And QUIC meets IoT: performance assessment of MQTT over QUIC

Fátima Fernández†, Mihail Zverev†, Pablo Garrido†, José R. Juárez†, Josu Bilbao†, Ramón Agüero¶

† Ikerlan Technology Research Centre,
Basque Research Technology Alliance (BRTA)
Arrasate/Mondragon, Spain
{ffernandez, mzverev, pgarrido, jrjuarez,
jbilbao}@ikerlan.es

¶Dpt. of Communication Engineering
University of Cantabria
Santander, Spain
ramon@tlmat.unican.es

***Abstract*—We study the performance of the Message Queuing Telemetry Transport Protocol (MQTT) over QUIC. QUIC has been recently proposed as a new transport protocol, and it is gaining relevance at a very fast pace, favored by the support of key players, such as Google. It overcomes some of the limitations of the more widespread alternative, TCP, especially regarding the overhead of connection establishment. However, its use for Internet of Things (IoT) scenarios is still under consideration. In this paper we integrate a GO-based implementation of the QUIC protocol with MQTT, and we compare the performance of this combination with that exhibited by the more traditional MQTT/TLS/TCP approach. We use Linux Containers and we emulate various wireless network technologies by means of the ns-3 simulator. The results of an extensive measurement campaign, show that QUIC protocol can indeed yield good performances for typical IoT use cases.**

***Index Terms*—Quic UDP Internet Connections (QUIC), Internet of Things (IoT), Message Queuing Telemetry Transport (MQTT), Lossy Networks, Performance Analysis**

I. Introduction

Industry and companies are witnessing a deep transformation, known as Industry 4.0, which is indeed considered as the fourth industrial revolution. This term, introduced in 2011 by the German government, sprung as a novel initiative towards the digitalization of its industry, based on four principles: interconnection, information transparency, decentralized decisions and technical assistance [1]. Industry 4.0 yields a remarkable productivity boost, thanks to (among other functionalities) the collection and analysis of real-time data [2].

One of the key technologies behind this industry digital transformation is the IoT paradigm, which enables the connection of industrial components, to collect data, monitor systems, exchange information, and analyze the environment [2]. However, the development of industrial 4.0 applications and services introduces stringent requirements. It is worth mentioning the following ones: low latency communications, high reliability and availability, energy and cost efficiency, and security and privacy [3].

Energy and cost efficiency is a pivotal feature. In order to monitor the industrial environment, a huge deployment of sensors and connected devices is required, but this deployment should not cause a cost increase of the manufacturing process. Besides, the connected devices and sensors must reach different locations and isolated areas and, in many cases, this will imply the use of battery-powered elements, which should stay alive for a long time. Low latency communications are also essential in many industrial applications. Very fast responses might be crucial in some scenarios in order to enhance manufacturing processes, to guarantee the security of many involved elements, and to provide the chance to enable remote control functions in real industrial scenarios.

The well-known MQTT protocol [4], [5] is widely used in IoT applications, due to its small code footprint, easy integration and good performance. It traditionally lays, as its transport protocol, on TCP [6], which offers a reliable end-to-end communication service. However, TCP suffers from many disadvantages, specially those related to the ossification of internet protocols, which are not able to adapt to the fast-changing technologies [7]. Dealing with lossy networks has been shown to be very challenging for TCP, drastically jeopardizing its performance and increasing end-to-end delay [8]. Moreover, the extremely low communication delay requirement of some applications is not appropriately guaranteed by TCP [9]. Therefore, many alternatives have emerged, some of them being TCP modifications or updates, such as SCTP [10], Real-Time TCP [11] and Network Coded TCP [12] among others. One of these alternatives is QUIC [13], an experimental transport protocol, whose main design principles were to reduce connection establishment and transport latencies, as well as to improve security standards with default end-to-end encryption in HTTP-based applications [14].

In this paper we propose to integrate IoT protocols (MQTT in particular) with QUIC, in order to assess the performance of such combination and to analyze the benefits that novel transport protocol approaches can bring to the IoT realm, compared to legacy ones.

The contributions of this paper are:

- Integration of a fully operational MQTT implementation (in GO programming language) with a QUIC GO implementation.

- All the code has been made available in a public git repository.
- Performance analysis of MQTT over QUIC, over various network technologies, by means of an extensive measurement campaign carried out over the ns-3 framework and Linux containers.

The rest of the paper is structured as follows. Section II discusses related works. Section III sketches the integration of MQTT and QUIC protocol. Section IV depicts the setup that was used to carry out the experiments, and discusses the results that were obtained, comparing the performance of the proposed scheme with a more traditional solution. Finally, Section V concludes the paper, outlining the aspects that we will tackle in our future work.

II. Related work

A. MQTT

MQTT [4] is a popular lightweight network protocol based on the publish-subscribe model. It has been widely implemented across a variety of industries since 1999 to connect small devices, thanks to its small code footprint and low network bandwidth required. Version 3.1.1 was submitted in 2014 by IBM, and it was standardized by International Organization for Standardization (ISO) and Organization for the Advancement of Structured Information Standards (OASIS). Furthermore, version 5.0 has been recently standardized [5].

In MQTT there exist three types of devices: subscriber, publisher and broker. Publishers are usually small sensors, which generate information and publish it into a common broker on specific topics. On the other hand, a subscriber consumes the data produced by the publishers. In this sense, they subscribe to a topic, and they would thus receive all published messages on that particular topic, by any publisher. Publishers and Subscribers can be both seen as clients on the corresponding network topology. The key element of MQTT is the broker server, which manages the subscriptions. All messages published in the network are sent to the broker, which takes care of distributing the information to the corresponding subscribers. The broker also considers the various Quality of Service (QoS) levels for the clients and so the potential retransmissions. Although the clients only interact with a broker, the system may contain several brokers, which exchange data based on their current subscribers' topics.

One of the strong aspects of MQTT is the isolation between producers and consumers. This facilitates the implementation of such functionalities in low computational devices, which can interact amongst them by means of the publish-subscription paradigm. Another advantage is that the publisher can send new content whenever it becomes available, thus decoupling the temporary relationship between a node's interest and the publication of the information.

B. QUIC

QUIC is a transport protocol originally developed by Google Inc. [14] and currently standardized by IETF [13]. QUIC addresses two big challenges of today's web traffic: minimizing latency for a better user experience; and securing the communications (end-to-end payload and header encryption [13], [15]), as Internet is shifting to a more secure web traffic [14]. Figure 1 depicts the QUIC protocol stack, comparing it with the traditional TLS/TCP approach, when used to transport http traffic.

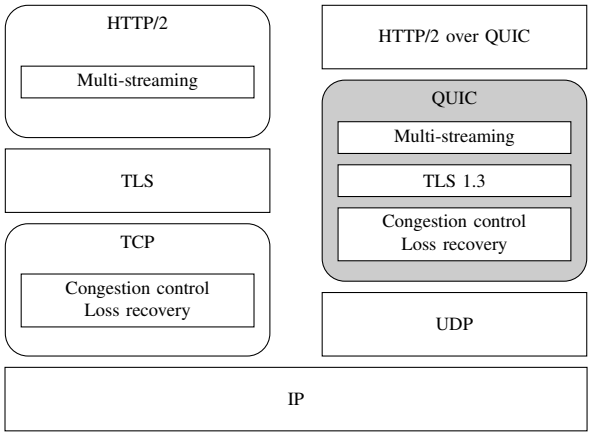


Fig. 1. QUIC archicture

QUIC reduces handshake latency, establishing a connection secured with TLS 1.3 in just 1 Round Trip Time (RTT), 0 RTT if the endpoints have previously established a communication [15], and the data can be sent before handshake happens in 0-RTT packets [13]. On the other hand, the legacy TCP 3-way handshake allows an endpoint to send data only after 1 RTT [6]. TCP alone does not encrypt its payload, and additional protocols, such as TLS, are required to establish a secure connection. TLS 1.2 connection establishment takes 2 RTT [16] and TLS 1.3 takes 1 RTT [17], leading to an overall TLS/TCP handshake of 3 or 2 RTT, respectively.

The information within a QUIC connection is organized in *streams*, which enables QUIC to prevent delays caused by head-of-line blocking. When a packet loss occurs, only streams with data in such packet are blocked waiting for a retransmission to be received, while other streams can go on [13]. QUIC also reduces latency with its loss detection algorithms, which include early retransmits and tail loss probes [18].

QUIC has been designed to overcome TCP ossification [7], [14]. QUIC uses UDP datagrams, so middleboxes consider it as UDP payload. Its header and payload encryption avoids incurring a dependency on middleboxes [13]. QUIC is implemented in user-space, which gives more freedom in terms of computational resources, and thus richer logging, more interaction with server systems and an easier protocol update [14]. However, implementation at user-space is not mandatory, and QUIC can be embedded into kernel for boosting its performance [19].

QUIC includes a version negotiation mechanism, enabling coexistence of different QUIC modifications. This simplifies protocol update, and enables protocol extension and customization. QUIC extensions could even be designed as plug-

ins for a major protocol extension, which allows an endpoint to share an extension that another endpoint is missing [20].

C. QUIC for IoT scenarios

Although QUIC development is still ongoing, some researchers have already included it within IoT protocol stack [21], [22]. However, to the best of our knowledge, there do not exist many works addressing the evaluation of QUIC in IoT scenarios.

Liri et al. assessed QUIC as an IoT protocol in [23]. Their study reveals that QUIC, as a replacement to more traditional IoT protocols, is outperformed by CoAP [24]. However, QUIC performance in lossy and disruptive environments is comparable to MQTT-SN (MQTT - Sensor Networks), a variant of MQTT for constrained devices. The authors suggest that a more streamlined version of QUIC could be a potential request-response IoT protocol alternative to CoAP.

Kumar and Dezfouli studied Google QUIC performance over IoT scenarios in [25]. They compared MQTT performance over QUIC and TCP in different testbeds, built with Raspberri Pi 3B devices. The performance was assessed in terms of packet overhead of connection establishment, latency in the presence of random packet erasures, processor and memory usage when one of the endpoints dropped its connection (half-open connection) and throughput drops during connection migration. Their results show that QUIC outperforms TCP in multiple aspects, and they identify some points that QUIC should improve for a better performance in IoT scenarios.

Lars Eggert has recently analyzed the feasibility of deploying QUIC over constrained IoT devices [26]. He used, for the prelimary evaluation, more constrained devices than those used in [25]: Particle Argon and ESP32-DevKitC V4. In order to reduce QUIC memory usage in low-end devices, some of the original QUIC features, considered impractical for IoT, were changed or removed. By evaluating memory usage and energy consumption of low-end devices running the modified QUIC, the paper concludes that it is indeed a practical alternative to be considered for IoT edge devices.

This paper analyzes QUIC, as defined in [13], performance over IoT scenarios. Rather than focusing on how well QUIC deployment adapts to the impairments of low-end devices, as was the main goal of [26], we broaden the evaluation of QUIC, paying special attention to its reduced latency. We consider different IoT scenarios, over which devices communicate by means of MQTT on top of both QUIC and TCP. We study the overall communication time, comparing the performance of both transport protocols over error-free and erasure channels with realistic values of Frame Error Rate (FER).

III. Implementation

In this paper, we use a QUIC implementation in GO, *quic-go*[1]. It follows the IETF QUIC draft, which is heavily under development.

The MQTT client, as well as the server broker, are based on the open-source Eclipse Paho[2] and VolantMQ[3], respectively, both of them also implemented in GO. They support the full specification of MQTT, 3.1 and 3.1.1 versions. The client can connect with the broker using TCP, TLS or WebSocket. We started from the open-source implementations of the MQTT client and broker, and we integrated QUIC on both projects.

Our client implementation has been made available, and can be accessed at a public git repository[4]. In short, all the functionalities at the transport layer, i.e opening and closing connections, or sending and receiving packets, are managed from the *net.go* interface. Thanks to this approach, we are able to integrate QUIC, with very few additional changes within the remainder of the code. This is of utter relevance, since we avoid changes in the MQTT Client and so support TCP, TCP+TLS, QUIC and WebSocket connections. In order to support QUIC connections, the *net.go* interface calls, from *quic-go*, the *client.go* interface. Due to 0-RTT approach, we make use of DialAddrEarly function to keep the ticket for resumption sessions and exchange "early" data before the handshake completes.

On the other hand, the Broker implementation is also publicly available in a git repository[5]. In this case, the transport level functionalities are implemented using the *transport/conn.go* interface. Due to implementation restrictions, the incompatibilities between the connection interfaces offered by TCP and QUIC, this MQTT server implementation only supports QUIC connections. We integrate *quic_udp.go* interface which calls the listener from *server.go* interface (*quic-go*). The listener function is ListenAddrEarly to enable 0-RTT on the server broker side.

In our implementation effort, we have also identified some aspects that are yet not fully operational in QUIC. For example, an interesting scenario might encompass periodical data bursts from publishers. Transport protocols, TCP in particular, are able to combine multiple packets from higher level protocols in one single transport packet. We found that *quic-go v0.15.1* is unable to group multiple short MQTT packets coming from the same data stream. We could not thus analyze such scenarios, which are left for future work.

IV. Results

In order to assess the performance of the combination of MQTT and QUIC we have carried out an extensive simulation campaign over the ns-3 framework, a widespread discrete-event network simulator. In particular, we exploit the feature that allows us to connect real application traffic, running over Linux containers, over a simulated network. We build an emulated environment, which is depicted in Fig. 2. As can

[1]A QUIC implementation in pure go. https://github.com/lucas-clemente/quic-go, version *v0.15.1*.

[2]Eclipse Paho MQTT client. https://github.com/eclipse/paho.mqtt.golang, version *v1.2.0*.

[3]High-Perforance MQTT Server. https://github.com/VolantMQ/volantmq, version *v0.4.0-rc.6*.

[4]Eclipse Paho MQTT client with QUIC support. https://github.com/pgOrtiz90/paho.mqtt.golang

[5]High-Perforance MQTT Server with QUIC support. https://github.com/fatimafp95/volantmq_2

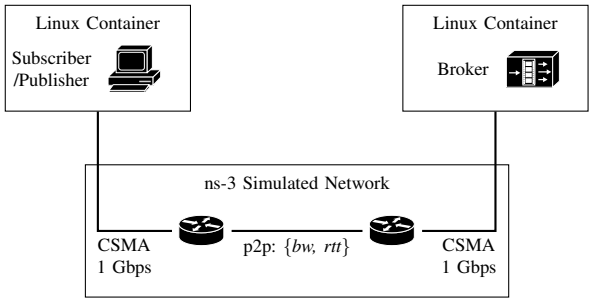


Fig. 2. Emulation scenario

TABLE I
NETWORK PARAMETER RANGES FOR DIFFERENT WIRELESS TECHNOLOGIES

	NetType1	**NetType2**	**NetType3**
	WiFi	**4G/LTE**	**Satellite**
Capacity [Mbps]	20	10	1.5
RTT [ms]	25	100	600
Loss [%]	0 .. 5	0 .. 5	0 .. 5

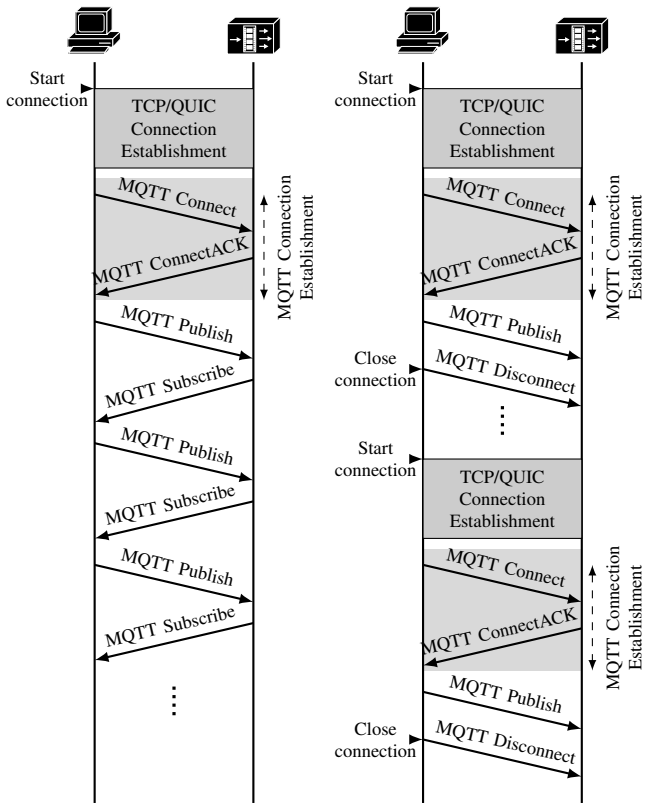
be seen, it entails two Linux containers connected through ns-3, which is used to simulate various wireless technologies, modeled by means of two parameters: bandwidth and delay. One Linux container runs the client application, which consists on a publisher and a subscriber, while another Linux container hosts the broker server. The containers are seen as ghost nodes by the ns-3 instance, each connected over a CSMA network to a node, which could be interpreted as the network router. The capacity of this connection is rather high, to ensure that the bottleneck is over the wireless link, which is where our interest lies. Routers are then connected over a point-to-point link, which can be configured with different combinations of bandwidth, delay and loss rates, mimicking various network technologies and conditions.

In Table I, we show how the parameters that were used to reflect different network technologies. Although there exists evidence that cellular networks maintain buffer sizes larger than the path's Bandwidth Delay Product (BDP), leading to the so-called bufferbloat effect [27], we adjust all buffers to be one BDP.

In our first scenario, which is depicted in Figure 3a, we send 1000 MQTT packets from the publisher to the broker, which then sends the corresponding message to the subscriber. We use one single MQTT connection, which is not closed during the experiment. We configure the MQTT service to follow a stop & wait behavior, so that the i^{th} packet is only sent after receiving the reply for the previous one. We measure the time required to finish with the transmission of all packets when using both TCP and QUIC: $T_{\text{TCP}}, T_{\text{QUIC}}$, respectively. We then define the completion ratio, ξ as:

$$\xi = \frac{T_{\text{QUIC}}}{T_{\text{TCP}}} \tag{1}$$

Hence, when $\xi < 1$, we could infer that QUIC outperforms TCP, since the required time would be shorter. We used the

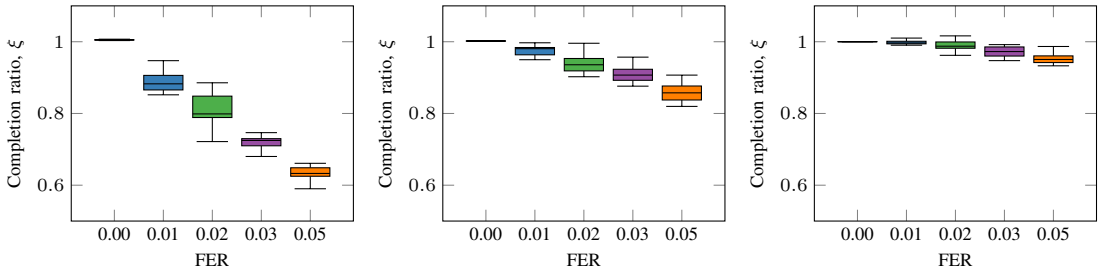


(a) 1 single MQTT connection (b) Multiple MQTT connections

Fig. 3. Scenarios used during the experiments

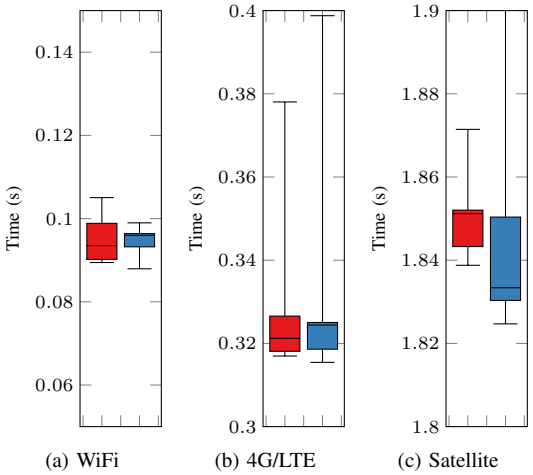
three network configurations that are depicted in Table I, with various packet loss rates. For each configuration, we carried out 20 independent experiments, and Figure 4 shows the whisker plots of the completion ratio. As can be seen, QUIC outperforms TCP in all cases, especially over networks with low RTTs (WiFi). This improvement gets more relevant over lossy networks. The results yield an almost 40% reduction when the FER is 0.05 over the WiFi network. We can indeed see that when the RTT gets higher, becoming the prominent contribution in the corresponding delay, the improvement yielded by QUIC is much less noticeable, especially for the satellite connection. In any case, QUIC is not outperformed by TCP, regardless of the configuration. Since the packets that are being transmitted are rather short, we can also say that the impact of the bandwidth over the delayed is almost negligible.

The second setup is depicted in Figure 3b. In this case, we force the MQTT to close after one single publish message transmission. The goal is thus to analyze the improvements brought by the lightweight connection establishment promoted by QUIC, compared to the heavier TCP/TLS traditional scheme. We measure the time since the connection was initially established until the moment when the publish message is sent. We establish two connections in each experiment, and we run an experiment for 20 times. Hence, we have 40 samples per configuration. Figure 5 shows the box plot of such times,



(a) WiFi (b) 4G/LTE (c) Satellite

Fig. 4. Performance of MQTT over QUIC and TCP for a stop&wait transmission



(a) WiFi (b) 4G/LTE (c) Satellite

Fig. 5. MQTT performance over TCP and QUIC: time required to publish a message after connection establishment. Red Whiskers is for QUIC and Blue ones is for TCP

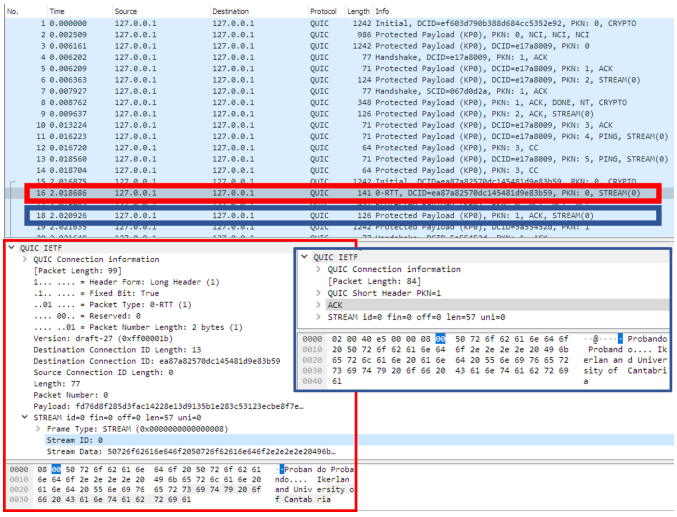


Fig. 6. QUIC connection establishment captured by Wireshark. Packet 16 corresponds to the 0-RTT packet with the test message "Probando Probando.... Ikerlan University of Cantabria". It can be seen that this message is repeated in the packet 18, which has a short header.

for both transport protocols, and the three networks that were previously introduced.

As can be seen, the performance of QUIC is slightly better than the one exhibited by the TCP protocol. We can see that the times are statistically tighter (showing less variability), especially for the cellular and satellite networks, while the corresponding average times are alike. However, we might have expected a bigger advantage, since the 0-RTT approach promoted by QUIC should have yielded shorter times.

This behavior is due to the fact that the quic-go server is not reading 0-RTT packets from its buffer before the client retransmits 0-RTT data in subsequent 1-RTT packets.

In order to assess the validity of this assumption, we modified the configuration of Wireshark[6] so it could correctly interpret QUIC protected payload. Figure 6 shows the corresponding packet exchange, captured with Wireshark in one of our experiments. It can be seen how the test message for 0-RTT packet (#16) is repeated in a short-header packet (#18), which should not be generated before 1-RTT protection is ready [13]. We are currently working towards tuning this behavior, which hinders the benefit that would be obtained with the 0-RTT connection establishment mechanism.

[6]https://www.wireshark.org/

V. Conclusions

In this paper we have studied the performance exhibited by the QUIC protocol when it is used over IoT scenarios. We have integrated a GO-based QUIC implementation with MQTT client/server. Thanks to Linux Containers technology and the emulation possibilities offered by the ns-3 platform, we have carried out an extensive measurement campaign, in which we compare the performance of the combination MQTT/QUIC with that exhibited by the more traditional MQTT/TLS/TCP combination.

We have established two complementary scenarios. On the first one, in which multiple publish/subscribe exchanges occur after the connection establishment, we have seen that the QUIC protocol clearly outperforms TCP, especially for connections having a low RTT. This improvement is more relevant when there are packet losses over the wireless network. As could have been expected the impact of the bandwidth is

almost negligible, since we are considering sporadic exchange of short packets.

The second scenario was conceived to ascertain the benefits brought by the 0-RTT scheme that QUIC promotes. Although the results are promising, since QUIC yields less variability than TCP, we will need to perform more in-depth analysis, since a shorter time could have been expected.

Provided that QUIC implementation is still under heavy development, there are features that are not yet included in its implementation, but we plan to analyze in our future work. One example would be the possibility to group various MQTT messages into a single QUIC packet. We will also exploit the methodology and setup that we have introduced in this paper to study the performance of more complex scenarios, comprising various types of IoT devices, as well as fog/cloud nodes, which would conform a multi-tier IoT architecture, tailored for industrial applications.

ACKNOWLEDGMENT

The authors are grateful for the funding of the Industrial Doctorates Program from the University of Cantabria (Call 2018). This work has been partially supported by the Basque Government through the Elkartek program under the DIGITAL project (Grant agreement no. KK-2019/00095), as well as by the Spanish Government (Ministerio de Economía y Competitividad, Fondo Europeo de Desarrollo Regional, FEDER) by means of the project FIERCE: Future Internet Enabled Resilient smart CitiEs (RTI2018-093475-AI00).

REFERENCES

[1] M. Hermann, T. Pentek, and B. Otto, "Design Principles for Industrie 4.0 Scenarios," in *2016 49th Hawaii International Conference on System Sciences (HICSS)*, 2016, pp. 3928–3937.

[2] G. Aceto, V. Persico, and A. Pescapé, "A Survey on Information and Communication Technologies for Industry 4.0: State-of-the-Art, Taxonomies, Perspectives, and Challenges," *IEEE Communications Surveys Tutorials*, vol. 21, no. 4, pp. 3467–3501, 2019.

[3] A. Varghese and D. Tandur, "Wireless requirements and challenges in Industry 4.0," in *2014 International Conference on Contemporary Computing and Informatics (IC3I)*, nov 2014, pp. 634–638.

[4] A. Banks and R. Gupta, "MQTT version 3.1.1," International Organization for Standardization (ISO), Standard, 2014.

[5] A. Banks, E. Briggs, K. Borgendale, and R. Gupta, "MQTT version 5.0," Organization for the Advancement of Structured Information Standards (OASIS), Standard, 2019.

[6] J. Postel, "Transmission control protocol," Internet Requests for Comments, RFC Editor, STD 7, September 1981, http://www.rfc-editor.org/rfc/rfc793.txt. [Online]. Available: http://www.rfc-editor.org/rfc/rfc793.txt

[7] M. Honda, Y. Nishida, C. Raiciu, A. Greenhalgh, M. Handley, and H. Tokuda, "Is it still possible to extend TCP?" *Proceedings of the ACM SIGCOMM Internet Measurement Conference, IMC*, pp. 181–194, 2011.

[8] Chonggang Wang, K. Sohraby, Bo Li, M. Daneshmand, and Yueming Hu, "A survey of transport protocols for wireless sensor networks," *IEEE Network*, vol. 20, no. 3, pp. 34–40, may 2006.

[9] J. Luo, J. Jin, and F. Shan, "Standardization of Low-Latency TCP with Explicit Congestion Notification: A Survey," *IEEE Internet Computing*, vol. 21, no. 1, pp. 48–55, 2017.

[10] R. Stewart, "Stream control transmission protocol," Internet Requests for Comments, RFC Editor, RFC 4960, September 2007, http://www.rfc-editor.org/rfc/rfc4960.txt. [Online]. Available: http://www.rfc-editor.org/rfc/rfc4960.txt

[11] C. Zhang and V. Tsaoussidis, "Tcp-real: Improving real-time capabilities of tcp over heterogeneous networks," in *Proceedings of the 11th International Workshop on Network and Operating Systems Support for Digital Audio and Video*, ser. NOSSDAV '01. New York, NY, USA: Association for Computing Machinery, 2001, p. 189–198.

[12] M. Kim, J. Cloud, A. ParandehGheibi, L. Urbina, K. Fouli, D. Leith, and M. Medard, "Network Coded TCP (CTCP)," *arXiv e-prints*, p. arXiv:1212.2291, 2012.

[13] J. Iyengar and M. Thomson, "Quic: A udp-based multiplexed and secure transport," Working Draft, IETF Secretariat, Internet-Draft draft-ietf-quic-transport-27, February 2020, http://www.ietf.org/internet-drafts/draft-ietf-quic-transport-27.txt. [Online]. Available: http://www.ietf.org/internet-drafts/draft-ietf-quic-transport-27.txt

[14] A. Langley, J. Iyengar, J. Bailey, J. Dorfman, J. Roskind, J. Kulik, P. Westin, R. Tenneti, R. Shade, R. Hamilton, V. Vasiliev, A. Riddoch, W.-t. Chang, Z. Shi, A. Wilk, A. Vicente, C. Krasic, D. Zhang, F. Yang, F. Kouranov, and I. Swett, "The QUIC Transport Protocol," in *Proceedings of the Conference of the ACM Special Interest Group on Data Communication - SIGCOMM '17*. New York, New York, USA: ACM Press, 2017, pp. 183–196.

[15] M. Thomson and S. Turner, "Using tls to secure quic," Working Draft, IETF Secretariat, Internet-Draft draft-ietf-quic-tls-27, February 2020, http://www.ietf.org/internet-drafts/draft-ietf-quic-tls-27.txt. [Online]. Available: http://www.ietf.org/internet-drafts/draft-ietf-quic-tls-27.txt

[16] T. Dierks and E. Rescorla, "The transport layer security (tls) protocol version 1.2," Internet Requests for Comments, RFC Editor, RFC 5246, August 2008, http://www.rfc-editor.org/rfc/rfc5246.txt. [Online]. Available: http://www.rfc-editor.org/rfc/rfc5246.txt

[17] E. Rescorla, "The transport layer security (tls) protocol version 1.3," Internet Requests for Comments, RFC Editor, RFC 8446, August 2018.

[18] J. Iyengar and I. Swett, "Quic loss detection and congestion control," Working Draft, IETF Secretariat, Internet-Draft draft-ietf-quic-recovery-27, March 2020, http://www.ietf.org/internet-drafts/draft-ietf-quic-recovery-27.txt. [Online]. Available: http://www.ietf.org/internet-drafts/draft-ietf-quic-recovery-27.txt

[19] P. Wang, C. Bianco, J. Riihijärvi, and M. Petrova, "Implementation and performance evaluation of the quic protocol in linux kernel," in *Proceedings of the 21st ACM International Conference on Modeling, Analysis and Simulation of Wireless and Mobile Systems*, ser. MSWIM '18. New York, NY, USA: Association for Computing Machinery, 2018, p. 227–234.

[20] Q. De Coninck, F. Michel, M. Piraux, F. Rochet, T. Given-Wilson, A. Legay, O. Pereira, and O. Bonaventure, "Pluginizing quic," in *Proceedings of the ACM Special Interest Group on Data Communication*, ser. SIGCOMM '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 59–74.

[21] A. Čolaković and M. Hadžialić, "Internet of Things (IoT): A review of enabling technologies, challenges, and open research issues," *Computer Networks*, vol. 144, pp. 17–39, oct 2018.

[22] Nikshepa and V. Pai, "Survey on iot security issues and security protocols," *International Journal of Computer Applications*, vol. 180, no. 42, pp. 16–21, May 2018.

[23] E. Liri, P. K. Singh, A. B. Rabiah, K. Kar, K. Makhijani, and K. Ramakrishnan, "Robustness of IoT Application Protocols to Network Impairments," in *2018 IEEE International Symposium on Local and Metropolitan Area Networks (LANMAN)*. IEEE, jun 2018, pp. 97–103.

[24] Z. Shelby, K. Hartke, and C. Bormann, "The constrained application protocol (coap)," Internet Requests for Comments, RFC Editor, RFC 7252, June 2014, http://www.rfc-editor.org/rfc/rfc7252.txt. [Online]. Available: http://www.rfc-editor.org/rfc/rfc7252.txt

[25] P. Kumar and B. Dezfouli, "Implementation and analysis of QUIC for MQTT," *Computer Networks*, vol. 150, pp. 28–45, feb 2019.

[26] L. Eggert, "Towards securing the internet of things with quic," EasyChair Preprint no. 2434, EasyChair, 2020.

[27] H. Jiang, Y. Wang, K. Lee, and I. Rhee, "Tackling bufferbloat in 3G/4G networks," in *Proc. of the 2012 ACM conference on Internet Measurement Conference (IMC)*, 2012, pp. 329–342.