols is described in Section III.

III. Experiment Setup

The middleware presented in section II-C was used to study the performance of the MQTT and CoAP protocols. The aim of the experiment was to identify the influence of various parameters on the performance of the protocols. The performance of the protocols was measured in terms of *delay* and *total data (bytes) transferred per message*. The total data transferred per message is an indicator of bandwidth usage of the protocol. The delay was measured as the difference between the time when a file (which emulates aggregated sensor data) was received and the time when the file was published.

A. Hardware Setup

The hardware setup used in the experimental evaluation consists of a laptop, a BeagleBoard-xM and a netbook to act as the server, publisher, subscriber, respectively, and a Wide Area Network (WAN) emulator as shown in Fig. 4. The common middleware implementation was deployed and executed in the BeagleBoard-xM which was connected to a layer-2 switch through Ethernet. This middleware played the role of a publisher and the messages published by the gateway were routed through the netbook. A Wide Area Network emulator application known as Wanem [7] was run on the netbook to emulate a lossy network connection as in the previous work done by Bhattacharyya and Bandyopadhyay [8].

The MQTT server and CoAP server were run in the laptop with 4GB RAM and this laptop was also connected to the layer-2 switch as shown in the Fig. 4.

In order to ensure time synchronization for calculating delay, the subscriber program also ran in the BeagleBoard-xM. Thus, any message published by the middleware in the BeagleBoard-xM would go through the netbook to reach the server in the laptop and then return back to the BeagleBoard-xM via the netbook.

B. Software Setup

Open source implementations of MQTT and CoAP protocol namely Mosquitto [9] and libcoap [10] were integrated with the middleware and used for conducting the experiments. To emulate the reception of data from sensor nodes, a small program was executed in the BeagleBoard-xM to generate sensor data, which was in turn published by the gateway.

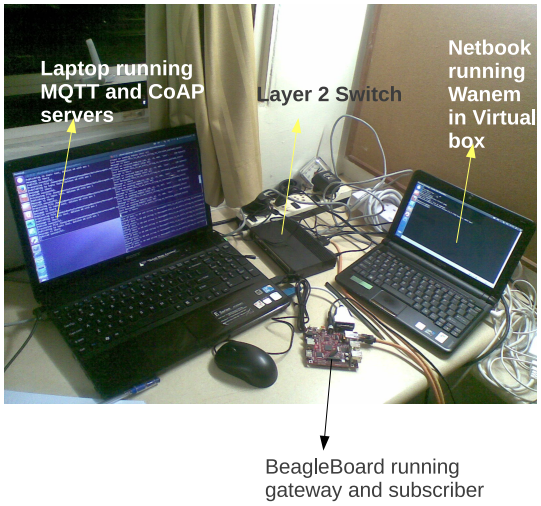

Fig. 4: Setup of the Experiment. The laptop acts as the server, the BeagleBoard-xM hosts both the publisher (the gateway program) and subscriber to ensure time synchronization, and the netbook runs WAN emulator to control the packet loss rate.

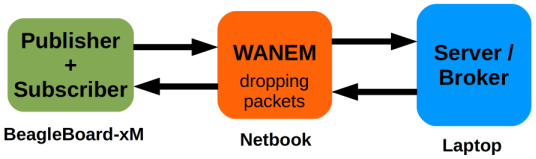

Fig. 5: Data flow in the experiment. The messages published by the publisher go through the WAN emulator to reach the Server. The server then sends the messages back to the subscriber through the WAN emulator.

In order to calculate the delays accurately, every message published by the gateway and received by the subscriber were recorded down. Wireshark [11] was used to measure the number of bytes transferred through the network throughout the experiments. Note that the total number of transferred bytes included protocol overheads as well as retransmissions. The results obtained from the experiments are discussed in Section IV.

IV. Experimental Results

As the simple experimental setup involved only one publisher, one server and one broker, both protocols achieved 100% message delivery ratio regardless of the packet loss rate in all of experiments carried out.

TABLE I: Average Delay for different Loss Rates for both MQTT and CoAP.

	Loss rate 0%	Loss rate 5%	Loss rate 10%	Loss rate 15%	Loss rate 20%	Loss rate 25%
MQTT (QoS1 – QoS1)	0.005289s	0.022609s	0.144252s	0.450860s	0.871025s	10.33348s
CoAP	0.008847s	0.552006s	0.642688s	1.026007s	1.886399s	2.824115s

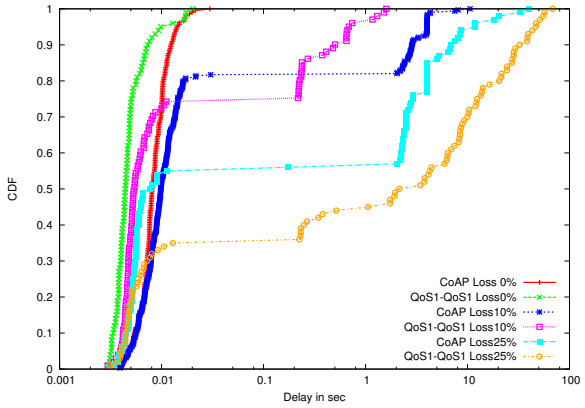

Fig. 6: Influence of Packet Loss Rates On Delay CDFs

This reflects that both MQTT and CoAP have good retransmission schemes to handle the packet loss in the lower layer. We therefore concentrate our comparison on the message delay and the total amount of data generated for every message.

A. *Influence of Packet Loss on Delay*

We first consider the impact of packet loss on the delay performance of the two protocols. The packet loss in any network leads to retransmission of messages and thus leading to longer delays in the message reception. For fair comparison, MQTT gateway QoS 1 and subscriber QoS 1 were selected to compare with confirmable messages of CoAP as these two settings are similar in terms of the transmission and acknowledgement of a message.

Table I enumerates the average delay of the two protocols under different packet loss rates while Fig. 6 plots the delay cumulative distribution function (CDF). It is evident that the messages experienced lower delay in MQTT for lower values of packet loss. However, as the packet loss increased, CoAP performed better than MQTT in terms of delay. This is because of the greater TCP overheads involved in the retransmission of messages compared to the smaller UDP overheads in CoAP when the packet loss rate is high.

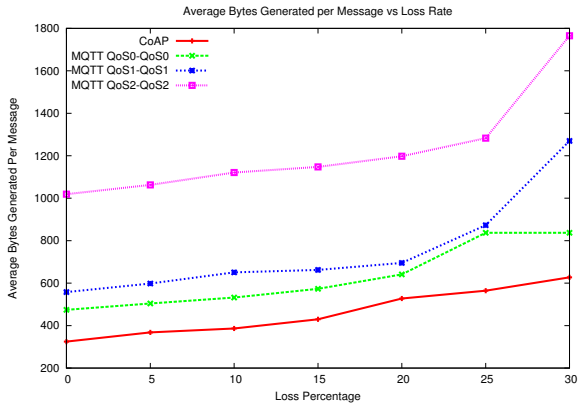

Fig. 7: Average Data Transferred Per Message vs Packet Loss Rate

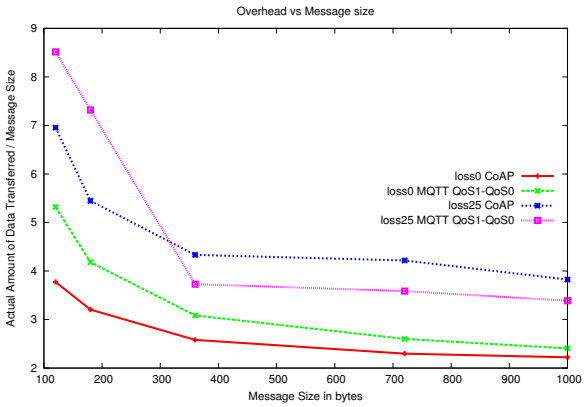

Fig. 8: Ratio of the Average Data Transferred to the Message Size vs Message Size under Different Packet Loss Rates.

B. Influence of Packet Loss on Data Transfer

The data transferred for every message was defined as the total amount of traffic generated divided by the total number of the successfully delivered messages. It is calculated from Wireshark for message size of 60 bytes at various values of loss rates. The quality of the service of the gateway and subscriber were made the same. The graph in Fig. 7 summarized the results obtained. The graph showed that QoS 2 messages occupied more bandwidth when compared to QoS 0 and QoS 1 for lower values of packet loss. This is because of the four-way handshake mechanism in QoS 2. QoS 0 has the least data transfer since no acknowledgement from the other host is needed.

C. Overhead For Various Message Sizes

The ratio of the total data transferred to the total message size is an indicator of the overheads involved in the data transfer. This ratio for various payload sizes is shown in Fig. 8. The ratio for smaller messages is large because the size of acknowledgements is comparable with that of the message. When the message size grows, the size of the acknowledgement has little effect on the extra overhead caused. The extra overhead is determined primarily by the message size and the number of retransmissions.

When the packet loss rate is low, CoAP generated less overhead than MQTT for all message sizes. When the packet loss rate is higher, CoAP still enjoyed a less overhead than MQTT when message size is small. When the message size grows, the reverse is true. This is because when the message size is large, the probability that UDP loses the message is higher than TCP, which causes CoAP to retransmit the whole message more often than MQTT.

D. Adaptively Changing Protocols

Sections IV-A, IV-B and IV-C showed that performance of both protocols differ at different network conditions. Using the right protocol for different network conditions will likely improve the performance of the network in different aspects, such as the average message delay or the total amount of data generated per delivered message. In addition to the network conditions, applications can also decide the choice for protocols as different applications have different requirements such as low latency or low bandwidth utilization.

To enable this feature, additional requirements for the gateway and the subscriber are needed. The gateway has to be able to detect the network conditions and must have prior knowledge of the performance of a protocol under different network conditions. Depending on the performance metric to be optimized, the gateway will then choose the best protocol to be used in the sensor data delivery. The subscriber has to be able to receive messages from different protocols. As a results, it is better for the subscriber to have a middleware to handle the complexity. The same middleware we developed can be extended to meet the requirements for the subscriber.

V. Conclusion and Future Work

In this paper, we studied the transport of the aggregated sensor data from the gateway node to the back-end server or broker. We proposed a common middleware that is flexible, exposes a common programming interface, and can be extended to be adaptive to network conditions. Using the common middleware, we studied the performance of MQTT and CoAP, two of the most commonly-used protocols for gateway-to-backend data

transport. Experimental results showed that the performance of different protocols are dependent on different network conditions. MQTT messages experienced lower delays than CoAP for lower packet loss and higher delays than CoAP for higher packet loss. Moreover, when the message size is small and the loss rate is equal to or less than 25%, CoAP generates less extra traffic than MQTT to ensure reliable transmission.

This difference in the performance can be exploited to improve the network performance with the help of the middleware that can decide which protocol to use based on the prevailing network conditions. Future areas of research include the detection of network conditions at the gateway and then switching to the protocol that gives maximum performance for the network conditions.

REFERENCES

[1] R. Musloiu-e, A. Terzis, K. Szlavecz, A. Szalay, J. Cogan, and J. Gray, "Life under your feet: A wireless soil ecology sensor network," 2006.

[2] K. Martinez, R. Ong, and J. K. Hart, "Glacsweb: a sensor network for hostile environments." in *SECON*. IEEE, Mar 2004, pp. 81–87. [Online]. Available: http://dblp.uni-trier.de/db/conf/secon/secon2004.html#MartinezOH04

[3] O. Gnawali, R. Fonseca, K. Jamieson, D. Moss, and P. Levis, "Collection tree protocol," in *Proc. ACM SenSys*, 2009, pp. 1–14.

[4] MQTT.org, "Mq telemetry transport," http://mqtt.org/, 2013.

[5] Z. Shelby, Sensinode, K. Hartke, C. Bormann, and U. B. TZI, "Constrained application protocol (coap) draft-ietf-core-coap-17," http://tools.ietf.org/html/draft-ietf-core-coap-17, 2013.

[6] P. T. Eugster, P. A. Felber, R. Guerraoui, and A.-M. Kermarrec, "The many faces of publish/subscribe," *ACM Comput. Surv.*, vol. 35, no. 2, pp. 114–131, Jun. 2003. [Online]. Available: http://doi.acm.org/10.1145/857076.857078

[7] T. C. S. L. TCS, "Wanem the wide area network emulator," http://wanem.sourceforge.net/, 2013.

[8] A. Bhattacharyya and S. Bandyopadhyay, "Lightweight internet protocols for web enablement of sensors using constrained gateway devices," *2013 International Conference on Computing, Networking and Communications (ICNC)*, vol. 0, pp. 334–340, Jan 2013.

[9] R. Light, "Mosquitto-an open source mqtt v3.1 broker," http://mosquitto.org/, 2013.

[10] Obgm, "libcoap: C-implementation of coap," http://sourceforge.net/projects/libcoap/, 2013.

[11] Wireshark.org, "Wireshark," http://www.wireshark.org/, 2013.