Robustness of IoT Application Protocols to Network Impairments

Elizabeth Liri[1], **Prateek Kumar Singh**[2], **Abdulrahman BIN Rabiah**[1], **Koushik Kar**[2], **Kiran Makhijani**[3] and **K.K. Ramakrishnan**[1]

[1]University of California, Riverside, CA 92521, Email: eliri001@ucr.edu, abinr001@ucr.edu, kk@cs.ucr.edu
[2]Rensselaer Polytechnic Institute, Troy, NY 12180, Email: singhp6@rpi.edu, koushik@ecse.rpi.edu
[3]Huawei Technologies, Santa Clara, CA 95050, Email: Kiran.Makhijani@huawei.com

***Abstract*—Constrained Application Protocol (CoAP) and Message Queue Telemetry Transport (MQTT) are two IoT application layer protocols that are seeing increased attention and industry deployment. CoAP uses a request-response model and runs over UDP, while MQTT follows a publish-subscribe model running over TCP. For more constrained IoT devices, MQTT-Sensor Networks (MQTT-SN) provides a UDP-based transport between the sensor and an MQTT-SN gateway, while using TCP between that gateway and the MQTT broker. Quick UDP Internet Connections (QUIC) is a new protocol and although not originally designed for IoT devices, some design features such as reduced connection establishment time may be useful in an IoT environment. Each of these protocols seeks optimizations in features and implementation complexity based on application domains rather than having the full flexibility and adaptability of traditional transport protocols such as TCP.**

We investigate and analyze four protocols, namely, CoAP, MQTT, MQTT-SN and QUIC, to understand the overhead of obtaining data from an IoT device at a sink to potentially disseminate this data downstream. These constrained IoT devices often operate under challenging, varying network conditions, and it is important to understand the limitations of the protocols in such conditions. Thus, we evaluate the performance of these protocols under varying loss, delay and disruption conditions to identify the most effective environment for their operation and understand the limitations of their dynamic range. Results show that with non-confirmable CoAP a more adaptive wait timer is required; and a more streamlined QUIC protocol may be a potential alternative IoT protocol.

I. Introduction

The Internet of Things (IoT) enables sensors and other devices to interact with each other, the environment, and communicate over the Internet [1]. In the past few years, there has been an exponential growth in the use of IoT devices in many fields, including vehicular related applications with smart cars and health with e-monitoring. IoT devices have the ability to sense, actuate, process and communicate over the Internet either directly via a cellular network or via a gateway using other radio technologies for Internet access. When designing protocols for an IoT environment, several factors should be considered including device and traffic characteristics, and the operating environment. We focus on constrained devices which have limited resources such as power, memory and bandwidth.

With the constrained devices, maximizing lifetime of the node is critical, and one way to achieve this is by use of energy efficient protocols for data management and transmission. In order for constrained devices to connect and communicate via the Internet, three core requirements need to be met by the communication stack [2]. It must be low-powered (for energy constrained IoT devices). It must be highly reliable, incorporating error detection, congestion and flow control schemes where necessary, and finally it must be IP enabled, allowing for integration into the Internet.

Application layer protocols for IoT seek to optimize for the environment by simplifying the protocol: having fixed parameters and avoiding the complexity and overhead of general purpose protocols. This may, however, limit the dynamic range of their applicability. General purpose protocols, such as TCP, add complexity by including functionalities, such as in-order delivery, congestion and flow control schemes, so as to be applicable in a wide variety of application scenarios. It is important to understand the tradeoff between application specific protocols vs. the use of a general purpose protocol in terms of performance, robustness and energy use: *What are the limitations of such application layer protocols when the network conditions are variable and impaired, which do not satisfy the assumptions that go into the optimized application layer protocols?*

Our work here is geared towards quantitatively understanding the application layer protocols for constrained IoT devices – How they may operate over networks that have impairments that arise from using low power (e.g. LoWPANs) or having excessive packet loss. Our quantitative comparison looks at a wide set of IoT protocols like CoAP, MQTT, MQTT-SN and QUIC under varying network conditions such as loss, communication disruption and high delay. We perform careful measurement experiments in both an emulated and physical wireless environment that emulates typical IoT environments, using open source implementations of the various IoT protocols, and Raspberry Pis for end-points.

II. Related Work

There are several surveys available which *qualitatively* compare various IoT protocols like CoAP and MQTT [3], [4]. However, there is not much work done on the quantitative comparison of IoT protocols. The authors in [3] present a survey of IoT by focusing on enabling technologies, protocols and applications. They provide a horizontal view of the IoT and look at providing a summary of the most relevant protocols

978-1-5386-5541-2/18/$31.00 ©2018 IEEE

used in IoT together with current application challenges. In this work, they discuss CoAP, MQTT and Advanced Message Queuing Protocol (AMQP) as some of the IoT protocols. Atzori et al. [4] describes the primary communication technologies for both wired and wireless along with the elements of wireless sensor networks (WSNs). The survey presented in [5] specifically discusses the IoT scenarios for specialized clinical wireless devices using 6LoWPAN/IEEE 802.15.4, Bluetooth and NFC for mHealth and eHealth applications. In [6], multiple issues with the development and deployment of IoT devices are discussed, and an IoT architecture is also presented. IoT applications utilizing multiple enabling technologies like RFID are discussed in [7]. Some of the existing surveys discuss challenges specifically for sensor networks, e.g., [8].

Among quantitative studies, [9] presents the performance of CoAP and MQTT under a common middleware with different packet loss rates. However, [9] does not describe the effect of other network conditions like disruption, delay or protocol overhead on performance. In [10], comparison between the performance of CoAPs request-response mode and MQTT's resource-observe mode is presented with packet loss being considered as the only network condition. In [11], both of these protocols, CoAP and MQTT, are compared in a smartphone application environment. While the paper concludes that CoAPs bandwidth usage and round trip time are smaller than those of MQTT, the performance measurement is done only for a network with 20 % packet loss. To the best of our knowledge, our work is the only quantitative comparison for such a wide set of IoT protocols like CoAP, MQTT, MQTT-SN and QUIC under a wide range of network conditions. The performance comparison of QUIC against other IoT protocols is also a novel aspect of this study.

III. IoT Protocols

A. CoAP

The Constrained Application Protocol (CoAP) created by the IETF [12] is designed to interact with resource constrained devices, such as sensors, using a RESTful API. A key design goal with CoAP was to keep message overhead small and limit packet size. RFC 7252 [12] recommends that CoAP messages fit within a single IP datagram and assumes an IP MTU of 1280 bytes though more conservative limits may be used. Other features of CoAP include use of UDP as the underlying framing and binding protocol, asynchronous message exchanges and stateless HTTP mapping. CoAP seeks to keep parsing complexity low and requires two nodes for communication; a client (request originator) and a server (destination or origin server, usually the sensor or IoT device). CoAP uses four types of messages between nodes: *Confirmable*, *Non-confirmable*, *Reset* and *Acknowledgement* messages.

Both the client and server can utilize Confirmable messages independently although the specification recommends that they be matched. Every Confirmable message sent by an endpoint must be acknowledged by an ACK from the receiver. The ACK can be piggybacked with the data response if the data is immediately available, otherwise the server sends the ACK back first and later transmits the data response as a separate message. For Non-confirmable requests, no ACK is expected by the client. Furthermore, Reset messages are used as a response when the initial received message has an error or is missing context. The total number of packets for servicing a data request ranges from 2 (both request and response are non-confirmable) - 4 (request and response are confirmable, with separate ACKs).

Fig. 1 and 2 illustrate the CoAP call flow in the Confirmable mode under loss and delay. CoAP implements a simple stop and wait ARQ for reliability, including an exponential back-off for Confirmable messages and typically runs over UDP.

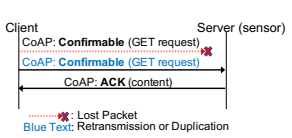

Fig. 1. Call Flow: CoAP under Loss in Confirmable Mode.

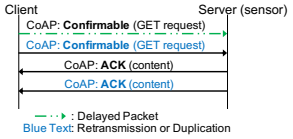

Fig. 2. Call Flow: CoAP with Delays in Confirmable Mode.

B. MQTT

MQTT [13] is a publish/subscribe protocol implementing one-to-one, one-to-many and many-to-many connections between IoT devices. The MQTT protocol requires three major components: *subscribers*, *publishers* and *brokers*. Subscribers are devices that subscribe for content being generated by the publishers, which are generally sensor/IoT devices. The subscribed content is identified by topic names and brokers act as intermediary devices, managing the list of subscribed topics and forwarding information from the publishers to the subscribers. The use of a broker is a key advantage of MQTT because it manages the subscriptions and also can handle retransmission of data to the subscriber when necessary. Another advantage is that the publisher can send new content whenever it is available, thus decoupling the temporal relationship between a node's interest and the publication of the information.

MQTT provides three levels of Quality of Service or reliability: fire-and-forget (QoS 0), deliver at least once (QoS 1) and deliver exactly once (QoS 2). QoS 2 messages are useful in scenarios where neither duplication nor loss is acceptable. The difference between QoS 0 and QoS 1 is that with QoS 1, the PUBLISH message from the sender, which contains the sensor data, requires a PUBLISH ACK back from the recipient. In order to provide the required QoS guarantee, the client and server must store session state for the entire duration of the session or active network connection. For constrained devices, this may be a resource constraint, and failures will cause loss or corruption of the session state. MQTT is built on TCP to get a lossless, ordered stream of bytes between the two nodes (sensor–broker or broker–subscriber). Fig. 3 and 4 illustrate one MQTT call flow scenario under loss and delay. When sending data from a sensor to the MQTT broker, a single data response may require more than 12 packets to be transferred between endpoints.

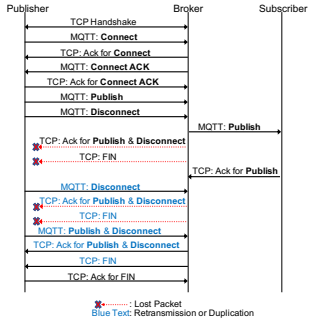

Fig. 3. Call Flow: MQTT under loss condition with QoS = 1.

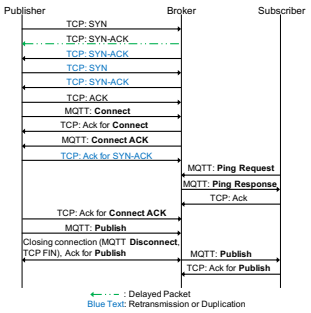

Fig. 4. Call Flow: MQTT under delay condition with QoS = 1.

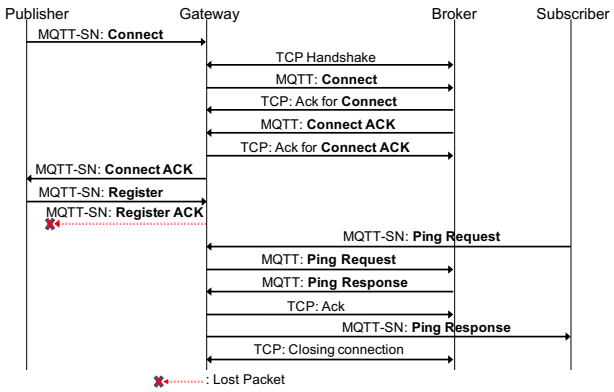

Fig. 5. MQTT-SN Call Flow under Packet Loss.

C. MQTT-SN

A variant of MQTT known as MQTT-Sensor Networks (MQTT-SN) [14] built on top of UDP seeks to reduce the overhead of TCP session maintenance associated with MQTT at the publisher. MQTT-SN was designed for constrained devices that communicate over a wireless channel. A new node, the MQTT-SN Gateway, is introduced to connect MQTT-SN clients to the MQTT broker, which basically translates messages sent from the publishers using the MQTT-SN protocol into MQTT protocol messages used by the broker, and vice versa.

MQTT-SN differs from MQTT in several ways. These include replacing the topic name by a shorter topic ID, the use of predefined topic IDs and a gateway advertisement, discovery and publish protocol. The MQTT-SN gateway can either be a separate node or co-located with the MQTT broker. Another node called a forwarder may also be used to allow the MQTT-SN clients to communicate with the MQTT-gateway when the MQTT-SN gateway is not directly attached to the network. MQTT-SN, like CoAP, does not support message fragmentation and reassembly. The advantage of MQTT-SN from an IoT device perspective is that since it can be implemented over an unreliable transport, in terms of the number of messages sent out from the publisher, it has less overhead and is more efficient, running over an unreliable transport. Fig. 5 illustrates one MQTT-SN call flow scenario under loss conditions. In addition to the MQTT packets between the MQTT-SN gateway and the MQTT broker, MQTT-SN requires at least an additional 6 packet transfers between the publisher and the MQTT-SN gateway to service a single data response, thus requiring approximately 18 packets overall.

D. QUIC

QUIC, a transport protocol developed by Google, seeks to address several transport and application layer challenges that modern applications experience [15].The protocol runs over UDP, emulating the functionality of the set of TCP+TLS+HTTP protocols. Some advantages of QUIC include reduced time to establish a connection since it uses one round trip to complete a handshake compared with TCP+TLS which use 1-3 round trips and multiplexing with no head of line blocking compared to TCP+HTTP.

QUIC uses TCP-CUBIC's mechanisms for congestion control and includes additional information in the congestion control algorithm, in particular including sequence numbers for both the original and retransmitted packets. This allows the QUIC sender to identify retransmitted ACKs and avoid the retransmission ambiguity problem. QUIC ACKs also carry the delay between packet receipt and acknowledgement to more accurately estimate round trip time. It supports up to 256 NACK (Negative ACK) ranges to make QUIC more resilient to reordering. Another key advantage with QUIC is that it avoids head-of-line blocking which occurs when one lost packet in a stream prevents delivery of subsequent packets until the lost packet is delivered. In HTTP2, head of line blocking affects all streams using that TCP connection but with QUIC, packets lost for an individual stream only impact that stream.

QUIC requires approximately 17 packets exchanged between the client and server to send data to a sink and most of the packet content is encrypted. Most of these extra packets are for increased reliability, congestion and flow control. Since the QUIC protocol was initially designed for web transactions, these packets provide improved performance in that environment. However reducing the number of packets may result in a lighter weight version of QUIC, with improved performance, that is suitable for an IoT environment while providing adequate reliability and congestion control.

We postulated that although QUIC has not been used in the context of IoT, some of its features may be useful for IoT communication.A streamlined QUIC protocol may be effective in an IoT environment. For example, the Connection ID feature could be used to enable reliable communication with fewer RTTs than TCP. The use of multiple streams to an individual IoT device could address the head of line blocking with TCP.

IV. Evaluation

A. Experiment Setup

The performance of CoAP, MQTT, MQTT-SN and QUIC was evaluated using the metrics of task completion time (which refers to the total duration of the experiment), and total packets transferred (by nearest node to the IoT device) during the experiment, to get an indication of the overhead with each protocol.

TABLE I
EXPERIMENT PARAMETERS.

Experiment	Value	Description
Base/Ideal	-	No loss, delay or disruptions
Packet Loss Range (%)	0-5, 0-10, 0-20	New value every 100 requests
Network Delay (s)	0-1, 0-2, 0-5	New value every 100 requests
Disruption duration (s)	20, 60	5 disruptions per experiment

A simple scenario of a client obtaining information from a sensor, is considered to allow us analyze the behavior at the level of an individual IoT device under different conditions (Fig. 6). The behaviour of a network of IoT devices can then be inferred from the results. For evaluation, the experiments were first performed in an emulated environment using VirtualBox VMs. A subset of the experiments were also run using Raspberry Pis [16] over a WiFi network to verify the same behavior in an actual wireless environment. Typical IoT environments utilize wireless communication and incur loss, delay and disruptions due to external factors. Therefore the use of WiFi while also varying the loss, delay and disruption parameters allows us to emulate a typical IoT environment. Delay and loss were emulated using traffic control TC [17].

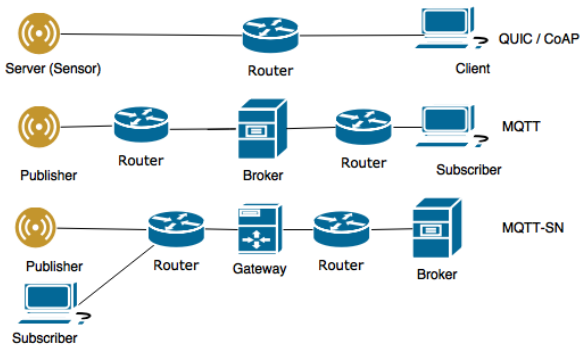

Fig. 6. Application Scenarios

The CoAP server and client are based on a C implementation, namely Libcoap [18]. The MQTT Publisher/Subscriber/Broker are based on the Mosquitto Broker v3.1 software [19]. MQTT-SN implementations leveraged the Eclipse's Paho Library [20] and we used a C implementation of QUIC developed by Google called proto-quic [21].

For each experiment, in the request-response protocols (CoAP and QUIC), the client sent 1,000 data requests and the server (IoT node) responded with 1,000 data responses. Each new data request was sent sequentially after receiving the response to the previous request. In the publish-subscribe protocols (MQTT and MQTT-SN), the publisher sent 1,000 data responses to the broker for a pre-subscribed topic.

A data request/response may contain multiple packet transfers between endpoints and the protocol's performance was studied under ideal conditions and then under varying network loss, delay and disruption conditions.

The parameters used are shown in Table I. CoAP experiments were run for Confirmable and Non-confirmable modes, while MQTT and MQTT-SN used the QoS 0 and QoS 1 variants. In the case of MQTT/MQTT-SN, a new TCP connection was established for each data response and experiments were run multiple times for each scenario.

One aspect under investigation was the behaviour of the protocols within their timer limits and when these timers expired. Retransmission timers were used to evaluate this behavior. In order to evaluate all the protocols under identical conditions, we used the CoAP default timer values as a guide because it has the largest timer values that are also static. The two timers used were the ACK_TIMEOUT and the total retransmission time. In the delay case, the default ACK_TIMEOUT for the first CoAP Confirmable message retransmission is between 2s and 3s. Therefore we selected 1s, 2s and 5s as the upper bounds on delay. For a CoAP Confirmable message, the default time from first transmission to last retransmission is 45s (i.e. MAX_TRANSMIT_SPAN) therefore we selected a short disruption (20s) within the MAX_TRANSMIT_SPAN and a long disruption 60s that exceeds the MAX_TRANSMIT_SPAN.

For the ideal case, there is no network loss or delay. For the experiments with network loss and delay, the instantaneous loss and delay values were changed every 100 requests and uniformly distributed in the range specified in Table I.

For network disruption experiment, five random network disconnections occurred for the given disruption duration in I. Wireshark traces were analyzed to evaluate protocol performance.

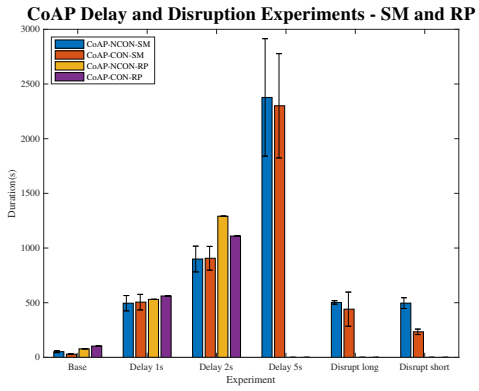

Fig. 7. CoAP: Task Completion Time with Delay and Disruption

B. Performance Analysis

*1) **CoAP**:* Task completion time for CoAP under delay and disruptions is shown in Figure 7 and loss in Figure 8. As the packet loss increases, task completion time increases. One significant observation (in both environments) is that the task completion time for the Non-confirmable CoAP is significantly longer than Confirmable CoAP.

However the Raspberry Pi experiment results (RP) are lower than in the emulation experiment (SM). This is because the SM case for the loss experiment includes a higher script processing delay when changing the loss values every 100 requests. The cause of the high duration with non-confirmable CoAP is explained next.

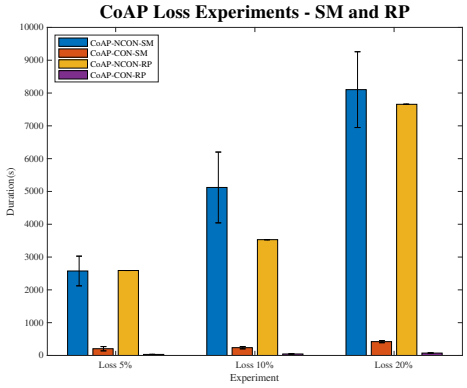

Fig. 8. CoAP: Task Completion Time with Loss

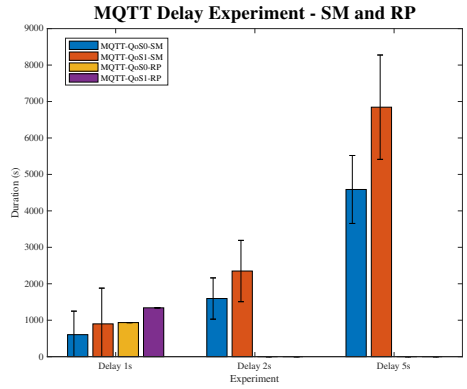

Fig. 9. MQTT: Task Completion Time with Delay

When a Confirmable client request does not receive a response, it uses a back-off timer to generate subsequent retransmissions. From the loss experiments, generally the second or third retransmission was successful leading to the reduced task completion time when compared with non-confirmable case. However considering the worst case scenario when the maximum number of retransmissions (i.e. 4 by default) has been reached the client would give up on that data request and send a new data request for the next piece of information.

If a client does not receive an immediate response to a non-confirmable request, it does not retransmit the request; instead, it waits for 90s before sending the next request. From our experiments, confirmable requests had a maximum wait time of approximately 30s if a response was lost (due to retransmissions), versus the 90s wait time in the Non-confirmable case. This may be categorized as an unexpected, and potentially undesirable, effect since the Non-confirmable option would likely be chosen without the expectation of introducing such a significant performance penalty while reducing overhead. Further analysis indicated this was due to two factors: the NSTART parameter and the timer determining when to send the next request.

The NSTART parameter is the maximum number of simultaneous outstanding interactions at any time between a specific client and server, and has a default value of 1 (for both

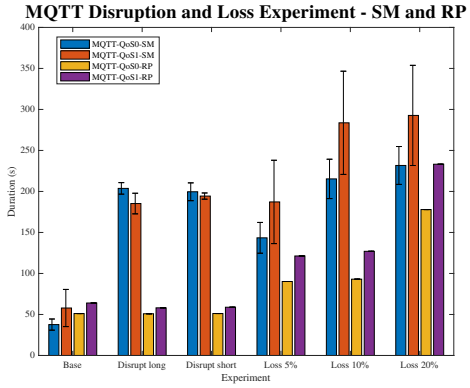

Fig. 10. MQTT: Task Completion Time with Disruption and Loss

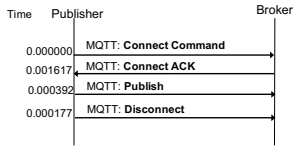

Fig. 11. MQTT: Time between MQTT Control Commands QoS 0

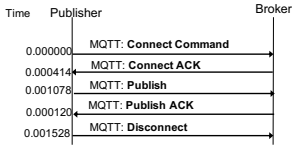

Fig. 12. MQTT: Time between MQTT Control Commands QoS 1

Confirmable and Non-confirmable). An outstanding interaction is an outstanding confirmable request which is still awaiting an ACK or a request still waiting for a response or ACK [22].

For the timer to send the next request for Non-confirmable messages, the RFC does not explicitly indicate which timer should be used to determine how long to wait or what algorithm to use to decide when to stop expecting a response. In the case of Confirmable requests, however, the MAX_TRANSMIT_WAIT timer determines the maximum time a sender should wait from the time of the first transmission of a Confirmable message to when it should give up waiting for an Acknowledgement or Reset message. Results from using libcoap indicate that the timer used had a value of 90s, corresponding to the MAX_TRANSMIT_WAIT timer. This points to a weakness of depending on non-adaptive application-layer timers, which have the potential of unintentionally introducing performance limitations.

In the case of delay experiments, the base time is the task completion time in the ideal case and requests are sent sequentially after receiving a response (due to NSTART=1). For the delay experiments, the new task completion time is approximately equal to the sum of base time + (delay * no of packets) + script processing time. The experimental results closely align with the expected result, with the network delay (delay * no of packets) contributing to over 90 % of the total task completion time. The remaining 10% is due to base time and script processing delay. The base time includes internal processing at the receiving node and propagation delay in the ideal case while the script processing delay is the time needed to switch to new delay parameters using TC. Compared to the emulation experiment, the switching delay is larger with the Raspberry Pi delay experiments. This caused the difference in

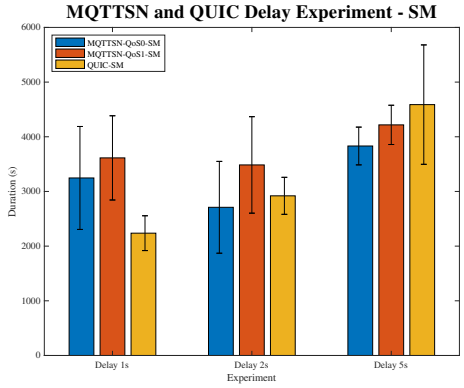

Fig. 13. MQTT-SN and QUIC: Task Completion Time with Delay

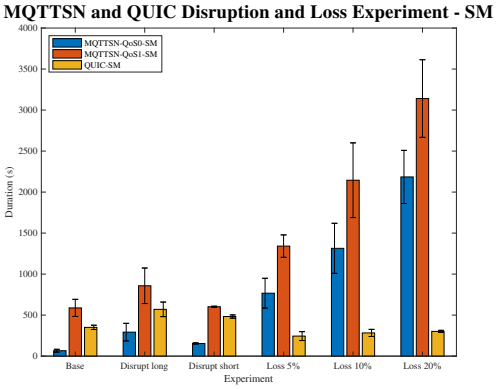

Fig. 14. MQTT-SN: Task Completion Time with Loss and Disruption

duration and depends on the current delay value i.e. changing the delay value in the 1s delay experiment takes less time than changing values in the 5s delay experiment.

For the disruption experiment, with Confirmable requests, in the majority of the cases with the short disruption (20s), the network had recovered by the time the last retransmission was due. For long disruptions (60s), the maximum number of retries was exceeded while the network was still unavailable, so the client dropped that request and proceeded to the next request. Since this new request is sent while the network is still unavailable, it may also require retransmissions.

In the disruption experiment Non-confirmable client exhibit the same behavior as described for packet loss in the network – when it receives no response, it waits for the 90s timer to expire before sending the next request. This calls for an adaptive timer, or a well-defined 'cookbook' of how applications should set the timer for the Non-confirmable case.

*2) **MQTT**:* With MQTT, increase in loss and delay causes a corresponding increase in the completion time of the experiment due to retransmissions (Fig. 9 and Fig. 10). The duration is significantly longer with the delay experiments because a single MQTT transaction requires multiple messages to be transferred, and each of these messages is affected by the delay, leading to a cascading effect on the duration. Similar to the CoAP delay experiment, the delay script processing

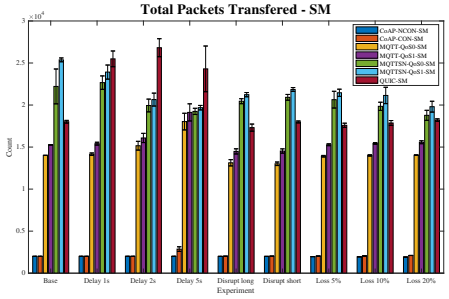

Fig. 15. Total Packets Comparison

time was significantly longer in the Raspberry Pi (RP) delay experiment than the emulation experiment (SM). On the other hand, in the loss experiment, the script processing delay was more pronounced in the emulation experiment.

In MQTT and MQTT-SN, the difference between QoS 0 and QoS 1 requests is that QoS 1 requires an acknowledgement for any PUBLISH messages that are sent. With MQTT, changing from QoS 0 to QoS 1, leads to an average increase of 48% and 30% in the duration for the delay and loss experiments respectively. This is because in QoS 0, the PUBLISH and DISCONNECT control messages are sent back-to-back from the publisher, while for QoS 1, the publisher needs to wait for the PUBLISH ACK and complete internal processing after receiving it before it can send the DISCONNECT. Fig. 11 illustrates the call flow of just these control messages for QoS 0, while Fig. 12 is for QoS 1.

In the case of QoS 0, the DISCONNECT is sent after approximately 0.177 ms after the PUBLISH. However, in the case of QoS 1, the DISCONNECT is sent after 1.648ms (1.528+0.12ms), which is an approximately 10x increase when compared with QoS 0.

In terms of total packets transmitted during the experiments, there is only a small increase of less than 10 % when changing from QoS 0 to QoS 1 in the loss and delay experiments (see Fig. 15 which shows results of the emulation experiment). Thus, the cost of increased reliability by using QoS 1 is because of the additional round trip significantly increasing time to completion although both QoS 0 and 1 have about the same number of packets. This implies that time sensitive applications can benefit with QoS 0 (without reliability).

For the delay experiments, since multiple messages are exchanged between the two nodes, the impact of delay is significantly higher, see. Fig. 9.

*3) **MQTT-SN**:* For MQTT-SN, increase in loss or delay results in a decrease in the total number of packets transferred.

This is because, if packet loss occurs between the publisher and gateway (e.g., the CONNECT ACK from gateway to the publisher is lost), the rest of the transaction packets such as the PUBLISH and DISCONNECT messages will not be sent.

Task completion time increases with delay and loss and this is partly caused by timers (Keep Alive value in CONNECT message) which control how long to wait before sending the the next message. Comparing the change in task completion time for varying delays, there is an increase of 10-30% in

the task completion time for QoS 1 compared to QoS 0. For loss and disruption, the increase in task completion time with QoS 1 is significantly higher, at over 40 %. The reason for this is similar to the MQTT case. With QoS 1, the publisher needs to wait for a Publish ACK before it can send the final disconnect to the gateway, causing a substantial increase in the time between those commands, increasing from about 100 microseconds to hundreds of milliseconds.

*4) **QUIC**:* In the case of delay, the task completion time with QUIC is comparable to MQTT-SN because a single QUIC transaction requires multiple messages to be exchanged between two end hosts. However in the case of delay, the total number of QUIC packets exchanged is significantly higher than when using any of the other protocols (see Fig. 15). In the disruption and loss cases, the total number of packets sent by QUIC is fewer than MQTT-SN with a smaller task completion time. Therefore in a lossy or disconnected environment, QUIC provides similar performance to MQTT-SN in terms of task completion time and packets transferred. QUIC, originally designed for resource rich nodes has significantly more packets transferred to get the same number of values from the sensor. However, reducing the packet transfers, by reducing congestion control and reliability information for example, may result in a lightweight version of QUIC, which may provide sufficient QoS guarantees and improved performance to be suitable in an IoT environment.

V. Conclusions

In this paper, we extensively analyzed the performance of four application layer protocols for constrained IoT devices over networks with impairments. We performed multiple experiments and evaluated the performance of CoAP, MQTT, MQTT-SN and QUIC in networks with delay, loss and disruptions. Results show that in terms of overhead, CoAP is the most efficient protocol as it only requires two messages for data transfer. Results also show that the cost of using additional reliability with MQTT QoS 1 has more of an impact on task completion time (over 30 % difference) than overhead (10 % difference). For QUIC, in its current version, the performance of QUIC in lossy and disruptive environments is comparable to MQTT-SN. Therefore, a streamlined QUIC protocol may significantly improve performance and enable QUIC to be a potential request-response IoT protocol alternative to CoAP.

Another key finding from the experiments is that for IoT protocols that use a fire-and-forget paradigm, such as CoAP non-confirmable, MQTT QoS 0 and MQTT-SN QoS 0, the wait timers (keep alive for MQTT/MQTT-SN) play a crucial role in performance. These timers determine how long the node should wait before giving up on expecting a response and/or determine when the protocol can proceed to the next request. In the case of CoAP non-confirmable, we recommend an adaptive timer or a well-defined cookbook of how applications should set the timer. Future work includes implementation of an adaptive timer for the non-confirmable CoAP as well as evaluating the performance of these protocols in networks of multiple heterogeneous and homogeneous IoT devices.

Acknowledgment

This work was supported in part by NSF grants CNS-1619441, CNS-1618344 and a grant from Futurewei Inc.

References

[1] O. Vermesan, P. Friess, P. Guillemin, S. Gusmeroli, H. Sundmaeker, A. Bassi, I. S. Jubert, M. Mazura, M. Harrison, M. Eisenhauer *et al.*, "Internet of things strategic research roadmap," *Internet of Things-Global Technological and Societal Trends*, vol. 1, no. 2011, pp. 9–52, 2011.

[2] M. R. Palattella, N. Accettura, X. Vilajosana, T. Watteyne, L. A. Grieco, G. Boggia, and M. Dohler, "Standardized protocol stack for the internet of (important) things," *IEEE communications surveys & tutorials*, vol. 15, no. 3, pp. 1389–1406, 2013.

[3] A. Al-Fuqaha, M. Guizani, M. Mohammadi, M. Aledhari, and M. Ayyash, "Internet of things: A survey on enabling technologies, protocols, and applications," *IEEE Communications Surveys & Tutorials*, vol. 17, no. 4, pp. 2347–2376, 2015.

[4] L. Atzori, A. Iera, and G. Morabito, "The internet of things: A survey," *Computer networks*, vol. 54, no. 15, pp. 2787–2805, 2010.

[5] P. López, D. Fernández, A. J. Jara, and A. F. Skarmeta, "Survey of internet of things technologies for clinical environments," in *Advanced Information Networking and Applications Workshops (WAINA), 2013 27th International Conference on*. IEEE, 2013, pp. 1349–1354.

[6] R. Khan, S. U. Khan, R. Zaheer, and S. Khan, "Future internet: the internet of things architecture, possible applications and key challenges," in *Frontiers of Information Technology (FIT), 2012 10th International Conference on*. IEEE, 2012, pp. 257–260.

[7] D.-L. Y. F. L. Yi and D. Liang, "A survey of the internet of things," *Proc. of ICEBI*, 2010.

[8] I. F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci, "Wireless sensor networks: a survey," *Computer networks*, vol. 38, no. 4, pp. 393–422, 2002.

[9] D. Thangavel, X. Ma, A. Valera, H.-X. Tan, and C. K.-Y. Tan, "Performance evaluation of mqtt and coap via a common middleware," in *Intelligent Sensors, Sensor Networks and Information Processing (ISSNIP), 2014 IEEE Ninth International Conference on*. IEEE, 2014, pp. 1–6.

[10] S. Bandyopadhyay and A. Bhattacharyya, "Lightweight internet protocols for web enablement of sensors using constrained gateway devices," in *Computing, Networking and Communications (ICNC), 2013 International Conference on*. IEEE, 2013, pp. 334–340.

[11] N. De Caro, W. Colitti, K. Steenhaut, G. Mangino, and G. Reali, "Comparison of two lightweight protocols for smartphone-based sensing," in *Communications and Vehicular Technology in the Benelux (SCVT), 2013 IEEE 20th Symposium on*. IEEE, 2013, pp. 1–6.

[12] C. Bormann, K. Hartke, and Z. Shelby, "The constrained application protocol (coap)," *RFC 7252*, 2015.

[13] U. Hunkeler, H. L. Truong, and A. Stanford-Clark, "Mqtt-sa publish/subscribe protocol for wireless sensor networks," in *Communication systems software and middleware and workshops, 2008. comsware 2008. 3rd international conference on*. IEEE, 2008, pp. 791–798.

[14] A. Stanford-Clark and H. L. Truong, "Mqtt for sensor networks (mqtt-sn) protocol specification," *International business machines (IBM) Corporation version*, vol. 1, 2013.

[15] J. Iyengar and M. Thomson, "QUIC: A UDP-Based Multiplexed and Secure Transport," Internet Engineering Task Force, Internet-Draft draft-ietf-quic-transport-11, work in Progress. [Online]. Available: https://datatracker.ietf.org/doc/html/draft-ietf-quic-transport-11

[16] M. Richardson and S. Wallace, *Getting started with raspberry PI.* " O'Reilly Media, Inc.", 2012.

[17] "TrafficControl." [Online]. Available: https://wiki.debian.org/TrafficControl

[18] O. Bergmann, "libcoap: C-implementation of coap," *URL: http://libcoap. sourceforge. net, Date of access 13.09*, 2012.

[19] R. A. Light, "Mosquitto: server and client implementation of the mqtt protocol," *Journal of Open Source Software*, vol. 2, no. 13, 2017.

[20] "Eclipse Paho - MQTT and MQTT-SN software." [Online]. Available: https://www.eclipse.org/paho/

[21] "QUIC." [Online]. Available: https://www.chromium.org/quic/playing-with-quic

[22] C. Bormann, A. P. Castellani, and Z. Shelby, "Coap: An application protocol for billions of tiny internet nodes," *IEEE Internet Computing*, vol. 16, no. 2, pp. 62–67, 2012.