Comparison of two lightweight protocols for smartphone-based sensing

Niccolò De Caro, Walter Colitti, Kris Steenhaut
Dept. ETRO-IRIS
Vrije Universiteit Brussel
Brussels, Belgium
{ndecaro, wcolitti, ksteenha}@etro.vub.ac.be

Giuseppe Mangino, Gianluca Reali
Dept. DIEI
Università di Perugia
Perugia, Italy
{giuseppe.mangino, gianluca.reali}@unipg.it

***Abstract*—Smartphones are equipped with numerous sensors and have become sophisticated sensing platforms. However, several sensing applications running on a smartphone can degrade the device performance. This can be overcome by using lightweight application protocols which improve the smartphone performance in terms of bandwidth consumption, battery lifetime and communication latency. This work focuses on two emerging application protocols: the Message Queuing Telemetry Transport (MQTT) and the Constrained Application Protocol (CoAP). Although both protocols have been designed for highly constrained environments such as sensors, they are also appropriate to be adopted in smartphone applications. We provide a qualitative and quantitative comparison between MQTT and CoAP when used as smartphone application protocols and we give preliminary indications on the application scenarios in which either protocol should be adopted. While MQTT has already been adopted in smartphone applications, CoAP is relatively new and has up to now mainly been considered for sensors and actuators. Our comparison shows that CoAP can be a valid alternative to MQTT for certain application scenarios.**

Keywords—smartphone, crowdsensing, application protocols, publish/subscribe, CoAP, MQTT

I. INTRODUCTION

With the integration of several types of embedded sensors, increased computing capability and programmability, smartphones have become a viable and sophisticated sensing platform which goes far beyond the normal telephony function. Modern smartphones include GPS, camera, microphone, gyroscope, accelerometer, proximity sensor, ambient light and compass. Humidity and temperature sensors have also started to appear in recently launched platforms. Integrated into Wireless Sensor and Actuator Networks (WSANs), smartphones can give a large scale mobile dimension to the internet of Things (IoT) [5].

The addition of sensing functionalities has posed challenges on the smartphone performance. A smartphone running several applications, which frequently collect sensor data, require high computational capabilities and can hamper the regular telephone functionalities [8]. The continuous improvement of the computational capabilities is not sufficient and requires an effort also on the optimization of the application protocols. Protocols like Hypertext Transfer Protocol (HTTP), based on a the request-response paradigm, and Extensible Messaging and Presence Protocol (XMPP), an instant messaging protocol which can be extended to support pub/sub mechanisms, have not been designed for pervasive networks and can cause performance degradation in terms of bandwidth usage and battery lifetime [9].

The lack of optimized application protocols for sensors, actuators and smartphones has given rise to the design of new standard lightweight application protocols. Two of them have recently gained momentum: the Message Queuing Telemetry Transport (MQTT) [4], designed by IBM and now under a standardization process, and the Constrained Application Protocol (CoAP) [7], recently designed by the internet Engineering Task Force (IETF). The two protocols have different design principles: MQTT is a very optimized messaging application protocol designed only for pub/sub based systems, while CoAP is an application protocol designed to provide constrained environments with HTTP like (e.g. request-response) web transfer mechanisms. In addition, CoAP has been equipped with a built in registration mechanism which makes the protocol also appropriate to pub/sub applications. While MQTT has already been adopted in smartphone applications (e.g. Facebook), CoAP has up to now mainly been considered in WSANs.

This work provides a preliminary comparison between MQTT and CoAP when used as application protocols for smartphones. The aim of the comparison is to analyze the two protocols from a qualitative as well as quantitative perspective and to give preliminary indications on the application scenarios in which either protocol should be adopted. The comparison also shows that CoAP, which to the best of our knowledge has never been used in smartphone applications, can be a viable alternative to MQTT for certain application scenarios.

The rest of the paper is organized as follows. Section II discusses on the related work. Section III provides a description of MQTT and CoAP. Section IV provides a qualitative comparison of the two protocols. Section V reports on the experimental study and discusses. Section VI discusses the results and draws the conclusions.

II. RELATED WORK

The comparison of lightweight communication protocols has received attention by recent literature work. In [11], Davis

978-1-4799-3206-1/13/$31.00 ©2013 IEEE

et al. reported on the performance comparison between MQTT-S [3], a specification of MQTT adapted to the characteristics of low-power WSN environment, and CoAP. Our previous work in [9] presented the comparison between CoAP and HTTP in WSNs and demonstrated how CoAP improves the performance of constrained devices such as sensors and actuators. In [12], high level features (such as messaging capabilities, security, reliability, availability of implementations…) of MQTT and Advanced Message Queuing Protocol (AMQP) are evaluated and compared. A similar approach has been followed in [13] in order to compare MQTT and HTTP features.

All the aforementioned comparisons of application protocols have not considered the use of smartphone but have only been limited in using lightweight application protocols in sensors and actuators. To the best of our knowledge, smartphone platforms have only been considered in [10], where Vergara et al describe a preliminary experimental evaluation of MQTT against HTTP. The comparison shows that the use of MQTT in Android devices decreases the energy consumption compared to HTTP in location based applications.

To the best of our knowledge, the comparison between CoAP and MQTT has only been addressed in [6]. The comparison shows that CoAP lead to a lower bandwidth utilization as a consequence of the use of UDP instead of TCP as transport protocol. However, the comparison only considers the case in which the two protocols run on constrained battery operated gateways. They do not consider neither sensors and actuators nor smartphones. .

III. COAP AND MQTT

A. Constrained Application Protocol

Due to the observed unsuitability of HTTP in resource constrained environments, the IETF CoRE Working Group has defined CoAP, a web transfer protocol optimized for resource constrained networks [15]. It consists of the re-design of a subset of HTTP functions by taking into account the low processing power and energy consumption constraints of embedded devices, such as sensor motes. In addition, different mechanisms have been modified and some new functions have been added as well. Since TCP flow control mechanism is not appropriate for devices with limited resources and its overhead is deemed too high for short-lived transactions, CoAP has been built on top of UDP, which has lower overhead and also provides multicast support.

CoAP is organized in two layers. The Transaction layer handles single message exchange between end points. The Request/Response layer is responsible of both request/response transmission and resource management. The dual layer approach allows CoAP to provide reliability mechanisms and basic congestion control even without the use of TCP as transport protocol. In fact, a message can be labeled as Confirmable (CON), which implies that it is retransmitted up to a maximum number of times by using a default response timeout and exponential back-off algorithm to schedule retransmissions, until the recipient sends either an Acknowledgement (ACK) message, or a Non Confirmable (NON) back to the transmitter. In addition, the two-layer approach enables asynchronous communication which is a key requirement for the web applications where web servers are not able to handle requests immediately. CoAP includes a built-in pub/sub mechanism (i.e., Observe option), which allows a client both to subscribe to resources hosted by a CoAP server and to receive notifications upon a change of the resource state. The protocol also includes a technique for discovering and advertising resource descriptions.

One of the major design goals of CoAP was to minimize the message overhead and limit the packet fragmentation. To this end, CoAP includes a short fixed-length compact binary header of 4 bytes followed by compact binary options. The length of the header of a typical request packet typically ranges between 10 and 20 bytes. The protocol stack of CoAP and its two-layer structure are illustrated in Figure 1, along with the stacks of HTTP and MQTT.

B. Message Queuing Telemetry Transport

MQTT was introduced by IBM in 1999. It is a lightweight protocol suitable for devices with limited processing and memory capabilities, in order to send data over low-bandwidth networks. MQTT is currently submitted for standardization at the Organization for the Advancement of Structured Information Standards (OASIS), from in March 2013. Today it is an open and royalty-free protocol.

MQTT is based on the pub/sub model, where multiple clients can establish a connection (with a *broker* in order to: i) subscribe to specific topics of interest and thereby receive the published messages related to these topics; ii) publish messages to topics. The former clients are called *subscribers*, the latter ones are called *publishers*. The connection is established by sending a CONNECT message, which needs to be acknowledged (through a CONNACK message). Clients can send either PUBLISH or SUBSCRIBE messages.

MQTT makes use of TCP. This means that although the MQTT protocol adds a very small header (i.e., only two bytes) to a message, TCP introduces a considerable overhead to provide reliable, ordered, and error-checked message delivery. On top of this, MQTT defines three Quality of Service (QoS) levels for message delivery. With QoS 0, messages are delivered at most once (MQTT is as reliable as TCP); with QoS 1, messages are delivered at least once by means of acknowledgments (PUBACK or SUBACK); with QoS 2, messages are delivered exactly once. In this latter case, the publisher first sends a PUBLISH, then it receives a PUBACK, finally it sends a PUBREL and it receives a PUBCOMP, which completes the transaction. Message delivery retry is performed a configurable number of times, after a timeout which increases across the retries.

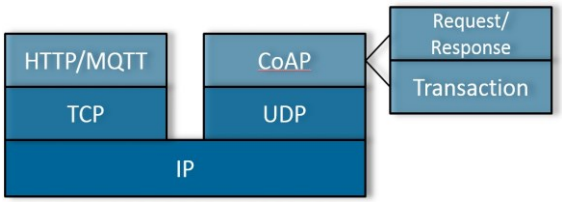

Fig. 1. Difference between the CoAP, MQTT and HTTP stacks

In order to keep the connection open, some additional traffic must be sent. The client must send at least one message within a time interval (Keep Alive time). If any data is sent during this interval, the client sends a PINGREQ which must be replied by a PINGRESP message. Another MQTT feature concerns retained messages: the Broker keeps messages even after they are sent to subscribers. This means that when a new subscription for the same topic happens, retained messages can be sent to the new subscriber. Furthermore, a connection can be set as durable by subscribers. This way, when a subscriber disconnects, its subscription is not cancelled, and any further messages is stored until it reconnects. Otherwise, all subscriptions are removed when the subscriber disconnects. Finally, MQTT provides support of "wills". A will is a message specified by the publisher when it connects to the broker. Such messages have to be sent to subscribers in case the publisher unexpectedly disconnect.

IV. QUALITATIVE COMPARISON

A. Communication pattern

The two protocols are inspired by different communication paradigms. MQTT is a pub/sub protocol, whilst CoAP is a RESTful protocol with built-in support of a pub/sub mechanisms through the Observe option. With this option, a CoAP client registers its interest on a resource by issuing a GET request to the CoAP server. Whenever the state of a resource changes, or at regular time intervals, the server notifies the client [14]. This twofold nature of CoAP makes it a more flexible solution for application developers. Being it RESTful and using a subset of the HTTP verbs to manipulate resources, CoAP enables a seamless integration with most web applications through HTTP-CoAP proxies. In addition, the Observe feature allows the development of efficient applications. On the other hand, this comes at the price of a more complex integration with the web applications, since HTTP does not support the pub/sub model.

B. Transport protocols and communication security

MQTT relies on TCP, whereas CoAP relies on UDP. From the developer point of view it follows that MQTT and CoAP inherit from TCP and UDP their advantages and disadvantages in terms of functionality and performance. Hence, we can expect MQTT to be more reliable than CoAP, at the price of a higher overhead. Communication security is not directly addressed by these protocols. However, encryption through the network can be achieved by using Secure Sockets Layer (SSL) for MQTT, or Datagram Transport Layer Security (DTLS) if CoAP is used. The most recent version of the MQTT specifications include a basic user security, which allows sending user name and password with a CONNECT packet.

C. Reliability and congestion control

MQTT provides three QoS levels (QoS 0, QoS 1 and QoS 2), whereas the CoAP reliability mechanism is based on NON/CON messages. MQTT QoS 0 and CoAP NON messages do not guarantee reliability, and MQTT QoS 1 and CoAP CON guarantee ACK-based reliability. MQTT QoS 2 level, which guarantees that duplicated messages are not delivered to the receiver, does not correspond to any similar QoS level in CoAP. It provides more flexibility to developers and makes MQTT suitable for applications which do not tolerate duplicate messages. With regards to congestion control, MQTT leverages on TCP built-in mechanisms. Moreover, both protocols provide basic congestion control at application layer, by retransmitting the messages that are not acknowledged by a back-off interval.

D. Design orientation

CoAP is document centric, meaning that it defines content format. On the contrary, MQTT is data centric, since it is agnostic to the content, transmitted bite-wise. This feature makes MQTT preferable for applications where very small data updates need to be frequently sent. However, CoAP includes short identifiers of media types, which enables the use of different content types with a minimal overhead.

E. Caching, retained messages, durable connections, wills

CoAP is able to cache responses in order to satisfy future requests by reusing the content of a prior response. Caching is managed by including information about validity and freshness in CoAP responses. It may be useful to deploy a cache within an intermediate point, such as an HTTP-CoAP proxy, with the aim of reducing response time and network bandwidth consumption. MQTT does not provide this feature, but it provides three additional functions that are not present in CoAP: retained messages, durable connections and wills.

F. Fragmentation

CoAP includes a mechanism called Blockwise. It provides a solution to transfer large data sets in a block-wise fashion. This feature enables transferring larger protocol data units than those typical of lower layers. On the contrary, MQTT does not allow message fragmentation, which hampers the transmission of large messages in constrained environments.

G. Discovery

MQTT does not include any discovery mechanism, which are present in MQTT-S. Differently, CoAP includes its own discovery mechanism (RFC 6690), based on two approaches recently defined by IETF for HTTP: the well-known resource path */.well-known/scheme* (RFC 5785), which makes use of a "well known location" or storing resources to be easily located, and the *Web Linking* (RFC 5988), which defines a framework for typed links.

H. Multicast

Multicast communication is intrinsically supported by MQTT. On the other hand, as CoAP runs over UDP, it supports the use of multicast IP destination addresses. However, at the time of writing, multicast messages cannot be carried in DTLS, which means that the secure communication cannot be achieved in a one-to-many scenario.

V. PERFORMANCE COMPARISON

In this section we analyse and compare the performance of MQTT and CoAP in terms of bandwidth usage, latency and

packet loss. To this end, we consider the simple scenario depicted in Figure 2. In the typical crowdsensing scenario, a smartphone user provides some sensed data (e.g., GPS location, noise level) to an application deployed locally or in the cloud, which in turn provides the requesting users and applications with these data.

When MQTT is used, the sensing smartphone acts as the information Publisher; the Broker is implemented by an application running either locally (reachable via WiFi) or in the cloud (reachable via mobile technologies such as 3G); the Subscriber can be either a user interested to the published data, or another application which has the role of aggregating and processing raw data in order to provide richer information to a final user. Clearly, more than one Publisher and more than one Subscriber can be deployed. In the scenario where CoAP is used, the sensing smartphone executes a CoAP Server, which provides an HTTP-based web service in the cloud with data. Such web service efficiently interfaces with the sensing device through a CoAP Client. On the other side, the web service is capable of satisfying HTTP requests coming from interested users or other applications. Again, this architecture can be extended to accommodate multiple users and applications and sensing smartphones. In this paper, in order to fairly compare MQTT and CoAP, we limited the MQTT evaluation to the communication between the Broker and the Publisher.

A. Test-bed description

In the test-bed used in our experiments, CoAP Server and Client are based on a Java implementation called Californium[1]. The MQTT Publisher is based on the MQTT Service[2] Android implementation, and the MQTT Broker is based on the Mosquitto[3] Broker v1.2 software. Both MQTT implementations leverage on Eclipse's Paho Library[4]. MQTT Publisher and CoAP Server run on a Samsung Galaxy S Plus smartphone with Android Jelly Bean 4.2.2, equipped with a 3.7 V, 1650 mAh battery. MQTT Broker and CoAP Client run on a Windows 7 OS laptop. The laptop and the smartphone join a wireless network through a WiFi Access Point.

B. Bandwidth usage

With the aim of fairly comparing the bandwidth usage of the two protocols, we have compared MQTT with the pub/sub feature of CoAP, i.e. the Observe option.

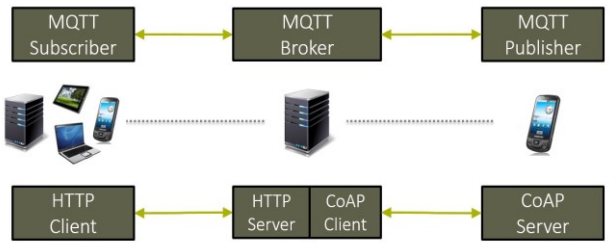

Fig. 2. Crowdsensing application scenario

[1] https://github.com/mkovatsc/Californium
[2] https://github.com/dirkmoors/MqttService
[3] http://mosquitto.org
[4] http://eclipse.org/paho/

More precisely, the CoAP Client registers with a resource called "foo" on the CoAP Server. Thereafter, the CoAP Server starts sending periodical notifications about the resource status. In the MQTT case, the Publisher establishes a connection with the Broker by setting a Keep Alive Time. After that, it periodically sends PUBLISH messages to the Broker with the topic "foo" (no subscription is required, here).

We have compared the two protocols in two scenarios: no-reliable scenario (MQTT QoS 0 vs. CoAP NON), and scenario with reliability (MQTT QoS 1 or 2 vs. CoAP CON). In the former, the transaction makes use of a single packet: CoAP NON or MQTT PUBLISH. In the scenario with reliability, a CoAP transaction consists of a CON and an ACK message; an MQTT QoS 1 transaction consists of two messages (PUBLISH and PUBACK), and MQTT QoS 2 transaction consists of four messages (PUBLISH, PUBACK, PUBREL, PUBCOMP). Results have been achieved by computing the transferred bytes, assuming that no packet losses happen, and by using IEEE 802.11b as link layer protocol.

Figures 3 and 4 show the per-layer bandwidth usage by means of bytes transferred between the CoAP Server and the CoAP Client, and between the MQTT Publisher and Broker, necessary to complete a transaction. It can be observed that CoAP packets have a slightly smaller total size in spite of the slightly larger application layer packet size, due to the larger TCP overhead. On the other hand, MQTT QoS 2 allows avoiding duplicated messages (a functionality which is missing in CoAP) at the price of a roughly doubled bandwidth usage.

Figures 5 and 6 show, in both the no-reliable scenario and in the scenario with reliability, the bandwidth used by the two protocols on the long run. In the CoAP case, the observation relationship is established and maintained for 30 minutes and, after that, it automatically expires. In the MQTT case, the MQTT connection is open, kept active for 30 minutes and then closed. In both cases, during this time interval, transactions happen every 30 seconds. In the no-reliable scenario, the CoAP Server needs to guarantee the CoAP Client is still connected by sending CON notifications every 5 minutes.

In the scenario with reliability, no action is taken in the CoAP case to maintain the observation relationship. In Figure 5, we can observe a slightly lower bandwidth usage by CoAP, than MQTT, which is due to a lighter registration/connection setup mechanism, although limited by the impact of the periodical CoAP CON notifications. In the scenario with reliability (Figure 6), CoAP reveals a bandwidth usage which is more than 15% lower than MQTT with QoS 1. Here, the setup phase has a very low impact on the results. As expected, MQTT QoS 2 shows a bandwidth usage which is roughly doubled.

C. Latency

We compared the Round Trip Time (RTT) values of CoAP and MQTT transactions. When CoAP is used, the RTT is the time elapsed from the transmission of a CON message to the reception of the related ACK. Similarly, in the MQTT case we consider the time for the PUBLISH and PUBACK messages to be exchanged, assuming the connection open.

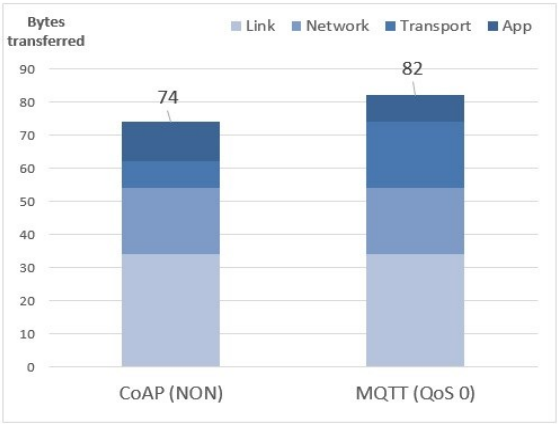

Fig. 3. Per-layer bandwidth usage (single transaction, no reliability)

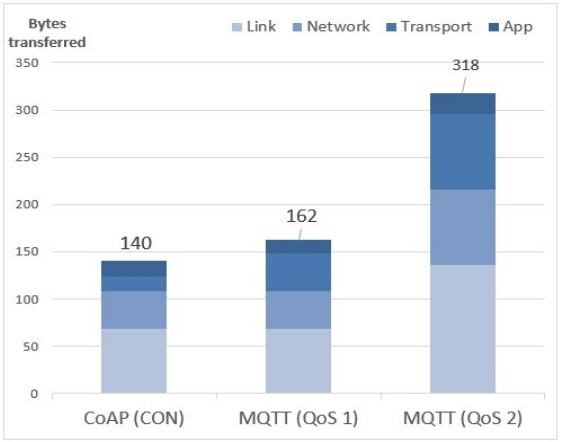

Fig. 4. Per-layer bandwidth usage (single transaction, with reliability)

To evaluate the RTT, only the scenario with reliability (CoAP CON vs. MQTT QoS 1) has been taken into account, where ACKs are sent back to the sender. The test has been repeated 150 times for each case, in the real test environment described in Section V.A. Figure 7 shows that the average CoAP RTT is 127 ms, more than 20% shorter than MQTT. This result is not due to different impact of performance of UDP and TCP. Indeed, we measured the RTT for UDP and TCP by using a packet of fixed length, and we obtained comparable results. Therefore, we could deduce that the latency improvement in the CoAP case is caused by the shorter length of the CoAP packet. The confidence interval has been set to 95%.

D. Packet loss

In order to emulate packet losses in the network, we introduced in the test-bed the NetEM (Network EMulation) tool[5], which provides network emulation functionality for testing protocols. In our experiment, NetEM has been configured on the same laptop where the MQTT Broker or CoAP Client are executed. We had to run the test in Ubuntu OS, since NetEM is a Linux tool. We configured NetEM so as to randomly drop the 20% of the incoming packets.

[5]http://www.linuxfoundation.org/collaborate/workgroups/networking/netem

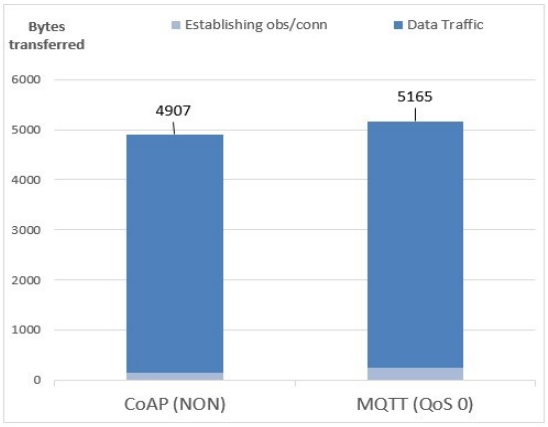

Fig. 5. Per-traffic type bandwidth usage (complete test, no reliability)

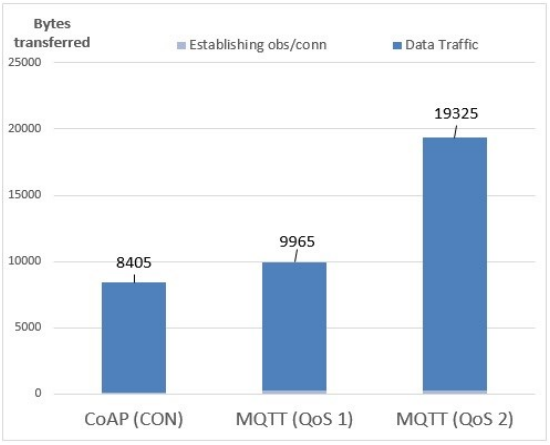

Fig. 6. Per-traffic type bandwidth usage (complete test, with reliability)

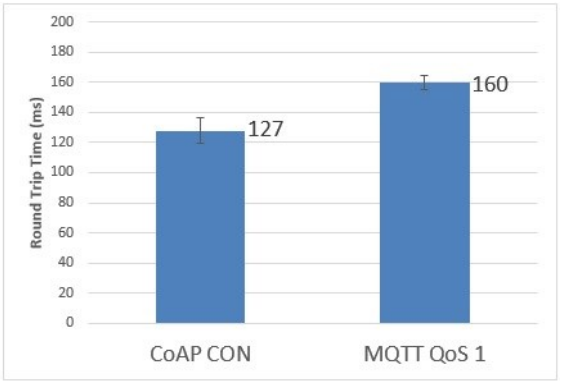

Fig. 7. Comparison of CoAP CON and MQTT QoS 1 Round Trip Time

The metric used here is the Packet Received Ratio (PRR), computed as the ratio between the total number of messages received by the CoAP Client or MQTT Broker, and the total number of notification messages generated by the CoAP Server or MQTT Publisher. The packet sniffer Wireshark has been used to measure the PRR. Table I reports the PRR in the scenario with reliability (CoAP CON vs. MQTT QoS 1), given two different values of interval between notifications: 30s and 5s. Each experiment, lasting 30 minutes, has been executed 5 times. It was assumed that the CoAP registration has been

completed, or the MQTT connection was open, at the beginning of the experiment. The maximum number of retransmission has been set to 4, and the response timeout has been set to 2 seconds. The results reported in Table 1 highlight that, if the interval between notifications is too short, the CoAP reliability mechanism does not compensate the losses introduced, but it even introduces additional losses due to its inefficient congestion control. On the other hand, results are more encouraging for both protocols when notifications are more sporadic. As expected, in both experiments MQTT exhibits better performance, mainly due to the TCP reliability.

TABLE I. AVERAGE PACKET RECEIVED RATIO (%)

Notification interval	**Avg. Packet Received Ratio, PRR (%)**	
	CoAP CON	**MQTT QoS 1**
5 seconds	65	86
30 seconds	96	100

VI. CONCLUSION AND FUTURE WORK

In this work, we have focused on CoAP and MQTT, two lightweight application protocols capable of satisfying most requirements of smartphone-based crowdsensing applications in terms of functionalities and performance. Subsequently, we have compared and discussed in detail these two protocols, firstly from a qualitative perspective and then from a quantitative one. The qualitative comparison suggests that MQTT is more appropriate for applications requiring advanced functionalities, such as messages persistence, wills and "exactly-once" delivery. Furthermore, CoAP limits the security support to unicast communications, which makes MQTT a preferable solution when secure multicast is a strong requirement.

On the other hand, our preliminary performance analysis shows that CoAP achieves better results both in terms of bandwidth usage and round trip time. This, combined with the CoAP caching feature, makes CoAP a more appropriate choice for the development of efficient applications having the goal of reducing both network utilization and device resource usage.

In terms of reliability, MQTT performs better as a consequence of its more sophisticated reliability and congestion control mechanisms. However, a significant difference is only observed when data have to be exchanged very frequently. For applications which do not require very high data transmission frequency, the difference between the reliability in CoAP and MQTT decreases. With regards to interoperability, CoAP fragmentation feature enables the efficient transmission of large messages in constrained networks, which facilitates the integration between smartphones and WSNs. Moreover, the RESTful nature of CoAP eases the interoperability between crowdsensing applications and other web services, hence making unnecessary the adoption of complex gateways like the one described in [2].

As future work, we intend to validate the preliminary results shown in this work into larger real world system in which the scalability can also play an important role in the choice of the application protocols.

ACKNOWLEDGMENTS

Part of the results described in this paper are obtained from the 6lowpan project. This project has been supported financially by the IWT – Flemish Agency for Innovation by Science and Technology under Tetra.

REFERENCES

[1] Ganti, R.K., Ye, F., Lei, H., Mobile crowdsensing: Current state and future challenges, IEEE Comm. Mag. 49(11), pp 32–39, 2011.

[2] M. Collina et al., Introducing the QEST broker: Scaling the IoT by bridging MQTT and REST, IEEE 23rd International Symposium on Personal Indoor and Mobile Radio Communications (PIMRC), 2012.

[3] MQTT For Sensor Networks (MQTT-S) Protocol Specification V. 1.2.

[4] MQTT V3.1 Protocol Specification.

[5] J. Burke et al., Participatory Sensing, Proc. Int'l Workshop World-Sensor-Web (WSW 06), ACM Press, pp 1–5, 2006.

[6] S. Bandyopadhyay, A. Bhattacharyya, Lightweight Internet protocols for web enablement of sensors using constrained gateway devices, International Conference on Computing, Networking and Communications (ICNC), 2013.

[7] Z. Shelby, K. Hartke, C. Bormann, Constrained Application Protocol (CoAP), Internet-Draft, draft-ietf-core-coap-18, 2013 (Work in progress).

[8] N. D. Lane, E. Miluzzo, Hong Lu, D. Peebles, T. Choudhury, A. T. Campbell, A Survey of Mobile Phone Sensing, IEEE Communications Magazine, Vol. 48, pp 140–150, 2010.

[9] W. Colitti, K. Steenhaut, N. De Caro, Integrating Wireless Sensor Networks with the Web. Proc. IP+SN, Chicago, IL, USA, 2011.

[10] E.J. Vergara, M. Prihodko, S. Nadjm-Tehrani, Mobile location sharing: an energy consumption study, e-Energy 2013, pp 289-290, 2013.

[11] E.G. Davis, A. Calveras, I. Demirkol, Improving Packet Delivery Performance of Publish/Subscribe Protocols in Wireless Sensor Networks, Sensors 13, No. 1, pp 648-680, 2013.

[12] StormMQ, A Comparison of AMQP and MQTT, White Paper.

[13] V. Lampkin et al., Building smarter planet solutions with MQTT and IBM WebSphere MQ telemetry, IBM, ITSO, 2012.

[14] K. Hartke, Observing Resources in CoAP, Internet-Draft, draft-ietf-coreobserve-07, 2012 (Work in progress).

[15] C. Bormann, A. P. Castellani, Z. Shelby, CoAP: An Application Protocol for Billions of Tiny Internet Nodes, IEEE Internet Computing, Vol. 16, No. 2, pp 62-67, 2012.

[16] R. Rana et al., Ear-Phone: An End-to-End Participatory Urban Noise Mapping System, IPSN '10, 2010.

[17] E. D'Hondt, M. Stevens, A. Jacobs, Participatory noise mapping works! An evaluation of participatory sensing as an alternative to standard techniques for environmental monitoring, Pervasive and Mobile Computing, Vol. 9, No. 5, pp 681-694, 2012.

[18] P. Mohan, V. Padmanabhan, R. Ramjee, Nericell, Rich monitoring of road and traffic conditions using mobile smartphones, ACM SenSys, pp 323–336, 2008.

[19] E. Miluzzo, N.D. Lane, B. Eisenmann, A.T. Campbell, CenceMe: Injecting Sensing Presence into Social Networking Applications, 2nd European Conf. Smart Sensing and Context, pp 1-28, 2007.

[20] J. G. Vieira, T. Galvão, J. Falcão e Cunha, P. M. Costa, and J. Pitt. Smart Mobile Sensing for Measuring Quality of Experience in Urban Public Transports, 2nd Workshop on Smart Mobile Applications, 2012.