A comparative evaluation of AMQP and MQTT protocols over unstable and mobile networks

Jorge E. Luzuriaga*, Miguel Perez†, Pablo Boronat†, Juan Carlos Cano*, Carlos Calafate*, Pietro Manzoni*
*Department of Computer Engineering
Universitat Politècnica de València, Valencia, SPAIN
jorlu@upv.es, {jucano,calafate,pmanzoni}@disca.upv.es
†Universitat Jaume I, Castelló de la Plana, SPAIN
mperez@uji.es, boronat@uji.es

***Abstract*—Message oriented middleware (MOM) refers to the software infrastructure supporting sending and receiving messages between distributed systems. AMQP and MQTT are the two most relevant protocols in this context. They are extensively used for exchanging messages since they provide an abstraction of the different participating system entities, alleviating their coordination and simplifying the communication programming details.**

These protocols, however, have not been thoroughly tested in the context of mobile or dynamic networks like vehicular networks. In this paper we present an experimental evaluation of both protocols in such scenarios, characterizing their behavior in terms of message loss, latency, jitter and saturation boundary values. Based on the results obtained, we provide criteria of applicability of these protocols, and we assess their performance and viability. This evaluation is of interest for the upcoming applications of MOM, especially to systems related to the Internet of Things.

I. Introduction

Modern Internet-based applications are becoming more oriented towards the interaction of wireless and mobile devices with cloud resources and services. For many years HTTP has been used as the reference communications protocol in this context. HTTP is a very wide spread protocol, and APIs for its use are available basically for every programming language.

However, more flexible middleware systems have been developed to ease the design of cloud-based applications. Significant research efforts have been dedicated to define new communication systems that connect distributed components via message passing; they are called Message Oriented Middleware (MOM). The basic idea of MOM is that communication takes place by adding messages to distributed queues, and by getting messages from those queues. Based on the model of Message Oriented Middleware, many protocols have been developed, e.g. DDS, STOMP, XMPP. The two most widely used proposals are: the Advanced Message Queuing Protocol (AMQP) and the Message Queuing Telemetry Transport (MQTT). AMQP was created in 2003 by John O'Hara of JPMorgan Chase and MQTT was originated in 1999 by Andy Stanford-Clark of IBM. Both protocols provide some platform agnostic methods to improve the communication and ensure that information is safely transported between systems.

AMQP is an open standard for enterprise messaging, designed to support messaging for almost any distributed or business application [6]. It works like instant messaging or email, and the difference towards these available solutions is that AMQP comprises both a network protocol, which specifies the entities (producer/consumer, broker) to interoperate with each other, and a protocol model, which specifies the representation of messages, and the commands to interoperate among the entities. Furthermore, AMQP messages are self-contained, and data content in a message is opaque and immutable. Also, there is no limit for the size of a message; it can either support a 4 *GByte* message or a 4 *KByte* one, in any case ensuring security, reliability, and performance.

MQTT is a lightweight machine-to-machine messaging protocol [2], and it has a clear focus on the mobile sector. Its information exchange procedure is resource-efficient, and it does not specify a particular data format. Additionally, it provides security that all messages transmitted even if the connection breaks off briefly, solving problems that arise upon unreliable communications.

Both protocols use similar techniques for message delivering. When a message is sent by a client to a message broker, it is placed in a queue, and after that, all customers subscribed to this queue automatically receive the message as a push notification [9]. To support a wide variety of usage scenarios, AMQP offers several possibilities for message delivery, namely: *point-to-point*, *store-and-forward*. However, the MQTT protocol provides a basic messaging *topic* based on specified topics (subscription's name) without possibility of *store-and-forward* mechanisms to support message delivery [13].

Both of these protocols were designed to facilitate the dialogue among the components of a system, by simplifying message exchange independently of their underlying platforms. There are libraries available for the most popular programming languages, and there are implementations for the most common operating systems and platforms. In addition, they take into consideration security and confidentiality issues without affecting significantly the communications performance.

In this paper we evaluate AMQP and MQTT, and we compare their capabilities and capacities through measurements under a mobile or unstable wireless network testbed. We call *unstable networks* those in which links can be frequently modified or broken without control. Examples of unstable networks are mobile networks or wireless networks in urban environments, like community networks or vehicular networks,

978-1-4799-6390-4/15/$31.00 ©2015 IEEE

which suffer from channel interferences or node blackouts. Our goal is to determine whether these protocols provide a satisfactory service, depending of the applications' load needs, in terms of message size and communication rates.

We present the evaluation results regarding the effect of a mobile producer/publisher changing from one WiFi access point (AP) to another in the same IP network. We developed a synthetic load generator, called *amqperf*, that sends messages with a sequence number to detect losses or messages delivered in different sending order. The size of messages and the frequency in which they are sent by the producer can be modified. Using a simple scenario composed by one producer/publisher, one consumer/subscriber and one message broker, we observe the effect of network changes on the messages' jitter, and we detect the saturation boundary values with the specific hardware used in the tests.

The rest of the article is organized as follows: Section II presents a literature review related to the topic. Section III provides a description of the methodology used in this work, showing how measurements have been done in order to be reproducible. Section IV presents the results and, finally, Section V concludes the paper, highlighting the next steps to follow in this research line.

II. Related work

The AMQP and MQTT protocols have proved to be useful in production scenarios, and have been used in challenging applications, including Autonomous Computing [4], Cloud computing [14] or in aspects related to the Internet of Things (IoT) [15].

There are several works in which the AMQP and MQTT protocols are evaluated separately. In [10] the performance of AMQP is assessed using Infiniband and Gigabit Ethernet networks with Qpid as AMQP middleware. Five simple synthetic benchmarks modeled after the OSU Micro-benchmarks for MPI were used. They exercise the number of Publishers, the number of Consumers, and the Exchange type. Each benchmark measures performance for data capacity (the amount of raw data in MegaBytes per second), message rate (the number of discrete messages transmitted), and speed (average time one message takes to travel from the publisher to the consumer).

In [1] authors present a way to evaluate the performance of AMQP by using an adapted version of the well-known *SPECjms2007* and *jms2009-PS* benchmarks. This would allow to compare AMQP with other messaging systems such as *JMS* (Java Message Service) in terms of performance, stability and scalability.

In [5] a performance comparison between AMQP and RESTful *web services* is presented. Three different tests are performed, which consist of several client applications sending messages during 30 minutes to the broker or the web server, respectively; once the messages arrive to the server they are stored in a database. Then, the average number of messages per second that have been sent is compared to the total number of messages stored in the database. They conclude that, when the AMQP protocol is used to exchange messages, a larger number of messages per second is supported.

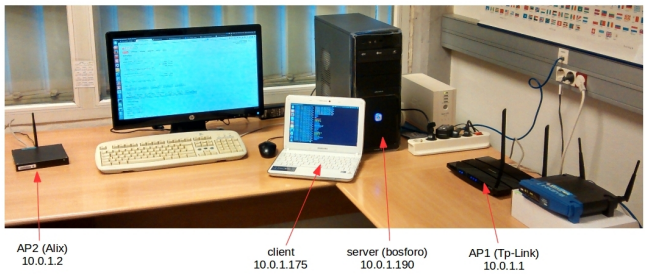


Fig. 1: A picture of the scenario.

A study about MQTT, a "light weight" publish-subscribe based messaging protocol, is presented in [7]. The correlation between the end-to-end latency and loss of system messages is studied. Three different QoS levels with different sizes of payload (from 1 to 16 Kbytes) are tested on a real world scenario with both wired and wireless clients using 3G. They prove that there is a strong correlation between these two variables.

However, few studies have focused on the effectiveness of both protocols over unstable networks.

III. Methodology of the experiments

The publish/subscribe topology allows us to use a simple scenario with decoupled components in order to interoperate among them. In our experiments, we use a message *producer/publisher* client, which at a given frequency, sends AMQP/MQTT messages of a prefixed size to a message broker.

In the case of AMQP, the message-broker accepts incoming messages from a producer in an exchange (an exchange is essentially a router [12]) and, based on a set of criterions routes the messages to a specific queue. In the case of MQTT, the message-broker forward the incoming messages from publishers directly to the subscribers. A subscription is initially created by a client application with a simple subscription name or a predefined topic.

In our scenario, the message *consumer/subscriber* is connected to the message-broker, and it is always ready to get or consume messages. The message-broker and the *consumer/subscriber* client are executed on the same computer. To reach the message-broker, the *producer/publisher* is connected to a WiFi access point within the same network. A picture of the scenario configuration used in our testbed is shown in Figure 1.

The consumer/subscriber records in a log file the sending (timestamped by the producer in each message) and reception times, along with the sequence number. There is not strict synchronization between the producer and consumer clocks. When there are changes in the producer/publisher link, the regularity of message reception is affected. Since the inter-message times are modified, message bursts can be delivered to the consumer, and even the sending order can be changed.

Current implementations of AMQP and MQTT use TCP/IP connections for communication in order to enhance reliability.

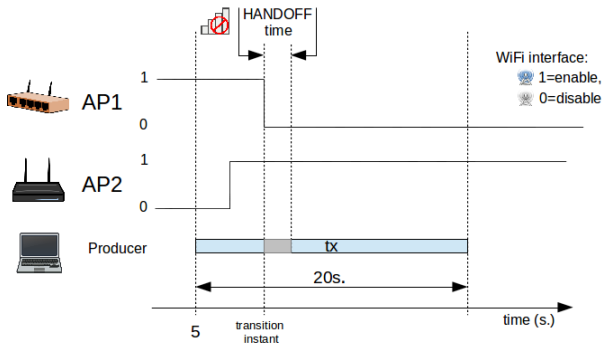


Fig. 2: Simulating the node mobility in an indoor scenario.

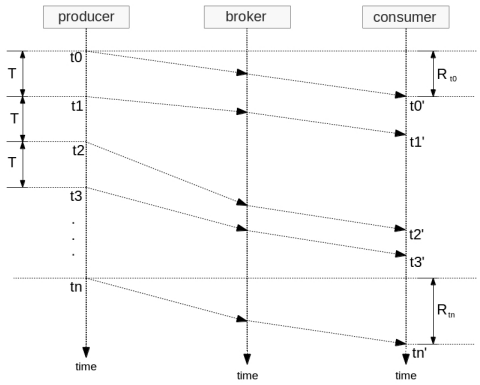


Fig. 3: Times involved in the experiments.

If the producer/publisher's connection suffers an interruption, the client (producer's AMQP or publisher's MQTT) accumulates the messages in its buffer and keeps it for a limited time, waiting till the connection with the message broker is re-established. The problem appears when, in the *producer/publisher* part the storage buffer capacity is depleted, thereby causing message losses.

A testing application, which we call *amqperf*, has been developed to generate a workload for the message queuing system on the producer. *Amqperf* uses the *RabbitMQ* library [16], which is an AMQP implementation, and the *Paho* library, which is an open source MQTT implementation. *Amqperf* also works as a consumer/subscriber.

In our experiments, the duration of the tests was about 20 seconds, which is sufficient to check the access point migration of the message producer. During the tests we checked whether there were messages losses or if messages arrived out of order.

In order to simulate node mobility in an indoor scenario, we used a set of scripts that shut down or activates the routers' radios and disassociate/associate all client devices. A schema of this approach is shown in Figure 2.

Due to the asynchrony among the internal clocks of the different entities of a distributed system, we cannot determine an exact latency value. Instead, we have calculated the variation in the delay of the received messages, i.e., the *jitter*.

The n^{th} message inter-arrival time jitter is computed through the following equation:

$$J_n = t'_n - t'_{n-1} - T$$

where t'_n is the arrival time of message n to the consumer, and T is the (fixed) inter-message production period. T is one of the variables fixed for each experiment. Note that, with this formula, we are not concerned by a possible asynchrony between the producer and the consumer, which are executed in different computers. An example of the timing involved in the experiments can be seen in Figure 3.

The reference for the values used in the test is about 5 Mbps, which is the bandwidth needed, for example, to support high definition video streaming. This value can be reached, for instance, by transmitting messages of 12500*B* (12.5*KBytes*) every 0.02 seconds. We made some tests to detect the point at which messages start being lost in both cases: with and without access point migration for the producer. The obtained values are detailed in the following section.

The message broker was created on a server with an AMD 8-core processor and 16*GBytes* of RAM memory. The client had an Atom N450 processor and 1*GByte* of RAM memory. Both of them were running Ubuntu 12.04 GNU/Linux distribution. For the wireless network we have used the OpenWRT GNU/Linux distribution with *Attitude Adjustment* version on an Alix PC-Enginees (alix2d2) and a Tplink (TL-WDR3600) routers. The tests were run on a dedicated LAN without external traffic.

IV. Results

In this section we present our experimental results. The scenario is based on a wireless producer which migrates from one access point to another maintaining the same IP address; the TCP connection between the producer/publisher and the message broker is not affected.

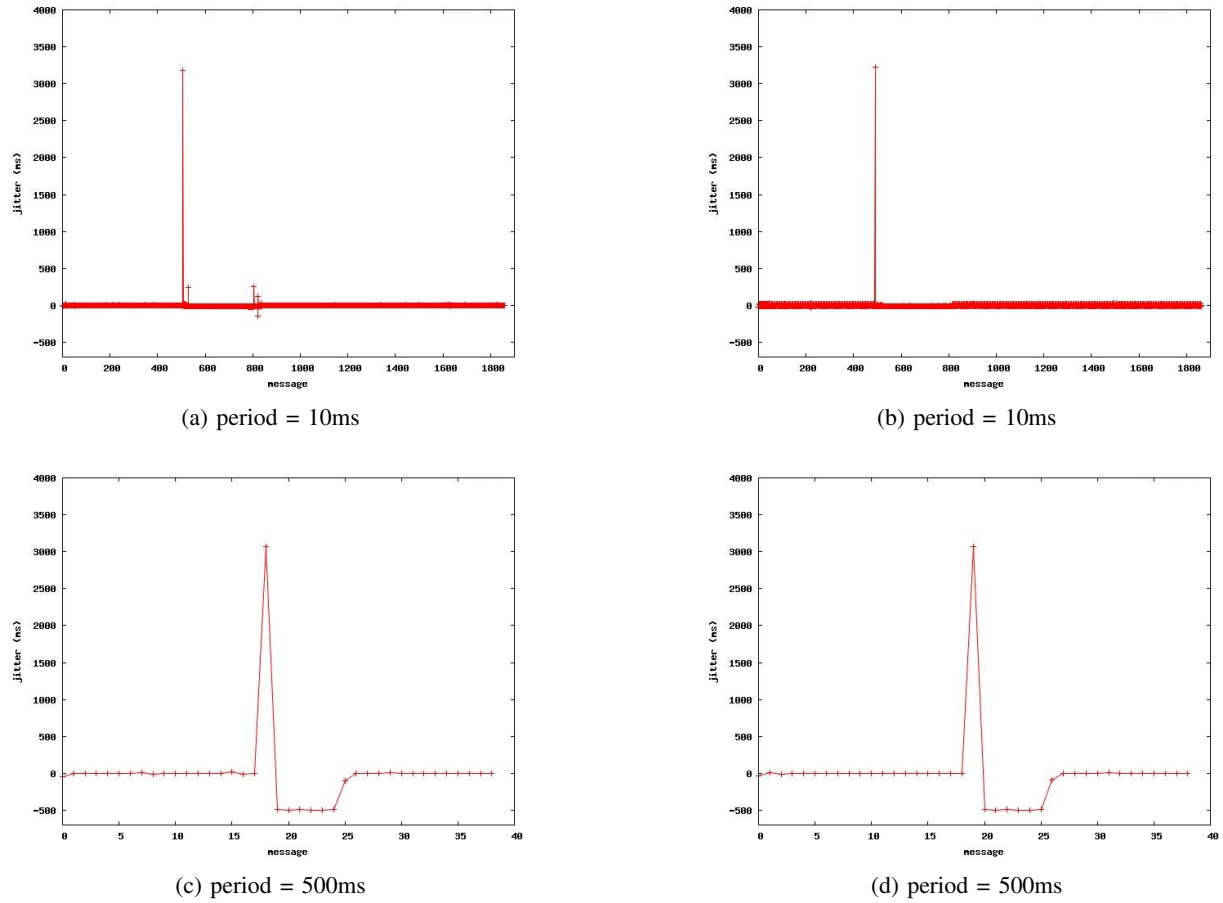
For the experiments, we have used a completely dedicated network without external traffic. The tests were repeated 100 times for each combination of inter-message period and message size. We analyse the behaviour for each scenario, and we generalize it through a cumulative distribution function.

A. Behaviour during access point transition

AMQP and MQTT protocols ensure that, when a client reconnects, it does not repeat messages and resumes the previous session with the message broker.

In order to understand the event, when a producer is migrating from one access point to another, we provide Figure 4. These figures show the typical jitter behaviour for each message received by the consumer/subscriber.

In Figures 4a and 4b, we can see a vertical line close to the message number 500*th* that reaches 3.3 seconds. These figures were obtained producing and sending, during 20 seconds



(a) period = 10ms

(b) period = 10ms

(c) period = 500ms

(d) period = 500ms

Fig. 4: Jitter's behaviour on the producer migration between access points, while it is sending messages of 512 B using: AMQP (left) and MQTT (right).

messages of 512 Bytes with an inter-message period 10*ms*. Due to the high amount of messages, it is difficult to distinguish the transition event. So, in Figures 4c and 4d, we show the profile of the curve with an inter-message period of 500*ms* in which we can see a positive peak corresponding to the hand-off time, and also jitter with negative values due to the reception of a burst of messages which the producer retained during the communication's interruption. As expected, the number of messages with negative jitter can be approximated by the peak positive jitter divided by the message producing period. For instance, in Figure 4d, it is $3000/500 \approx 6$ messages.

Also, Figure 4a shows a small oscillation after the hand-off peak when using the AMQP protocol. We have found that, during the message burst, the delivery follows a LIFO (last-input first-output) order, which results in messages consumed in inverted order. This does not occur with MQTT protocol, where the delivery is in order.

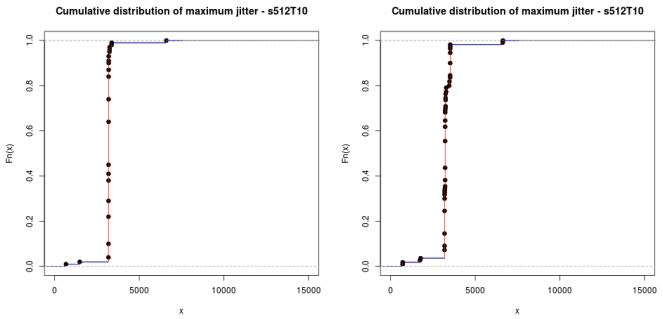
B. Jitter analysis

We repeated the experiments over the same scenario, with the same combination of message production rates and message sizes. We used sizes between 0.5 *KBytes* and 6 *KBytes*, and periods between message production of 10, 50, 100, 500 and 1000 *ms*. We know that the maximum jitter in our tests is the consequence of access point migration given that the network had no external traffic, and that the workloads used in these tests do not saturate the system.

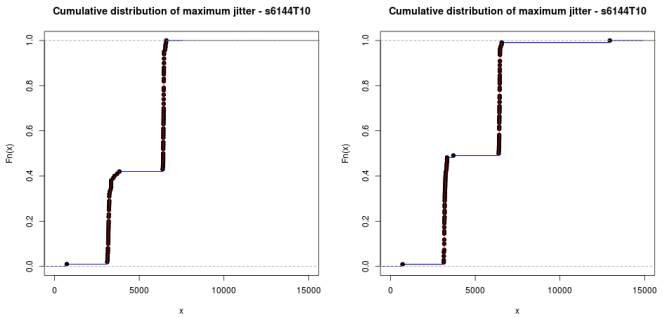
Using the Cumulative Distribution Function on the set of experimental data, we have analysed the behaviour of the message's jitter that arrives to the consumer focusing on the instant when the producer makes an access point migration.

Figure 5 shows the distribution function of the jitter using a period of 10*ms* for message production and message sizes of 512 *Bytes* and 6 *KBytes* for each protocol tested. When message size is 512 *Bytes* (Figures 5a and 5b), we observe that the jitter value is concentrated around 3.3 seconds, with sporadic cases of jitters of 7 seconds, without significant differences between these protocols. When the message size is bigger (Figures 5c and 5d) about half of the cases present jitter values close to 3.3 seconds while the other half of the cases double this value. We consider that this behavior for bigger messages is due to the fragmentation of their payload by both protocols.

To study the jitter evolution, we have used the statistical analysis for rounding mode values (rounded to the nearest hundred) to fit the most representative value instead of using the value that appears most often in the data sets. We have



(a) message size = 512 Bytes (b) message size = 512 Bytes



(c) message size = 6 KBytes (d) message size = 6 KBytes

Fig. 5: Cumulative Distribution Function of the maximum jitter with a inter-message production period of 10 ms, using: AMQP (left) and MQTT (right).

represented the jitter's mode in two ways: as a function of the message size (Figure 6a) or as function of the inter-message period (Figure 6b).

C. Workload boundary

In order to know the capacity of the messaging system to handle heavy workloads, we executed the experiments without access point migration. There is, therefore, no interruption in the wireless link between the producer/publisher and messaging broker. Note that these saturation boundary values can be dependent on the platform used, and even on their configuration.

A typical user application sends a few messages per second, with average load below 5 Mbps, which is well managed both by message protocols and the network. Performing this exhaustive delimitation of the workloads, in Figure 7 we show an approximation of the capacity of the system in terms of message size and number of messages produced per second.

For loads above the lines in each case, the system is saturated, causing that a certain proportion of all produced messages do not arrive to consumers. The limits are close to 20 Mbps, which is near to the bandwidth that we have obtained with the *iperf* tool for the TCP test.

We note that the payload limit of a message in the MQTT protocol is greater than for AMQP. We consider that is mainly caused by the difference between the frame header: AMQP

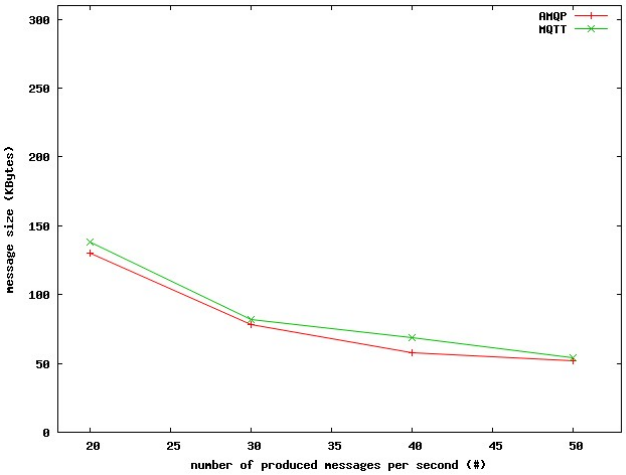


Fig. 7: Threshold limit of messages losses for different inter-message period and message size.

has a fixed size header of 8 *Bytes* while MQTT has only a 2 *Byte* header.

V. Conclusions and Future Work

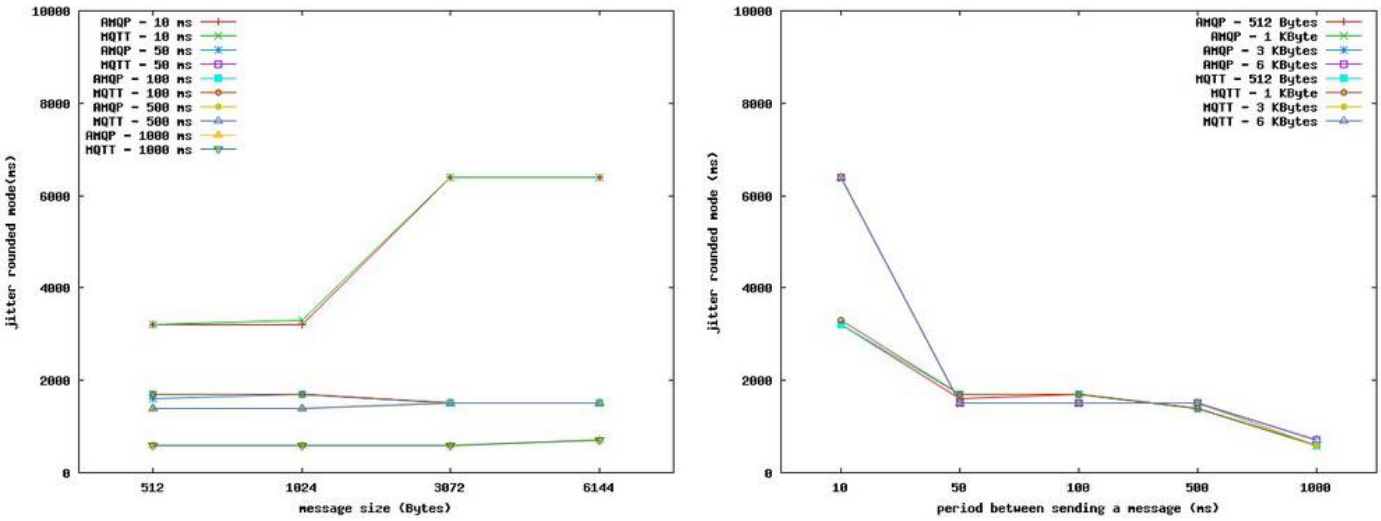
In this paper we presented an experimental analysis of the behavior of two application protocols based on Message Oriented Middleware standards, namely AMQP and MQTT. We focussed on the information exchange over unstable and mobile networks, like in vehicular networks, to provide a characterization of the *publish-and-subscribe* models in these scenarios.

We have evaluated scenarios where the producer/publisher suffers a handover process to measure effects such as the variability of the jitter and the information loss, with a simple workload model of one producer/publisher and one consumer/subscriber, in an extended wireless network (i.e. several access points conforming a same Service Set). We have observed that the mean jitter values during transitions tend to oscillate between 3 and 6 seconds, although 7 seconds peaks have also been detected for high transmission rates (e.g., 100 messages per second).

We have found that, during message bursts, the delivery follows a LIFO (last-input first-output) order, which results in messages consumed in inverted order. But this is not done MQTT protocol, where the delivery is always in order.

We have demonstrated that there is no information loss during the hand-off; then, we can say that the messaging system of these protocols is robust, and that it guarantees message delivery without losses. Messages losses are present only when, in the producer side the load is higher than its system buffer capacity.

In order to select the right protocol to build systems and applications with mobile communications over unstable network environments, both of these protocols can be used. The application/system architect's decision to choose one of them, will be determined according to different criteria, such as security and energy efficiency. AMQP offers more aspects related to security [17], and MQTT is more energy efficient [7].



(a) message size

(b) inter-message period

Fig. 6: Evolution of maximum jitter as a function of (a) and (b)

We recommend the use of AMQP protocol to build reliable, scalable, and advanced clustering messaging infrastructures over an ideal WLAN, and the use of MQTT protocol to support connections with edge nodes (simple sensors/actuators) under constrained environments (low-speed wireless access).

As future work, we plan to evaluate these protocols on more complex scenarios in which a roaming producer changes between IP networks and, consequently, the active TCP connection has to be reset.

ACKNOWLEDGMENTS

This work was partially supported by the *Ministerio de Ciencia e Innovación*, Spain, under Grant TIN2011-27543-C03-01.

REFERENCES

[1] Appel, Stefan and Sachs, Kai and Buchmann, Alejandro, *Towards benchmarking of AMQP*, ACM, Proceedings of the Fourth ACM International Conference on Distributed Event-Based Systems, 99–100, 2010.

[2] Chen, Whei-Jen and Gupta, Rahul and Lampkin, Valerie and Robertson, Dale M. and Subrahmanyam, Nagesh, *Responsive Mobile User Experience Using MQTT and IBM MessageSight* IBM Corp., 2014.

[3] Gluhak, A. and Krco, S. and Nati, M. and Pfisterer, D. and Mitton, N. and Razafindralambo, T., *A survey on facilities for experimental internet of things research*, IEEE, Communications Magazine, pp. 58-67, 2009.

[4] Gusmeroli, S. and Piccione, S. and Rotondi, D., *IoT@Work automation middleware system design and architecture*, IEEE International Conference on Emerging Technologies and Factory Automation ETFA, 2012.

[5] Fernandes, J.L. and Lopes, I.C. and Rodrigues, J.J.P.C. and Ullah, S., *Performance evaluation of RESTful web services and AMQP protocol*, IEEE ICUFN, pp. 810–815, 2013.

[6] Kramer, Joshua *Advanced Message Queuing Protocol (AMQP)*, Linux Journal 187, Houston, TX, 2009.

[7] Lee, Shinho and Kim, Hyeonwoo and Hong, Dong Kweon and Ju, Hongtaek *Correlation analysis of MQTT loss and delay according to QoS level*, International Conference on Information Networking, pp. 714-717, 2013.

[8] Luzuriaga, J. and Perez, M. and Boronat, P. and Cano, J. C. and Calafate, C. and Manzoni, P. *Testing AMQP Protocol on Unstable and Mobile Networks*, Internet and Distributed Computing Systems, 7th International Conference, IDCS 2014, Italy, Proceedings Lecture Notes in Computer Science, pp. 250-260.

[9] O'Hara, John, *Toward a Commodity Enterprise Middleware*, ACM, Communications Magazine, 2007

[10] Subramoni, Hari and Marsh, Gregory and Narravula, Sundeep and Lai, Ping and Panda, Dhabaleswar K. *Design and evaluation of benchmarks for financial applications using advanced message queuing protocol (AMQP) over infiniband*, Workshop on High Performance Computational Finance, WHPCF 2008.

[11] Vermesan, O. and Harrison, V. M. and Kalaboukas, H.K. and Tomasella, M. and (Eds.) *The Internet of Things - Strategic Research Roadmap*, Cluster of European Research Projects on the IoT, CERP-IoT, 2009.

[12] Vinoski, S, *Advanced Message Queuing Protocol*, IEEE Internet Computing, vol.10, 2006.

[13] Cohn, Raphael *StormMQ: A Comparison of AMQP and MQTT*, Available: www.stormmq.com, 2011.

[14] OpenStack, Foundation *AMQP and Nova*, Available: http://docs.openstack.org/developer/nova/devref/rpc.html, 2014.

[15] IMatix, Corporation *Security and Robustness*, Available: http://zeromq.org/docs:welcome-from-amqp, 2014.

[16] Pivotal Software, Inc., *Messaging that just works*, Available: https://www.rabbitmq.com/reliability.html, 2014.

[17] OASIS, Standard *OASIS Advanced Message Queuing Protocol AMQP Version 1.0*, 2014.