From IoT to Cloud: Applications and Performance of the MQTT Protocol

From IoT to Cloud: Applications and Performance of the MQTT Protocol

Invited paper

Davide Borsatti[1], Walter Cerroni[1], Federico Tonini[2], Carla Raffaelli[1]
[1] *DEI, University of Bologna, Viale Risorgimento 2, 40136 Bologna, Italy*
[2] *Chalmers University of Technology, SE-412 96 Gothenburg, Sweden*
Email: {davide.borsatti, walter.cerroni, carla.raffaelli}@unibo.it, tonini@chalmers.se

ABSTRACT

A study of the MQTT publish/subscribe protocol with different QoS levels is presented with the aim to extend the Internet of Things (IoT) concept across access, edge and transport networks and reach cloud computing facilities. A simple testbed is set up with related software components to measure the end-to-end delivery latency between the publisher and the subscribers and the impact of the network delay caused by network configurations with different service deployments. In particular, the latency is shown to rise up to more than 7 times the average network delay when the QoS 2 level is applied, thus indicating that its adoption must be carefully considered.

Keywords: 5G, IoT, Edge Cloud, MQTT, Latency, Test bed

1. INTRODUCTION

Many 5G experiments have been carried out in the last few years to show the potential of 5G technology and architecture in supporting an extremely large variety of services with different quality requirements. The key support to this development is the network slicing technology, which allows the sharing of the same physical communication infrastructure and computing resources for flexible and scalable configuration of multiple services with different performance requirements [1]. Three main classes of service have been identified: enhanced Mobile Broadband (eMBB), ultra Reliable Low Latency Communications (uRLLC), and massive Machine Type Communications (mMTC). According to specific service requirements, these classes can be applied to configure a given slice using a description defined in terms of NEtwork Slice Type (NEST) [2], which includes specific values to be assigned to relevant service level attributes supported by the slice, such as throughput, latency, and reliability.

Many 5G services can take advantage of the Internet of Things (IoT), that is typically enabled by a large number of sensors suitably placed in the environment, e.g. for monitoring purposes. The introduction of the 5G technology enhances the role of the IoT by integrating its functionality within a virtualized network infrastructure, possibly controlled by a Software-Defined Networking (SDN) approach, and the cloud. This results in high application potential for many different purposes. To this end, multiple aspects need to be considered in the implementation of a 5G use case, ranging from physical layer connectivity to virtualization capability and application-level protocols, up to service component placement in an edge and/or cloud computing environment, with the consequent impact on latency requirements.

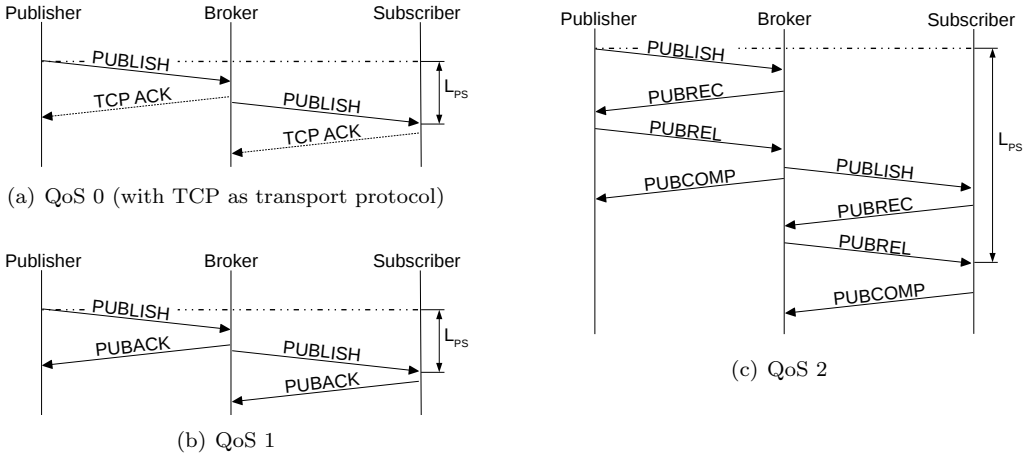
The Message Queue Telemetry Transport (MQTT) protocol is considered as a prominent messaging transport protocol to efficiently manage data originated from the IoT [3]. This protocol adopts a publish/subscribe procedure based on three main entities: the *subscriber* (S), the *broker* (B) and the *publisher* (P), which interact to collect data from remote devices and make them available to interested applications. The MQTT performance depends on the relative positioning of these entities in the access network, edge and remote cloud, and on the characteristics of the interaction which provides support for different quality of service levels. This work aims at performing practical measurements of different MQTT configurations to evaluate the impact of P, B and S placement on the end-to-end data delivery latency.

The paper is organized as follows. In section 2. an introduction to the MQTT protocol and related parameters is given. In section 3. the descriptions of testbed setup for latency measurements and data acquisition procedure are provided. In section 4. numerical evaluations are reported and discussed. In section 5. the conclusions of the work are drawn.

2. MQTT PROTOCOL FOR THE IOT

The MQTT is a very well known publish/subscribe protocol used in various environments, from web applications to IoT [3]. It is a messaging transport protocol positioned above the transport layer and used by different applications to collect and distribute data. Each MQTT message is composed by a *topic* and a *payload*: the former is used to characterize the content of the message whereas the latter contains the actual data to transmit. The MQTT message exchange involves three entities: publisher, broker and

978-1-7281-8423-4/20/$31.00



(a) QoS 0 (with TCP as transport protocol)

(b) QoS 1

(c) QoS 2

Figure 1: MQTT message exchange with different QoS levels.

subscriber. A publisher is the source of a message: it receives data from the application layer, adds a topic to the message as specified by the application, and sends it to the broker. The broker is the entity in charge of receiving messages from publishers and forward them to the subscribers according to the topic specified in the messages. Finally, a subscriber is the recipient of an MQTT message after having interacted with the broker by stating about which topics it is interested to receive data. All these message exchanges are performed by sending PUBLISH messages from the publisher to the broker and from the broker to the subscribers. Since MQTT was designed to work also on top of unreliable networks, it offers three levels of message delivery assurance, namely *Quality of Service* (QoS), specified for each PUBLISH message: QoS 0, QoS 1 and QoS 2. An example of an MQTT message exchange with different QoS levels over a TCP connection is depicted in Fig. 1. QoS 0 provides a best-effort (*at most once*) delivery of the PUBLISH message, as there is no guarantee that it is received at the MQTT level. The correct delivery of the message is delegated to the underlying transport protocol (i.e., TCP or UDP). QoS 1 provides PUBACK messages to acknowledge the reception of the PUBLISH. If a PUBACK is not received within a certain time, the PUBLISH message is re-transmitted. In this case, the PUBLISH is sent *at least once* to the receiver, that must take care of duplicate messages. The MQTT sender (either publisher or broker) stores the message until the acknowledgement is received (respectively from the broker or subscriber). QoS 2 is the highest level as it ensures that messages are delivered *exactly once*. This is done by means of PUBREC, PUBREL, and PUBCOMP messages. The reception of a PUBLISH is acknowledged with a PUBREC, that allows the sender to remove the original message from the queue. The sender then replies with a PUBREL, allowing the receiver to move the original PUBLISH forward. The message exchanged is then ended with a PUBCOMP. The time needed to complete an end-to-end delivery of a PUBLISH message is the publisher-to-subscriber latency L_{PS}. L_{PS} is given by the sum of (i) the transmission and propagation delay introduced by network links connecting the different MQTT entities, (ii) the processing delay, including packetization and queuing delay, at each node, and (iii) the additional delay introduced by the protocol due to the number of MQTT messages to be exchanged before the PUBLISH is eventually delivered to the application. Therefore, the QoS level chosen for a given message exchange has a direct impact on the value of L_{PS}, as shown in Fig. 1.

3. A TEST-BED FOR MQTT LATENCY MEASUREMENTS

To assess the behavior of the protocol, a testbed has been setup as depicted in Fig. 2. It consists of two Ubuntu 18 LTS Virtual Machines: VM_1 hosts the functionalities of both the publisher (P) and the subscriber (S), whereas VM_2 hosts the broker (B) functionalities. The reason why the two entities P and S are located in the same virtual machine VM_1 is to have a single time reference to accurately measure the end-to-end latency L_{PS}. The two VMs are connected by two separate virtual networks, Net_1 and Net_2, so that different delays can be set for each individual link, D_{PB} and D_{BS}, respectively. As a consequence, different deployments of the MQTT setup can be tested, emulating the placement of the three entities at the edge or core cloud. The variable link delays are emulated with the Linux traffic control command line tool (`tc`) [5], which allows to use the `netem` functionality to add delay, loss, duplication and further characteristics to the packets outgoing a given network interface. Fixed or variable delay can be set, the latter according to a specific distribution, such as normal or Pareto, further specified by its average value

and standard deviation. Mosquitto [6] is the software solution adopted to implement the broker, whereas a Python script based on the Paho library [7] has been developed to implement publisher and subscriber.

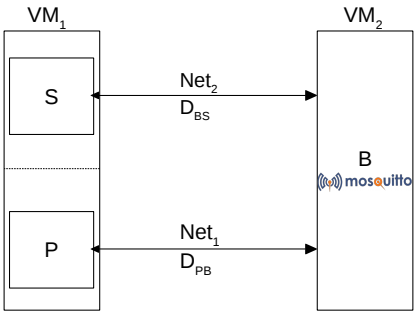


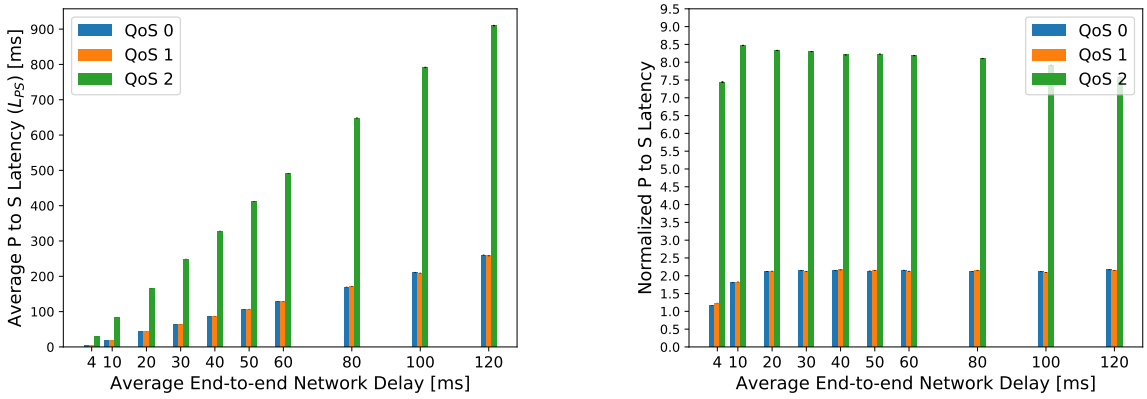
Figure 2: Setup of the MQTT testbed for latency measurements. P: Publisher, S: Subscriber, B: Broker.

To test the behaviour of the MQTT protocol several sets of experiments were run, generating 10000 PUBLISH messages in each experiment with a frequency of 100 messages per second. Each message had a fixed length of 20 bytes. The transport protocol chosen for this evaluation was TCP, by setting the RTO value so high that it is never reached and the latency can be studied without any conditioning due to the transport protocol. A symmetrical scenario was tested first, where both links were characterized by the same average delay, i.e. $D_{PB} = D_{BS}$. In this scenario, all three distributions supported by `tc` were used, with the following average delay values: $[2,5,10,15,20,25,30,40,50,60]$ ms. Anyway the results reported here refer to the normal distribution. Each set of measurements was repeated for the three different MQTT QoS values. Given the limited size of the TCP segments carrying the MQTT messages (up to around 600 bytes, measured in the highest aggregation case), the packet transmission delay for typical network interface transmission rates (e.g., 100 Mbit/s) can be considered negligible with respect to the emulated network delays. Then, the experiments were repeated in a few asymmetrical scenarios, by associating different average delay values to the two links Net_1 and Net_2, according to the normal distribution. These scenarios allow to consider different deployments of the MQTT entities. For instance, the two cases of a broker co-located with the publisher at the access or with the subscriber in the cloud can be emulated by choosing $D_{PB} = 0$ or $D_{BS} = 0$, respectively. Different placements of the broker and/or subscriber at the edge or core cloud can be tested by properly setting the relevant average delay values. Based on all these experiment configurations, the end-to-end latency L_{PS} from the publisher to the subscriber was measured.

4. NUMERICAL RESULTS

In this section, the measurement results obtained using the testbed in Fig. 2 are presented, considering a normal delay distribution in both the symmetrical and asymmetrical scenarios, and with different relative placements of the P, B, S entities. Very similar results were obtained with other delay distributions (not shown). In Fig. 3(a) L_{PS} is presented as a function of the average end-to-end delay, both expressed in milliseconds. As expected, the absolute value of the latency increases with the end-to-end delay, with a remarkable influence of the quality of service level, especially as far as QoS 2 is concerned. Therefore, in order to evaluate the impact on latency requirements of an MQTT service, not only the bare network delay needs to be considered, but also the contribution of the procedures related to the quality of service level chosen. In fact, the latency increases as shown in Fig. 3(b), where L_{PS} is reported as normalized to the end-to-end delay value. With QoS 0 and 1, in the only case where the round-trip delay (2×4 ms) is smaller than the message generation period (10 ms), it takes on average a single end-to-end interval to deliver the message, whereas a whole round-trip time is needed when the delay increases. The effect of the additional protocol exchanges in the QoS 2 case brings to an average L_{PS} at least four times higher.

The effect of the network delay on the aggregation of MQTT messages is shown in Fig. 4, where the average number of MQTT packets transported by each TCP segment is reported. The TCP segments were verified to be large enough (less than a typical MSS value, 1460 bytes) to collect all PUBLISH messages queued at the sender before the acknowledgement of the previous packet is received. The increasing delay on the links causes the aggregation of more MQTT packets, but this is only slightly influenced by the QoS level. In Fig. 5 a few asymmetrical scenarios are considered. The notation indicates the pair of delay values associated to Net_1 and Net_2, namely D_{PB} and D_{BS}. For instance, the 0 - 20 configuration represents the case of a broker co-located with the publisher and the subscriber placed in the remote cloud, whereas the 2 - 18 case refers to the broker and the subscriber located in the edge and remote cloud, respectively.



(a) (b)

Figure 3: P-to-S latency as a function of the average end-to-end network delay (symmetrical scenario) according to a normal distribution, varying the MQTT QoS level: a) average L_{PS} latency; b) average L_{PS} latency normalized to the average end-to-end network delay. 100 MQTT messages per second are published.

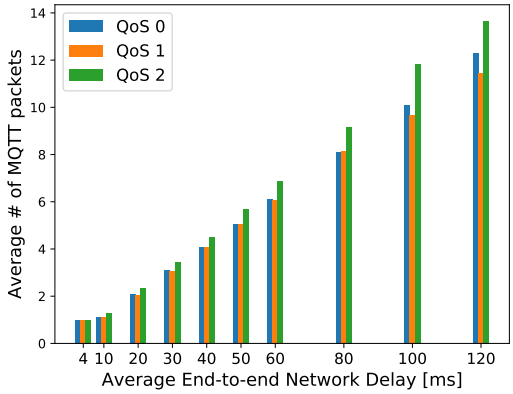


Figure 4: Average number of MQTT packets carried by a TCP segment as a function of the average delay with the normal distribution, for different QoS levels (symmetrical scenario).

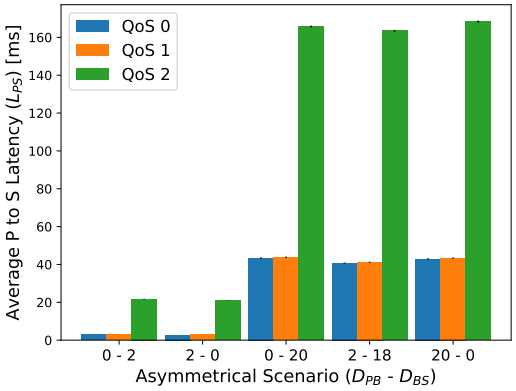


Figure 5: Latency L_{PS} for asymmetrical scenarios, represented by different pairs of delays D_{PB} and D_{BS} for different QoS levels.

The P-to-S latency is influenced by the overall end-to-end delay value and the QoS level, but not by how the P-B-S functionalities are placed. The absolute effect of the QoS 2 scheme is once again evidenced as particularly sensitive to the end-to-end delay when its value increases.

5. CONCLUSION

MQTT offers the functionalities to enhance IoT effectiveness while sharing data in the cloud among different applications. To this end MQTT operates with different QoS levels which influence the latency in data delivery. The results of this paper show by measurement how the network delay combined with the quality of service level of the MQTT protocol impacts on the latency. The main suggestion coming from the results is that the QoS configuration must be carefully chosen when the network delay increases, being it the main cause of a remarkable increase of the publisher-to-subscriber latency when a high QoS level is applied. Results from the asymmetrical scenario show that, for a given end-to-end delay, the broker placement does not influence the latency, still being the QoS level the main impacting aspect.

References

[1] "Network Slicing Use Case Requirements", GSM Association, 2018.
[2] "From Vertical Industry Requirements to Network Slice Characteristics", GSM Association, 2018.
[3] A. Banks, E. Briggs, K. Borgendale, and R. Gupta (Eds.), "MQTT Version 5.0," 07 March 2019.
[4] "3GPP TR38.801 Radio access architecture and interfaces."
[5] Tc-netem - Linux Foundation `https://wiki.linuxfoundation.org/networking/netem`
[6] R. A. Light, "Mosquitto: server and client implementation of the MQTT protocol," Journal of Open Source Software, 2(13), 2017
[7] Eclipse Paho `https://www.eclipse.org/paho/`