CoAP vs. MQTT-SN: Comparison and Performance Evaluation in Publish-Subscribe Environments

Fabio Palmese, Edoardo Longo, Alessandro E. C. Redondi, Matteo Cesana
DEIB, Politecnico di Milano
Milano, Italy
Email: {name.surname}@polimi.it

***Abstract*—The Publish/Subscribe communication pattern has proved to be particularly tailored to the IoT world, with the MQTT protocol being the nowadays standard de-facto for IoT applications. Request/response protocols explicitly designed for the IoT, such as CoAP, have been revised to support also Publish/Subscribe. The purpose of this paper is to perform a comparison between two protocols: MQTT-SN, the version of MQTT thought specifically for sensor networks, and CoAP in its Pub/Sub version, defined in a recent IETF draft. Both protocols are Pub/Sub in nature and based on UDP at the transport layer, allowing therefore a fair comparison of their functionalities. We propose a open-source implementation of the CoAP Pub/Sub version and we compare the two protocols: first from a theoretical perspective and, then, in a simulated environment characterized by varying number of clients and network conditions. Results show that CoAP represents a valid alternative to MQTT-SN for publish-subscribe environments; in particular, CoAP results being the best choice for highly dynamic networks.**

***Index Terms*—CoAP, MQTT-SN, Publish-Subscribe, application protocols, Internet of Things**

I. Introduction

According to the latest Cisco's Annual Internet Report [1], the number of IoT and Machine-to-Machine (M2M) connections will account for half of the global number connections by 2023, with more than 14 billion IoT devices connected and generating about 79.4 zettabytes ($10^{21} \simeq 2^{70}$ Bytes) of applicationtraffic.

Managing such a massive amount of traffic requires to develop specific communication protocols that can adequately cope with the resource-constrained scenarios typical of the IoT world, where the available network bandwidth, processing power and energy are limited. Indeed, in the last few years, many protocols spanning the whole network stack have been proposed from both the industry and the academia, generating a fragmented and sometimes confusing plethora of solutions and standards. Leaving aside the specific communication technology adopted at the bottom layer and focusing only on the application layer, the IoT is nowadays characterized by a growing dichotomy between Representational State Transfer (REST) and Publish/Subscribe. The former approach, based on a Request-Response interaction, allows a one-to-one communication between clients and servers. The most famous example is undoubtedly the Constrained Application Protocol (CoAP), directly derived from HTTP and standardized by IETF. Conversely, the Publish/Subscribe pattern allows for many-to-many communication through a central broker connecting client devices. The Message Queue Telemetry Transport protocol (MQTT), is surely the most well-known example of this kind: due to its extreme simplicity and renovated popularity, MQTT is practically becoming the standard de-facto for M2M and IoT applications.

Both MQTT and CoAP have been subject to a series of improvement steps in the last few years with the objective of making the two protocols even more tailored to IoT scenarios' peculiarities. A lightweight and UDP-based version of MQTT named MQTT-SN [12] has been proposed to target the limitations typical of the world of Wireless Sensor Networks. At the same time, the IETF has recently issued a draft describing publish-subscribe functionalities for CoAP [9], implicitly acknowledging the benefits of such a communication approach in several IoT scenarios.

With this work, we aim to compare MQTT-SN with the Pub/Sub version of CoAP. Our contribution is twofold: first, we provide a working implementation of the Pub/Sub functionalities for CoAP described in the IETF draft. Then, we compare it with MQTT-SN from both a theoretical and a practical perspective, focusing on the traffic characteristics. Since both protocols are Pub/Sub in nature and supported by UDP at the transport layer, we believe such a comparison is fairer than other works in the literature setting side by side the legacy versions of MQTT and CoAP (e.g., [2] [10] [14]).

This paper's remainder is organized as follows: the main related works are described in Section II. Section III provides a brief background on MQTT-SN and CoAP Pub/Sub. Section IV provides a general comparison of CoAP Pub/Sub and MQTT-SN characteristics, whose performance are analyzed through the experiments in Section V. Finally Section VI contains concluding remarks and future research directions.

II. Related Work

In September 2015, the IETF released RFC 7641 [4], introducing the *observe* mode. Observing resources in CoAP allows the protocol to work in a publish-subscribe fashion in which both publisher/subscriber are CoAP clients, and the server takes the role of a central broker allowing a many-to-many communication. The RFC introduces a new option (Observe) to transform a simple GET request into a Subscribe/Unsubscribe request. Since the release of the extension, several works focused on the Pub/Sub capabilities of the protocol, sometimes proposing innovative features.

In [6], the authors propose an improvement of the publish-subscribe model for CoAP by introducing new Options and new response codes for faster communication. In particular, the work focuses on aggregating functionalities (e.g., creating a topic, subscribing or publishing to it) with a single request to minimize traffic. The work describes the results obtained theoretically, and leaves as future work its implementation for practical results.

The work in [7] describes a distributed CoAP Pub/Sub protocol, where multiple connected brokers substitute the central broker. The authors accurately discuss all the details of the distributed scenario, ranging from joining procedures for the brokers to load-balancing techniques to distribute topics among the different brokers. Results have been obtained through an implementation relying on Java Californium and highlighting a higher time needed for the subscription phase but a lower end-to-end delay than MQTT and centralized CoAP implementations. Several new features are introduced for the CoAP protocol in many of its aspects in [8] in which CoAP 2.0 is presented. Among the improvements, authors introduce new features for the publish-subscribe model by adding new rules to subscriptions. A Rule option is added in the subscribe request so as to receive from the server only those notifications matching that rule (e.g., values greater or lower than a particular threshold).

For completeness, we also briefly discuss the main related works focusing on comparing CoAP (in both its Request/Response or Pub/Sub versions) to MQTT or MQTT-SN. In [5] the authors make use of a publish-subscribe version of CoAP (implemented by using the observe extension but without respecting the requirements specified in the IETF draft [9]) for Fog Computing applications, comparing its performance with MQTT for the transfer of XML files. The authors of [14] used CoAP with the observe extension to compare the behaviour of CoAP in a publish-subscribe model with MQTT, using one publisher and one subscriber. In particular, the work compares the behaviour in terms of traffic generated and average delay of the two implementations, using *libcoap* for CoAP and Mosquitto for MQTT, changing the network packet loss rate through a network emulator (WANEM).

In [2], a comparison between CoAP and MQTT-SN is performed in the use of robotic applications. CoAP is used in its simple request-response model by sending a POST request and waiting for the response. MQTT-SN is analyzed by performing a publish request and waiting for the publish ack, starting from a disconnected client and calculating the needed time to publish completion. The authors highlight that the delay introduced in the connecting and the registering phases makes MQTT-SN slower than CoAP for the first message transmission. However, they obtain better results in the average transmission thanks to the smaller packet size.
Finally, the work in [10] compares MQTT-SN and CoAP in its request-response paradigm, focusing mainly on energy consumption. Results obtained with a simulator reveal that the two protocols act almost in the same way in terms of consumed energy, with slightly better performance for MQTT-SN.
To the best of our knowledge, a comparison of CoAP and MQTT-SN in Publish-Subscribe environments has not been previously explored in depth. Hence, the scope of this work is to compare the traffic behaviour of the two protocols in such cases in order to highlight the advantages of using one with respect to the other.

III. Pub/Sub protocols for IoT

A. MQTT and MQTT-SN

MQTT is a publish/subscribe communication protocol where all the communications between nodes are made available via a broker. The broker accepts messages published by devices on specific topics and forwards the messages to the clients subscribed to those topics, ultimately controlling all aspects of communication between devices. In MQTT, publishers and subscribers are connected to the broker via TCP, a transport protocol known to be resource-eager and therefore generally avoided in resource-constrained applications. To overcome this issue, MQTT-SN adapts the functionalities of the MQTT protocol to resource limited application scenarios such as Sensor Networks [12]. In particular, MQTT-SN operates over UDP and is based on an MQTT-SN Gateway which interconnects MQTT-SN clients to a legacy MQTT broker. Also, MQTT-SN introduces several features like the possibility of registering a topic through a 2-bytes topic identifier to decrease the size of published packets drastically. MQTT-SN maintains the same Quality-of-Service available in legacy MQTT, with the addition of a quick publish (QoS = -1) which does not require any connection to the broker or gateway.

B. CoAP and CoAP Pub/Sub

In contrast to MQTT, CoAP is a Request/Response application protocol developed by the IETF for constrained networks. The protocol is standardized in RFC 7252 by IETF [11]. CoAP functionalities are directly derived from HTTP, allowing a request-response communication in which two nodes can exchange messages in a client-server interaction. The protocol maintains the same structure for requests and responses but, unlike HTTP, it runs over UDP. Four request methods (GET, POST, PUT, DELETE) and HTTP-like response codes are defined to allow flexible M2M interactions. Quality of Service is managed at the application layer by using two types of CoAP messages: Confirmable and Non-Confirmable.

The release of the Observe extension made the protocol adaptable for Publish/Subscribe environments. By sending a GET request with Observe = 0, a client can express interest in a resource to receive a notification every time such a resource changes its value; to cancel the subscription a client can send a GET request with Observe = 1. In [9], the IETF draft standardizes both the CoAP broker and publishers/subscribers' behaviour to achieve correct communications in a publish-subscribe fashion. In a nutshell, a CoAP client may SUBSCRIBE (UNSUBSCRIBE) on a topic by issuing a CoAP GET request with Observe = 0 (1). The topic, in this case,

TABLE I: CoAP/MQTT-SN KEY DIFFERENCES

	CoAP Pub/Sub	**MQTT-SN**
Connection management	Connection-less	Connection-Oriented
Topic structure	Resource (Uri-Path)	Topic (Topic-Id)
Quality of Service	CON/NON type	4 QoS levels
Fragmentation	Block-Wise Transfer	Not supported

is identified by a resource object with a specific Uri-Path. Similarly, a CoAP client may PUBLISH on a topic by issuing a CoAP PUT or POST request. The draft also specifies other request methods: a client can discover what topics are available on a broker through the DISCOVERY request, and topics can be created/deleted using the CREATE/REMOVE requests.

In order to analyze the capabilities of such version of CoAP and compare them with MQTT-SN, we implemented a working version which is strictly compliant to the IETF draft specifications, providing a set of API for easily generating CoAP Pub/Sub traffic for the DISCOVERY, READ, PUBLISH, SUBSCRIBE, UNSUBSCRIBE, CREATE, and REMOVE requests. We based our implementation on the CoAPthon library [13], written in Python, which already implements the main features of CoAP. The source code of our implementation is publicly available for reproducible research[1].

IV. CoAP Pub/Sub vs MQTT-SN

In this Section, we dissect and analyze the two protocols to explain their main analogies and differences, which are also summarized in Table I. On the one hand, we recall that both protocols are Pub/Sub in nature and based on UDP at the transport layer. On the other hand, several differences can be observed:

a) Connection management: One of the main differences between the two protocols is the connection management. Apart from the case when QoS = -1, which corresponds to a publish-only fire and forget mode, an MQTT-SN client needs to maintain an active connection with the broker (through the CONNECT/CONNACK message exchange) in order to publish, subscribe or register a topic. Conversely, a CoAP client can perform any action on the broker without establishing a connection. A CoAP client can hence access to all broker functionalities in a faster way than MQTT-SN.

b) Resource-based vs Topic-based Environments: A key difference between CoAP and MQTT-SN is the representation of resources and topics. CoAP uses a resource-oriented model: a broker contains resource objects (identified by URIs), and a client may subscribe or publish to a specific resource present in the broker. Resources are created or deleted by clients using CREATE or REMOVE requests. A client is able to discover the available resources of a broker through a DISCOVERY request. MQTT-SN uses instead a topic-oriented environment: the broker stores only the interests of clients to particular topics, and, whenever a new message is published, the broker forwards it to all the interested clients. This approach allows an MQTT-SN client to subscribe to a topic not yet referenced (created) by other clients, while this is not possible in CoAP. In addition, MQTT-SN offers the *retain* option to store a published message on the broker for clients interested in a topic but not yet subscribed.

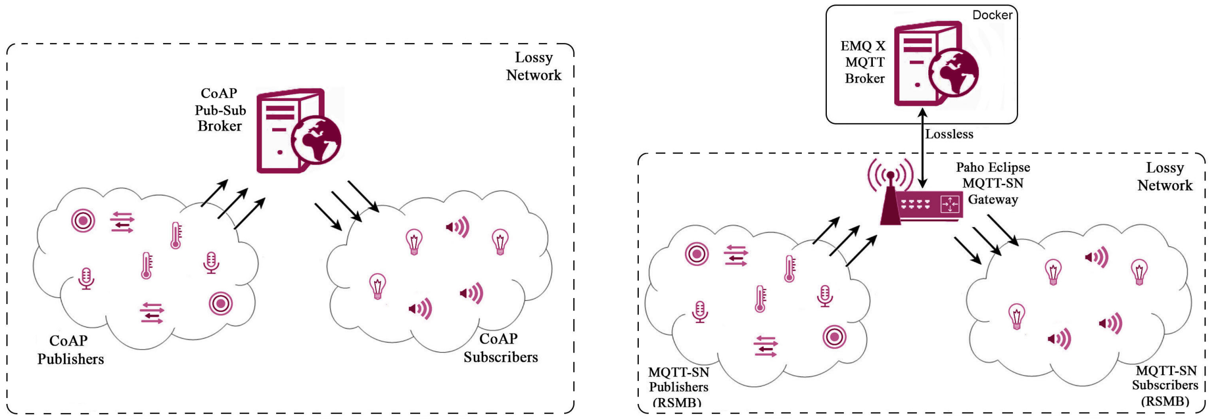
The use of topics also impacts the publish message's size: in MQTT-SN, a topic can be registered and encoded with a 2-byte topic identifier used instead of the original topic name. This allows reducing the message size significantly, at the cost of a registration step to be performed a priori. CoAP does not allow such an option.

c) Notifications: Regardless of the specific protocol, at the reception of the publish message, the broker needs to notify all the interested subscribers of the newly published value for that topic/resource. An MQTT-SN broker notifies the subscribers simply forwarding the received publish message, possibly modifying only header fields such as the message-id and the QoS. In CoAP, instead, the broker notifies the subscribers by sending a notification that represents a response to a previous subscribe request. Such notification is characterized by a response code which needs to have the same token as the subscribe request. Moreover, a CoAP broker can sort notifications in order of arrival through the Observe option, allowing the subscribers to understand which one is the freshest, a feature not achievable in MQTT-SN.

d) QoS management: Since both protocols run over UDP, quality of service must be implemented at the application layer. CoAP specifies two types of messages: Confirmable (i.e., requiring an acknowledgement) and Non-Confirmable. Note that both Requests and Responses can be of Confirmable type, thus requiring an acknowledgement independent of the actual Request/Response exchange. MQTT-SN instead specifies four levels of QoS, which can be independently chosen by publishers and subscribers. The first two QoS levels (-1 and 0) do not require any acknowledgement, with QoS = -1 being a publish-only fire and forget mode not requiring a connection. These two QoS levels have the same effect of using a CoAP Non-Confirmable request message with the No-Response Option set to force the endpoint not to reply. QoS level 1 refers to "at least once" delivery, where an acknowledgement is requested from the broker or the subscriber receiving the message. Compared to CoAP Confirmable message, this QoS mode produce effectively the same number of messages to perform a publish operation. There is a difference among the two models caused by the different method used to handle duplicate messages: CoAP Confirmable ensures an "exactly once" delivery while MQTT-SN with QoS 1 ensures an "at least once" delivery, not removing the presence of duplicate notifications. To ensure an "exactly once" delivery in MQTT-SN, a publisher should use QoS = 2, which is based on a four-way handshake and therefore involves higher traffic and delay.

The last difference regards the subscribers QoS: while in MQTT-SN each subscriber can choose the QoS for the notifications to receive from the broker, CoAP does not allow that and leaves the decision to the broker.

[1]https://github.com/fpalmese/CoAP-Pub-Sub



(a) CoAP **(b) MQTT-SN**

Fig. 1: Sketch of the testing architectures.

e) Fragmentation: Although the size of IoT messages is generally small, some applications (e.g., IoT surveillance cameras) may produce payloads whose size exceed the maximum allowed by the network, requiring to fragment the original payload in multiple parts. CoAP provides message fragmentation at the application layer with the Block option (RFC 7959 [3]), which allows to request acknowledgements for each fragment separately. Conversely, MQTT-SN does not allow message fragmentation; therefore, the maximum message length is limited by the underlying network.

V. Experiments

A. Testing setup

To compare the two protocols from a practical perspective, we set up a testing environment on a single machine based on an Intel i7-8750 with 6 CPUs @ 2.2GHz, 16 GB of RAM, running Ubuntu 18.04.4 and Python 3.6.9 Version. For what concerns CoAP Pub/Sub we use our proposed implementation for both the broker and the clients (publishers and subscribers). As for MQTT-SN, we use a standard MQTT broker based on EMQX[2] and executed within a docker. The MQTT broker is connected to an MQTT-SN gateway derived from the Paho-Eclipse projects[3]. Finally, MQTT-SN clients are based on Really Small Message Broker (RSMB) project[4]. In both cases, publishers and subscribers are connected to the broker on the loopback interface via a lossy network, which is emulated through the use of the Linux `traffic control (tc)` tool[5].

A sketch of the architecture used for the two protocols is reported in Figure 1.

For each scenario the tests are executed as follows. First, the broker is started, then:

1) A specific number of subscribers is started, each one subscribing to the same topic on the broker. We vary the number of subscribers in the range {1, 10, 100}. For both the protocols, the topic used by clients is `ps/topic/`, equal to 9 bytes of overhead in the request header. Only for MQTT-SN, we assume the topic is already registered in the broker and linked to a 2-bytes short topic id. In both scenarios, the broker notifies the subscribers at the lowest QoS available (with NON messages in CoAP, and with QoS 0 in MQTT-SN).
2) A specific number of publishers is started, publishing messages on the same topics mentioned above every 0.25s (4 msg/s). We vary the number of publishers in the range {1, 10, 50}. In order to increase the number of notifications to the subscribers leaving unchanged the number of requests, we kept the total volume of traffic published on the broker constant and equal to 50 messages (which is respectively 50, 5, 1 messages for each 1, 10, 50 publishers).
   A fixed payload of 18 bytes composes each published message. Table II reports the size of all messages used in the simulation.
3) Finally, to analyze the two protocols' behaviour at different QoS levels, we perform different tests comparing CoAP non-confirmable publish messages (with No-Response option) with MQTT-SN publish at QoS -1 and 0, as well as CoAP confirmable publish messages with MQTT-SN publish at QoS 1. All tests are repeated varying the packet loss rate of the loopback interface, through the `tc` tool. In particular, the following values are used for the packet loss rate: 0% (reliable network), 5%, 10%, 20% and 30%.

TABLE II: CoAP and MQTT-SN messages size (bytes)

CoAP		MQTT-SN	
Subscribe Request	60	Subscribe Request	58
Subscribe Response	55	Subscribe Ack	52
Publish Request	77	Publish Request	68
Publish Response	58	Publish Ack	51
Notification	71	Connect Request	54
		Connect Ack	47
		Register Request	59
		Register Ack	51

[2]https://www.emqx.io/
[3]https://github.com/eclipse/paho.mqtt-sn.embedded-c
[4]https://github.com/eclipse/mosquitto.rsmb
[5]https://man7.org/linux/man-pages/man8/tc.8.html

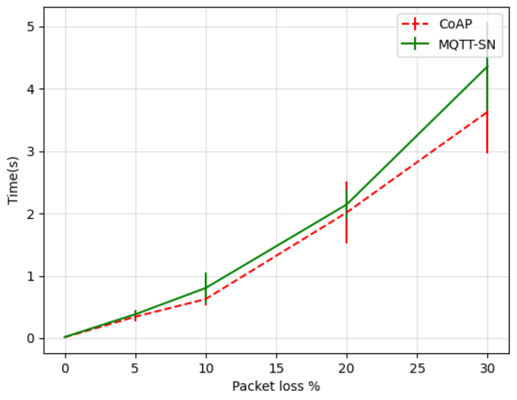


Fig. 2: Average time needed to perform a subscribe request.

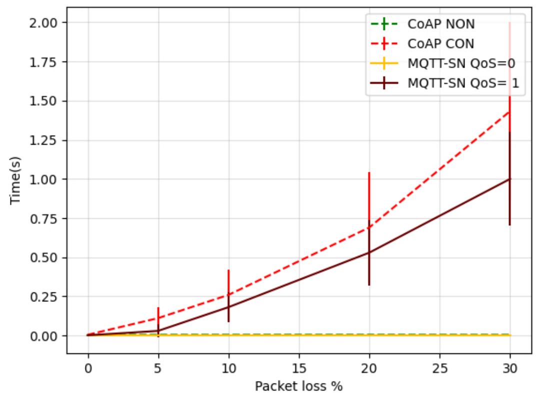


Fig. 3: Average end-to-end delay with 1 Subscriber.

Each test is repeated 10 times in order to analyze average behaviors and the corresponding deviations.

B. Experimental Results

1) Subscribe time: First, we analyze the behaviour of the two protocols for what concerns the time elapsing between a subscribe request from a client and the reception of the corresponding response from the broker (i.e., a SUBACK message for MQTT-SN or a standard response in case of CoAP Pub/Sub). Results are shown in Figure 2, for the case with 100 subscribers[6]. As one can see the two protocols have a similar behaviour with a slightly better result for CoAP, showing a lower processing time than MQTT-SN for the subscription phase.

2) End-to-End delay: We also analyze the end-to-end delay, that is the average time elapsed from the generation of the publish request to the arrival of the notification at the subscriber side. Therefore, such delay includes (i) the publisher's transmission time to the broker (including any retransmission due to lost packets), (ii) the broker's processing time, and (iii) the broker-to-subscriber notification time. To perform a fair comparison, we set the retransmission timeout of MQTT-SN to the one suggested in the CoAP standard. Figure 3 and 4 show the results obtained for the two cases with 1 publisher and 1 subscriber and 50 publishers and 100 subscribers, respectively. As one can see, here, the comparison is in favour of MQTT-SN. The main reason behind this lies in the way MQTT-SN handles notifications compared to CoAP. As explained in Section IV (Paragraph c) in MQTT-SN, the broker just forwards the received publish message to the list of subscribers, while in CoAP the broker needs to create and transmit a new specific response for each subscriber. Overall, for low packet loss rates (e.g., less than 10%) the two protocols behave very similarly. In both cases, the effect of increasing the number of subscribers from 1 to 100 is a very moderate increase in the average end-to-end delay.

3) Traffic volume: The total volume of traffic exchanged during the simulation directly impacts on the network and the energy resources. Figure 5 shows the percentage variations of the different scenarios compared to the traffic volume produced by CoAP in non-confirmable mode for a different number of subscribers (we recall the number of messages published to the broker is constant). Only in this case, we performed the test in a lossless network to avoid the effect of retransmissions. As one can see, for multipoint-to-point communications (e.g., when only one subscriber is involved), MQTT-SN produces around 5% less traffic than CoAP for both the acknowledged and non acknowledged cases, thanks to the smaller packet size (especially regarding the publish requests), balancing the additional traffic introduced by the connection and topic registration. Instead, for multipoint-to-multipoint scenarios (e.g., when the number of subscribers is greater than one), CoAP always produces less traffic than MQTT-SN. This is due to the increased number of connection/registration phases present in MQTT-SN, which is no more leveraged by the smaller packet size of publish requests (since the number of publish requests is left unchanged).

4) Number of notifications: Finally, Figure 6 highlights the difference of the number of notifications transmitted by the broker to the subscribers when using the acknowledged versions of the protocols (CoAP confirmable and MQTT-SN with Qos = 1) with different packet loss rates. As one can see, the number of notifications produced by the MQTT-

[6]no significant changes were observed with a lower number of subscribers

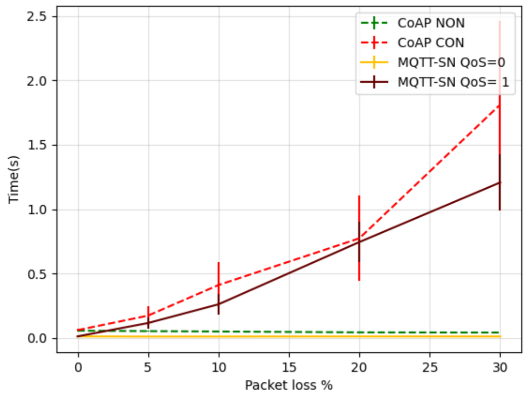


Fig. 4: Average end-to-end delay with 100 Subscribers.

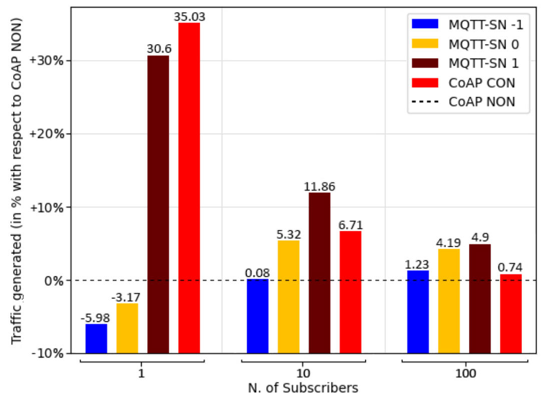


Fig. 5: Percentage of traffic compared to CoAP NON

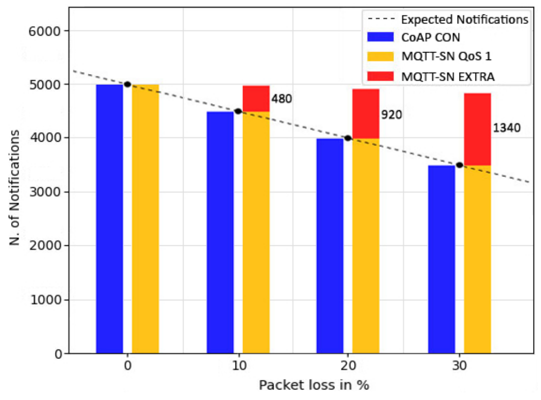


Fig. 6: Total notifications in the simulation with 100 Subs

SN is much higher. This is due to the loss of publication acknowledgements that trigger publishers to retransmit their messages, which are forwarded by the broker as duplicated notifications.

C. Discussion

From the analyses of the experimental results, it is clear that CoAP Pub/Sub and MQTT-SN behave similarly. At the same time, the small differences between the two protocols can have a non-negligible impact in specific scenarios.
For applications in which clients are not subject to duty cycle (i.e., stay always connected with the broker) and transmit periodic messages (e.g., environmental monitoring), MQTT-SN is to be preferred over CoAP due to its smaller packet size that balances out the connection and topic registration overhead. Conversely, if clients connect sporadically or with high dynamicity (e.g., asynchronous events), CoAP Pub/Sub would be the best choice thanks to its connection-less communication model.
Applications characterized by large payload and running over high loss networks should always use CoAP over MQTT-SN, due to the inherent support to application layer fragmentation and thus avoiding the messages to be dropped for the exceeding number of retransmissions.

VI. Conclusions

This work focuses on the publish-subscribe model for CoAP, comparing its behaviour with the already popular MQTT-SN protocol. First, we proposed a publicly available Python implementation of the CoAP Pub-Sub model described in a recent IETF draft. Then, we compared CoAP and MQTT-SN from both a theoretical and practical perspective, putting in evidence the advantages and disadvantages of using one with respect to the other. As future research directions, we plan to enhance CoAP Pub/Sub introducing in our implementation the features theoretically described in [6] and [8] as well as integrating the MQTT protocol in the implementation built in this work to obtain a broker able to process both MQTT/MQTT-SN and CoAP Publish-Subscribe requests.

References

[1] Cisco Annual Internet Report (2018–2023), March 2020.
[2] Muhammad Harith Amaran, Nazmin Arif Mohd Noh, Mohd Saufy Rohmad, and Habibah Hashim. A Comparison of Lightweight Communication Protocols in Robotic Applications. *Procedia Computer Science*, 76:400 – 405, 2015.
[3] C. Bormann and Z. Shelby. Block-Wise Transfers in the Constrained Application Protocol (CoAP). RFC 7959, IETF, August 2016.
[4] K. Hartke. Observing Resources in the Constrained Application Protocol (CoAP). RFC 7641, IETF, September 2015.
[5] J. Huang, P. Tsai, and I. Liao. Implementing publish/subscribe pattern for CoAP in fog computing environment. In *2017 8th IEEE Annual Information Technology, Electronics and Mobile Communication Conference (IEMCON)*, pages 175–180, 2017.
[6] M. Iglesias-Urkia, D. Casado-Mansilla, S. Mayer, and A. Urbieta. Enhanced Publish/Subscribe in CoAP: Describing Advanced Subscription Mechanisms for the Observe Extension. In *Proceedings of the 8th International Conference on the Internet of Things*, IOT '18, New York, NY, USA, 2018. Association for Computing Machinery.
[7] J. Jung, D. Choi, and S. Koh. Distributed pub/sub model in CoAP-based Internet-of-Things networks. In *2018 International Conference on Information Networking (ICOIN)*, pages 657–662, 2018.
[8] M. L. Kome, F. Cuppens, N. Cuppens-Boulahia, and V. Frey. CoAP Enhancement for a Better IoT Centric Protocol: CoAP 2.0. In *2018 Fifth International Conference on Internet of Things: Systems, Management and Security*, pages 139–146, 2018.
[9] M. Koster, A. Keranen, and J. Jimenez. Publish-Subscribe Broker for the Constrained Application Protocol (CoAP). I-D 9, IETF, September 2019.
[10] Marti, Garcia-Rubio, and Campo. Performance Evaluation of CoAP and MQTT-SN in an IoT Environment. *Proceedings*, 31:49, 11 2019.
[11] Z. Shelby, K. Hartke, and C. Bormann. The Constrained Application Protocol (CoAP). RFC 7252, IETF, June 2014.
[12] A. Stanford-Clark and H. L. Truong. MQTT-SN Version 1.2. Protocol Specification, Oasis, November 2013.
[13] G. Tanganelli, C. Vallati, and E. Mingozzi. CoAPthon: Easy development of CoAP-based IoT applications with Python. In *2015 IEEE 2nd World Forum on Internet of Things (WF-IoT)*, pages 63–68, 2015.
[14] D. Thangavel, X. Ma, A. Valera, H. Tan, and C. K. Tan. Performance evaluation of MQTT and CoAP via a common middleware. In *2014 IEEE Ninth International Conference on Intelligent Sensors, Sensor Networks and Information Processing (ISSNIP)*, pages 1–6, 2014.