Handling Mobility in IoT applications based on the MQTT protocol (Author Version)

Jorge E. Luzuriaga[1], Miguel Perez[2], Pablo Boronat[2], Juan Carlos Cano[1], Carlos Calafate[1], and Pietro Manzoni[1]
[1] Department of Computer Engineering
Universitat Politècnica de València, Valencia, SPAIN
jorlu@upv.es, {jucano, calafate, pmanzoni}@disca.upv.es
[2] Universitat Jaume I, Castelló de la Plana, SPAIN
{mperez, boronat}@uji.es

***Abstract*— Connectivity clearly plays an important role in Internet of Things (IoT) solutions, and the efficient handling of mobility is crucial for the overall performance of IoT applications. Currently, the most widely adopted protocols for IoT and Machine to Machine (M2M) environments, namely MQTT, CoAP or LWM2M, are directly dependent on the TCP/IP protocol suite. This suite is highly reliable when using wired networks, but it is not the best solution in the presence of intermittent connections. In this work we provide a solution to improve MQTT with an emphasis on mobile scenarios. The advantage of the solution we propose is making the system more immune to changes in the point of attachment of mobile devices. This way we avoid IoT service developers having to explicitly consider this issue. Moreover, our solution does not need extra support from the network through protocols like MobileIP or LISP. The obtained results show that our proposal, based on intermediate buffering, guarantees that there is no information loss during hand-off periods due to node mobility; furthermore, based on discrete event simulation results, we determine the maximum number of sources and the required amount of buffers for a mobile node.**

Keywords—mobility; iot; m2m; handover; mqtt; handoff management.

I. INTRODUCTION

The *Internet of Things* (IoT) paradigm basically refers to the concept of connecting any device to any other device through the Internet. Devices can range from cell phones to coffee makers, washing machines, headphones, lamps, wearable devices, and basically any device that may have an associated IP address.

A large number of communication protocols, also referred to as "middleware", are used in nowadays IoT industries. From the industrial protocol used to collect temperature data on a sensor, to the communication protocol used to send this data to a server in the Cloud, there are various alternative middleware options for building an end-to-end IoT solution.

Currently, the most widely adopted protocols in the IoT and M2M[1] fields are MQTT[2], CoAP[3] and LWM2M[4].

Connectivity clearly plays an important role in IoT, and the efficient handling of mobility is crucial for the overall performance of any IoT application. To achieve stable and reliable communications, the crucial aspects to be considered are: (a) links can be frequently modified or broken without control, (b) channels can suffer from interferences, (c) nodes can become isolated, and (d) the service offered may not be available at any time. The presence of one or several of these factors have negative effects on the quality of the information transmission, producing: data loss, fail to access a service, and bad overall performances.

All the protocols indicated referred above, i.e., MQTT, CoAP and LWM2M, are directly dependent on the TCP/IP protocol suite; this is a highly reliable set of protocols when using wired networks, but do not perform adequately in intermittently connected scenarios. For example, in a broken connection case, using TCP over IP will cause the receiver to inform the sender about which packets must be resent. This approach would work perfectly if, after the re-connection, the IP address of the nodes remained the same, but it will fail when one of the nodes changes its address. Unfortunately, most applications do not support IP address changes, being severely affected by these events. These issues are totally outside the developer's control, who usually assume that the middleware used takes care of these problems.

In this work we focus on this specific issue, providing a solution to adapt MQTT protocol to mobile scenarios. The advantage is that developers will not have to explicitly consider the changes in the point of attachment to the network. In addition, there is no need to have the network support through protocols like MobileIP [5] or LISP [6].

The rest of this paper is organized as follows: Section II presents a brief literature review. Section III describes the MQTT protocol. Section IV provides a description of our proposal and the methodology used in this work. Section V presents the results of the evaluation, and Section VI extends the results of the empirical evaluation using queueing networks. Finally, some conclusions are presented in Section VII.

II. RELATED WORKS

The general issue of dealing with node mobility is typically taken care by advanced solutions like Mobile IP or LISP. This is anyway a very active area, and new standards and protocols have been developed at the application level to connect all the things that surround us to the Internet.

In [7], the MQTT and CoAP protocols are analyzed comparatively in terms of latency and throughput, using a network emulation tool to simulate different network characteristics (like packet loss, delay, and bandwidth limitations) in their connections. The authors have developed a platform called "*Ponte*", which is basically a message broker to adapt different IoT protocols to a publisher/subscriber architecture model. Currently, this platform is included in the Eclipse repositories [8]. The authors suggest using the MQTT protocol in the presence of high delays, and the use of CoAP when the data traffic increased significantly, since CoAP uses UDP as the transport protocol.

In [9] authors present a study of the reliability mechanism used by the MQTT-S and CoAP protocols. The authors demonstrate that both protocols would achieve a better performance by modifying the Retransmission Time Out (RTO) calculation method from a fixed to an adaptative value. The authors state that the network conditions are not taken into account when using a fixed RTO value (of ten to fifteen *seconds*), and that the features provided by a publish/subscribe model will be not fully exploited. Their evaluations show that, when considering the network conditions as a parameter for calculating the RTO value, these protocols would avoid increasing spurious retransmissions and resource wastage (with a small RTO value), while also avoiding to react too late when recovering packet losses (with a large RTO value).

In [10] the authors determine the correlation between end-to-end latency and message loss. They use a real world scenario with both wired and wireless clients exchanging messages of different sizes (payload from 1 to 16 Kbytes), using the three different MQTT QoS levels available [see section III].

Mobility support of the next generation of smart devices is one of the most important issues in the future Internet [11]. Our work is aimed at scenarios where the flow of information is basically unidirectional, with just a few signalling messages in the upward direction. So, we need a simpler solution that avoids extra reliability functions besides those already provided by the IoT middleware.

III. Overview of the MQTT Protocol

MQ Telemetry Transport (MQTT) [1] is a lightweight messaging protocol designed to be open, simple, lightweight and easy to implement. MQTT supports the publish and subscribe messaging model, i.e., when a message is sent by a client it is placed in a queue on the message-broker and, afterward, all customers subscribed to this queue automatically receive the message as a push notification.

These characteristics make it ideal for use in constrained environments, like for example when the network is expensive, has low bandwidth, is unreliable, or when it runs on an embedded device with limited processor or memory resources.

The messages that are transported by MQTT have an associated Quality of Service level (QoS) and Topic Name. The protocol defines three levels of QoS, namely level zero (QoS=0), where the sender sends the message only once and no retries are performed, and levels one and two (QoS>0), where the protocol ensures that no published message will be lost, thus involving a one- or two- steps acknowledgement process, respectively. However, the decision to use one of these levels impacts on the application performance as well as on the use of bandwidth and battery life on devices.

To guarantee that a message has been received, an acknowledgements exchange mechanism taking place between the client and the broker is used. This mechanism is associated with a Quality of Service level specified on each message. With QoS=0, which means "fire and forget" [1], it totally depends on the reliability of TCP/IP. If a TCP/IP session is broken, the messages are lost.

When the QoS level is set to one, the protocol ensures that a message arrives at the server "at least once". A published message is stored in the publisher internal buffer until it receives the ACK packet. Once the acknowledgement is received, the message is discarded from the buffer, and the delivery is complete. If a TCP/IP session is broken, only a few messages can be stored in the buffer until the time when the session is again restored and acknowledgement messages again are delivered.

When the QoS level is set to two, the protocol guarantees that a published message will be delivered "exactly once". Neither loss nor duplication of messages are acceptable, a requirement which is met through a two-step acknowledgement process [1]. The problem associated with this level is the increased overhead, since the transmission of one message involves the interchange of four messages.

IV. Proposed Solution

In this section we describe the proposed enhancements to the MQTT protocol. Our proposal maintains the *publish/subscribe* approach but decouples the pure data generation process by the data sending process. It is based on a technique called *intermediate buffering*. This decoupling allows for recovery when the communication channel presents disruption periods, even if these are very frequent and last for several seconds, a situation where TCP fails to recover from.

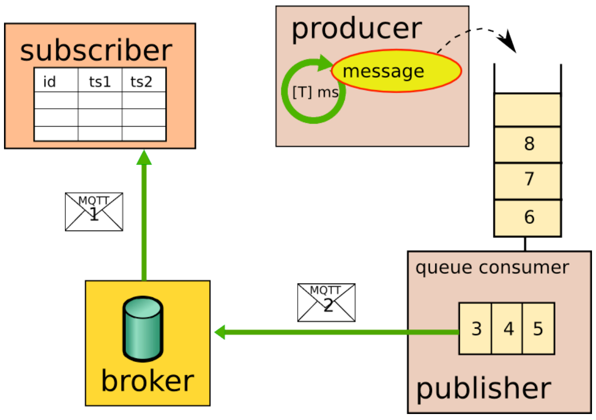

Fig. 1. Structure of the proposed solution based on intermediate buffering.

Figure 1 illustrates our proposal. We consider a message producer that continuously generates messages with a given frequency. An MQTT publisher takes the produced messages and turns them into MQTT messages, to be published with a given periodicity to a predefined MQTT broker. The latter will then forward the incoming messages directly to subscribers. A subscription is initially created by a client application on a predefined topic (simple subscription name).

When the connection between the node publisher and the message-broker suffers an interruption, the node enters in *roam mode*. The in-flight published messages (messages that have not received the acknowledgement from the message-broker) are stored in the MQTT internal buffer, which is constrained in terms of capacity. These messages are delivered only when the node recovers the connection with the last access point (that means recovering the last IP address); otherwise these messages are lost.

When a longer disruption takes place, our intermediate buffer is in charge of storing all the published messages that have not received the acknowledgement. Meanwhile, the MQTT network control mechanism manages the creation of the new connection, and the correct closing of the aborted session. With the new connection, and independently from the IP address that the node obtain once the connection with the MQTT-broker is re-established, we can guarantee the delivery of all these messages in the same order as originally published, followed by the messages which are continuously being generated.

A. *Testbed implementation details*

We implemented and evaluated our proposal using a testbed representing a small-scale scenario (see Figure 2). In our scenario, the message-broker and the subscriber client are executed on the same computer, and the publisher is connected through one of the two WiFi access points. A DHCP server is running on each access point and configured to assign IP addresses to clients on different and independent subnets. Inside a subnet, all clients communicate with each other, and routes to the MQTT message-broker are always available.

In the testbed, to generate workloads for the message queuing system, we have developed a testing application called *mqttperf*. *Mqttperf* uses the *Paho* [14] library, which is an open source implementation of MQTT protocol.

Each time a new message arrives, the sequence number and the production time stamps are recorded in a log file together with the reception time-stamp. When a client leaves a subnet and enters another one, the regularity of message exchange is affected due to the inter-message arrival times, the bursty delivery to the subscriber, message losses, and out-of-order arrival of messages. All these issues are considered in the data processing phase at the subscriber side.

In order to simulate node mobility in an outdoor scenario, we used a set of scripts that turned the APs' radios up or down, thus associating or disassociating the related client devices. A schema of this approach is shown in Figure 2.

Due to the asynchrony between the internal clocks of the different entities in a distributed system, we cannot obtain an accurate latency value [13]. Instead, we have calculated the variation in the delay of the messages received (*jitter*). This way we are not affected by a possible asynchrony among the producer, broker and consumer, since we take as the reference clock only one of them, namely the subscriber client clock.

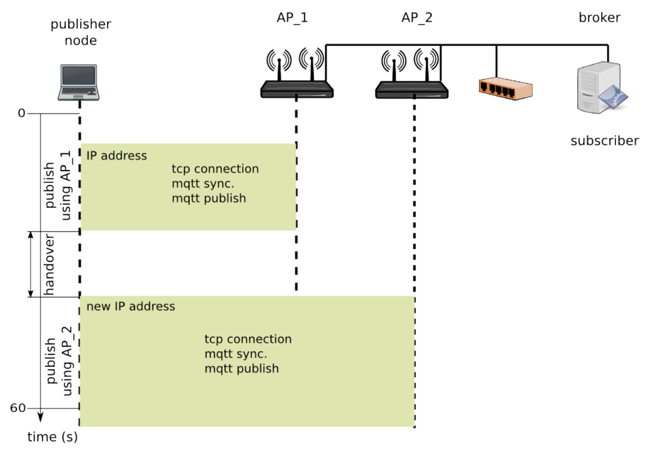

Fig. 2. Simulating node mobility in an outdoor scenario.

For example, let us consider two consecutive messages received by the subscriber. For message *n*, its arrival time is defined as t'_n. T is the inter-message production period; for each experiment, T is a fixed parameter. So, the inter-arrival jitter time to the n^{th} message is computed through the following formula:

$$J_n = t'_n - t'_{n-1} - T \qquad (1)$$

In terms of channel bandwidth, we have taken as a reference, the bandwidth needed to support high definition video streaming, which is about 5 Mbps. In the test, using the *mqttperf* tool, this value can be reached, for instance, by transmitting messages of 12500*B* (12.5*KBytes*) every 0.02 *seconds*.

We made a set of tests to detect the point at which messages start being lost in both cases: with and without access point migration for the publisher. We have observed that the minimum time to remove the messages from our queue is of 50 *ms* on average. Thus, we cannot use a period of time smaller than this value. We therefore used 100 *ms* as the minimum period between consecutive published messages. We also found that the publisher saturates when using a publish rate of 100 *msg/s*, using a QoS level of 1; in this case there is not enough time to exchange the confirmation messages between the entities involved.

V. Experimental Results

This section presents the experimental results using a scenario based on a wireless producer node moving between two different sub-networks while communication is ongoing. In each sub network the producer gets a different IP address. The tests were repeated 100 times for each combination of inter-message period and message size. The data message size ranged between 0.5 *KBytes* and 6 *KBytes*, and the intervals of time between consecutive messages where of 100, 500, and 1000 *ms*. The duration of each test was of about 60 *seconds*, enough time to observe the behavior of a mobile client moving along these sub-networks.

The MQTT broker was deployed on a server with an AMD 8-core processor and 16*GBytes* of RAM. The mobile client had an Atom N450 processor and 1*GByte* of RAM memory. The tests were run on a dedicated network, without external traffic.

A. Behaviour during access point transition

When the communication link is stable and reliable, the jitter values for each message were of just a few milliseconds, very close to zero. In the mobility case, when a producer is migrating from one access point to another and changes its IP address, the delay jitter had a considerable value, in the order of tens of seconds.

In order to understand such event a just of Figure 3, shows the typical jitter behavior for each message received by the subscriber. This value were obtained when producing and sending messages of different size during 60 *seconds* with different inter-message periods. We can see a positive peak corresponding to the hand-off time until a new connection between the different APs is again achieved. The computed value of the delay jitter on transitions was in all the cases of: 35.8 ± 0.03 *seconds*.

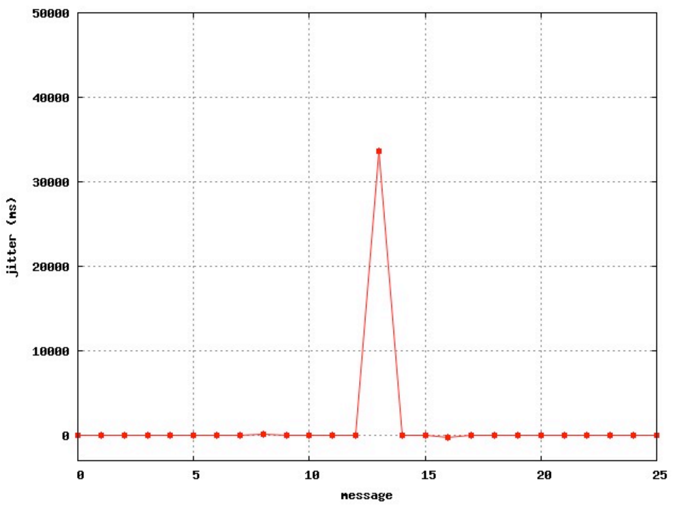

Fig. 3. Jitter values for received MQTT messages as the producer/publisher move between different access points.

B. Jitter analysis

The messages were produced at a constant rate and were published depending of the connection with the MQTT message-broker. These messages reach the subscriber with different delays depending on network conditions. Taking the difference of the arrival timestamps using *equation* (1), the jitter values of each message was typically of only a few milliseconds. The maximum jitter value in our tests occurs as a consequence of a disconnection of the publisher node. Notice that in our tests the network had no external traffic, and the workloads used do not saturate the system.

Using the Cumulative Distribution Function (CDF) on the set of experimental data, we have analyzed the behavior of the message jitter focusing on the instant when the producer/publisher makes an AP migration. Figure 4, shows the CDF for the jitter using a message production period of 100*ms*. With all our evaluated message sizes (from 512 *Bytes* to 6 *Kbytes*) the observations present a self-similar behavior.

We have observed that the jitter value is independent both from the message size and from the inter-message publishing period, being concentrated on values close to 36 *seconds*, without significant variation.

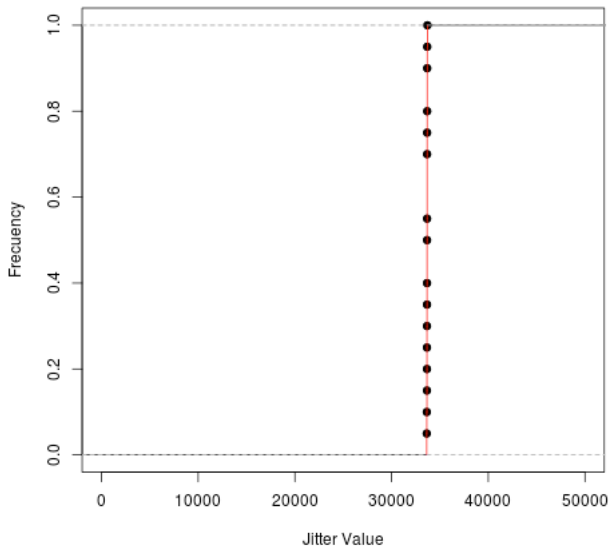

Fig. 4. Cumulative Distribution Function of the maximum jitter value with a different inter-message production periods and different message sizes too.

To study the jitter evolution as a function of the message size and the publishing period, we have used the 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. In Figure 5 we represent the jitter mode values as a function of message size.

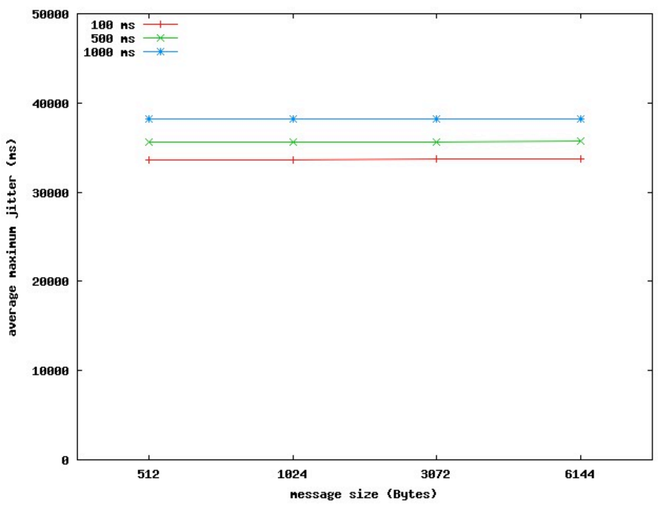

Fig. 5. Evolution of maximum jitter as a function of the message size.

Overall, we find that there is essentially a same behavior, independently of the message size (from 512 *Bytes* up to 6 *KBytes*) and the inter-message publishing period (from 100 *ms* to 1 *s*). After a disconnection and a connection re-establishment, the handover time of a user moves through the area covered by a Wi-Fi hotspot is less than 36 *seconds*. This actual handover time includes factors such as the time to detect another access point and sets up a secure link towards the MQTT message broker.

VI. System Modelling

To extend the results of the empirical evaluation described in the previous Section, we modeled a more general scenario using queueing networks. Our model is shown in Figure 6; the model was implemented[3] using *SimPy*, a process-based discrete-event simulation framework based on standard Python.

We supposed a unique *Consumer* and *m Mobile Nodes,* each with *n* associated *Producers*. The *Producers* generate messages with an inter-delay that we supposed constant with value λP. Messages reach the *Consumer* from the *Mobile Nodes* through the *Network*, an unbounded queue server with an exponentially distributed mean service time μ_{NET}. The *Consumer* processes the incoming messages with a mean service time μ_C that we supposed exponentially distributed. Queues are modeled as being unbounded so that no message is lost.

In this work we supposed that the Producers and the Consumer were located geographically close (e.g., the same city), and also supposed the Consumer to be a fast server. Specifically we take values to $\mu_{NET} = 3ms$, and $\mu_C = 100\ \mu s$.

Node handovers, i.e., changes in the point of attachment, are independent (Bernoulli) events with probability P_{HO}; this parameter allows us to tune the number of handover processes that take place in a test. The mobile nodes service time μ_N, can be either proportional to the message size *s* (i.e., equal to s*8.0/T_M , where T_M is the maximum network throughput, experimentally obtained with the testbed and set to 20Mbps) or to the handover delay (d_{HO}), which again was experimentally obtained with the testbed and set to 35.0*s*.

The main objective of this model was to evaluate the stability of our proposal when considering the complications imposed by mobility to mobile nodes. In Figure 7 we show end-to-end delay of messages with a fixed mobility pattern while increasing the message generation frequency. We considered a scenario with 100 mobile nodes, each with 3 producers, and each producer sending 1000 messages. We varied λP in the range [15.0, 30.0, 60.0, 90.0, and 120.0] *seconds*. Parameter P_{HO}=0.01 produced an average of 3000 handovers over a simulation period of about 4 hours (i.e., about one handover every 5 *seconds*); a total of 300000 messages were generated.

Figure 7 shows the 95 percentile (yellow lines with stars), and the median (red lines with triangles), the straight lines are relative to a message size of 1*MB*, while the dashed lines are relative to a message size of 512*B*.

[3] The code is available to interested readers upon request.

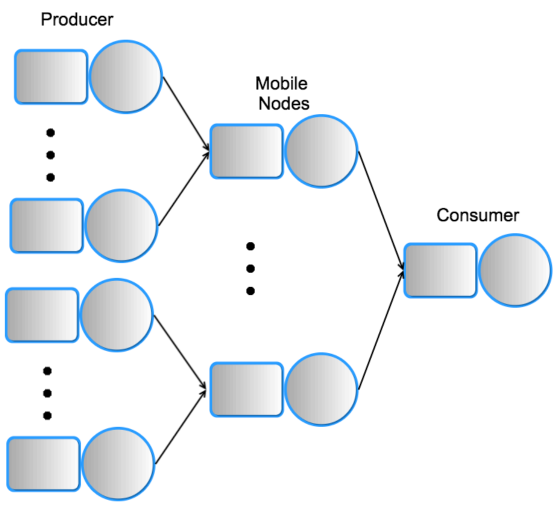

Fig. 6. Two states model of the mobile node behavior.

As we can see, our approach is quite stable when varying the data generation frequency and the message size, even with a high handover rate. At the lowest frequency used, i.e., below 30.0*s*, the system showed the first signs of saturation, with end-to-end delays of up to 20 *seconds*. In general the upper limit for the 95percentile was less that 1*s* for a message size of 512*B*, and less than 2*s* for a message size of 1*MB*.

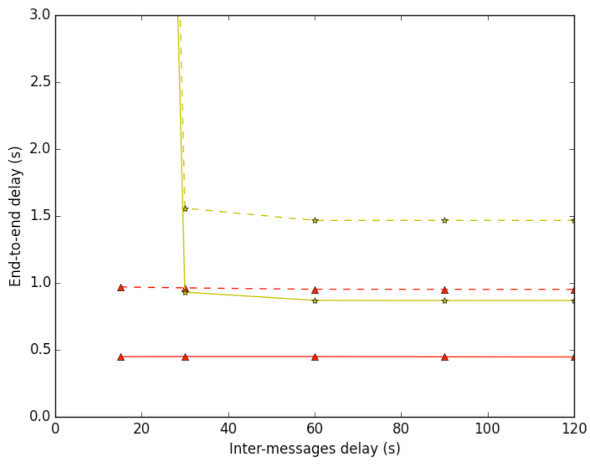

Fig. 7. Two-state model of the mobile node behavior.

To more precisely evaluate the behavior inside a mobile node, we considered a set-up with a single mobile node, but with a growing number of sources (in the range [1, 100]) each producer sending 1000 messages at a rate λP=60s. The other parameters where kept the same as before, again considering a message size of either 512*B* or 1*MB*.

Figure 8 shows the result of the evaluation; the blue bars indicate the mean value and the red bars the 99 percentile. The results obtained where similar for the two message sizes considered. As can be seen, the mean queue length is approximately equal to half the number of sources, while the

99-percentile is basically equal to the double of the number of sources. This is an important result for dimensioning the number of sources for the mobile node, and to calculate the required amount of buffers in the mobile node as a function of the number of sources and the size of the messages. Basically the relation would be:

$$\text{buffer size } (\textit{bytes}) = 2*s*n \tag{2}$$

in order to limit the number of messages lost due to buffer overflow. If the managed data is not critical, we could even reduce the amount of resources to:

$$\text{buffer size } (\textit{bytes}) = s*n/2 \tag{3}$$

that would allow to handle, on average, half of the produced messages, while significantly reducing the buffer size required.

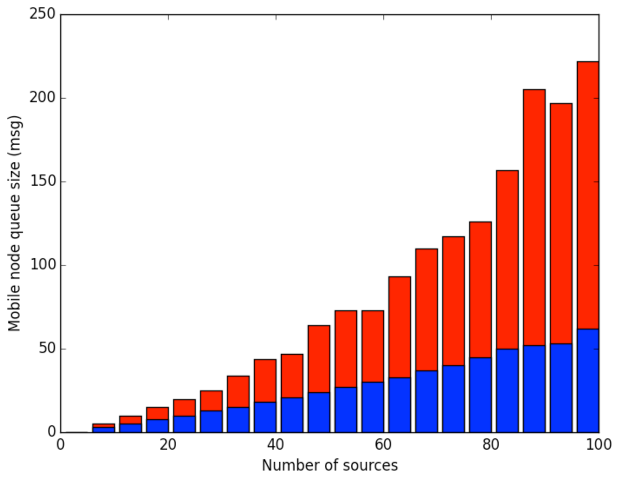

Fig. 8. Mobile node queue size. The blue bars indicate the mean value and the red bars the 99 percentile.

VII. Conclusions

In this work we focused on providing a solution to adapt MQTT to mobile scenarios. The advantage of our solution is that developers do not have to explicitly consider the changes in the point of attachment to the network without requiring network support through protocols like MobileIP or LISP.

We described our proposal based on intermediate buffering, and evaluated it in various scenarios where the publisher node suffers a handover process due to mobility. We measured jitter variability and information loss, with a simple workload model of one *producer/publisher* node and one subscriber node, in an extended wireless network. We have observed that this approach introduces a mean jitter value that ranges from 35 to 38 *seconds*.

We proved that our solution guarantees that there is no information loss during the hand-off of the *producer/publisher* node, thus making a messaging system based on MQTT protocol robust, and able to guarantee message delivery without losses in the presence of publisher node mobility. Messages losses would be present only when roaming time tends to infinite, a situation prone to cause memory leaks and the system buffer capacity to be overloaded.

To extend the results of the empirical evaluation we modeled a more general scenario using queueing networks. Our model was implemented using *SimPy*, a process-based discrete-event simulation framework based on standard Python. Through the model we obtained important results for dimensioning the number of sources transmitting towards the mobile node, and to calculate the required amount of buffers in the mobile node as a function of the number of sources and the message size. Basically, the relation would be: buffer size (*bytes*) = 2**s***n* in order to limit the number of messages lost due to buffer overflow. If the managed data is not critical, we could even reduce the amount of resource to a value: buffer size (*bytes*) = *s***n*/2, which allows handling half of the produced messages on average, although reducing buffer requirements by 75%.

References

[1] R. Cohn, R. Coppen, A. Banks, and R. Gupta, "MQTT Version 3.1.1, Specification URIs," Available: http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/mqtt-v3.1.1.html (accessed on Feb 21, 2015).

[2] "Machine to Machine (M2M) Communication Study Report," IEEE C 80216 10_0002r7, May, 2010. (accessed on Feb 21, 2015).

[3] Internet Engineering Task Force (IETF), "The Constrained Application Protocol (CoAP)," Available: http://tools.ietf.org/html/rfc7252, 2014.

[4] OMA Lightweight Machine to Machine Protocol v1,0 "http://technical.openmobilealliance.org/Technical/release_program/lightweightM2M_v1_0.asp" (accessed on Mar 10, 2015).

[5] C. Perkins, "Mobile IP", IEEE Comm., Vol. 35, No. 5, 1997, pp.84-99.

[6] David Meyer, "The Locator Identifier Separation Protocol (LISP)", in "The Internet Protocol Journal", Vol. 11 No. 1, March 2008, pp. 23-36.

[7] M. Collina, M. Bartolucci, A. Vanelli-Coralli, and G.E. Corazza, "Internet of Things application layer protocol analysis over error and delay prone links," Advanced Satellite Multimedia Systems Conference and the 13th Signal Processing for Space Communications Workshop (ASMS/SPSC), pp.398,404, 8-10 Sept. 2014.

[8] Ponte Eclipse Project, "Connecting Things to Developers," Available: http://eclipse.org/ponte, 2015 (accessed on Feb 21, 2015).

[9] E. Garcia Davis, A. Calveras, and I. Demirkol, "Improving packet delivery performance of publish/subscribe protocols in wireless sensor networks," Sensors 2013, 13, 648-680; doi:10.3390/s130100648.

[10] W. Chen, R. Gupta, V. Lampkin, D. Robertson, and N. Subrahmanyam, "Responsive Mobile User Experience Using MQTT and IBM MessageSight," IBM Corp., 2014.

[11] A. Jara, and A. Skarmeta, "Mobility support for the small and smart Future Internet devices," Available: http://www.iab.org/wp-content/IAB-uploads/2011/03/Jara.pdf, 2011, (accessed on Feb 21, 2015.

[12] S. Lee, H. Kim, D. K. Hong, and H. Ju, "Correlation analysis of MQTT loss and delay according to QoS level," International Conference on Information Networking," pp. 714-717, 2013.

[13] J. Luzuriaga, M. Perez, P. Boronat, J. C. Cano, C. Calafate, and P. Manzoni, "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.

[14] Eclipse.org., "Paho is an iot.eclipse.org project," Available: http://eclipse.org/paho/, 2014 (accessed on Feb 21, 2015).