Mass Configuration of Heterogeneous IIoT Nodes: A Proposal and Experimental Evaluation

Darius Saif*, Ashraf Matrawy†
Carleton University, Department of Systems and Computer Engineering*, School of Information Technology†
Email: {dariussaif*, amatrawy†}@sce.carleton.ca

Abstract—**Securely managing Internet of Things (IoT) devices is a fundamental challenge. Management protocols for IoT must be scalable and extensible, especially when dealing with device heterogeneity. Because of device resource limitations in IoT and the sheer size of such networks, traditional management protocols are not well-suited. In this paper, we propose our HTTP/3 publish-subscribe architecture to meet the ends of *en masse* device configuration, on-boarding, and monitoring. We compared our solution to industry-standard protocols by collecting data from real traffic in networks of up to 500 clients. We found our solution to excel in ease of use and performance, while having slightly higher CPU utilization compared to the other protocols.**

Index Terms—**Management Protocols, IIoT, QUIC, MQTT**

I. Introduction

In an industrial setting, the Internet of Things (IoT) is capable of reducing operational overhead – thus maximizing productivity, sustainability, and bottom lines [1]. In order to reap these benefits however, expertise is needed to overcome the challenges of planning, configuration, and maintenance of any Industrial IoT (IIoT) solution in a lightweight way.

These networks can be complex: comprised of many resource-constrained devices from different vendors and supporting varying technologies. Devices may have limited In/Out (I/O) capabilities and can be deployed in unforgiving locations, further complicating diagnostic and maintenance tasks.

A definitive standard with matured and easy to use tools does not exist in the industry [2]. This is true of communication protocols as well as data models conveying configurations. Because of their wide-ranging applications and heterogeneity, a *de facto* taxonomy for describing an IIoT solution can also be difficult to produce [2].

The purpose of this paper is to address the challenges related to configuration and maintenance of heterogeneous IIoT systems in a scalable, extensible, and operator-oriented way. Specifically, our aim is to provide a means whereby devices can be i.) logically grouped together by common attributes, ii.) configured *en masse*, iii.) monitored for debugging or fault, and iv.) efficiently on-boarded into the network.

Contributions of this paper include an updated view on the state of IoT management protocols as well as the proposal of our solution, evaluated against industry-standard protocols in terms of performance, overhead, and ease of use.

We propose an IoT management protocol which is tooled from our QUIC-based HTTP/3 publish-subscribe (pub-sub) implementation [3]. Although we focus on an IIoT use case in this paper, we note that this protocol can be applied in IoT deployments without any loss for generality. Our implementation has been tuned specifically for IoT and tested on the Raspberry Pi Zero [4]. In this paper, we have adapted our implementation to execute configuration commands received on subscribe channels and subsequently provide diagnostic information through publishing channels.

In our evaluation, we took advantage of Docker to scale to network sizes of up to 500 clients. Our solution was found to be the most scalable in terms of performance, memory consumption, and data transmitted. On the other hand, clients required slightly more CPU compared to the other techniques.

The rest of this paper is organized as follows: Section II discusses management protocols in IoT. Related work in this area is highlighted in Section III. Our proposed architecture and evaluation approach are detailed in Sections IV and V, respectively. The experimental setup is defined in Section VI. The collected results are presented in Section VII. Finally, this paper's conclusions are drawn in Section VIII.

II. Management Protocols in IoT

Sinche *et al.* surveyed the landscape of management protocols and frameworks for IoT [2]. The full list of protocols they identified and reviewed is provided

TABLE I
IoT Management Protocols Surveyed by Sinche *et al.* [2]

Simple Network Management Protocol (SNMP) [5]
LowPAN Network Management Protocol (LMNP) [6]
Network Configuration Protocol (NETCONF) [7]
Representational State Transfer Configuration (RESTCONF) [8]
Lightweight Machine-to-Machine (LwM2M) [9]
Device Management Protocol (DM) [10]
CoAP Management Interface (CoMI) [11]
Message Queue Telemetry Transport (MQTT) [12]

in Table I. By their account, a management protocol must cover authentication, provisioning, configuration, monitoring, and maintenance.

Open challenges identified by Sinche *et al.* in this space include an absense of: i.) a unified taxonomy for describing IoT deployments, ii.) a widely accepted IoT management architecture, iii.) common data models for object representation, iv.) matured (and user-oriented) tools and resources, v.) rigorous security and privacy research.

In the literature surveyed by Sinche *et al.*, results showed that traditional management protocols like SNMP, NETCONF, and RESTCONF lacked compatibility across the spectrum of IoT devices. In cases where they *could* run, they did not scale well and had a large footprint on device resource consumption. Conversely, protocols like CoMI, LwM2M, and MQTT (although not specifically developed as a management protocol) were highlighted as state-of-the-art for IoT device management. Their investigation into industry platforms (like Microsoft Azure, Google Cloud, etc.) showed that LwM2M and MQTT were the most widely adopted. Both protocols have lightweight yet rich feature-sets and open-source availability, attributing to their popularity. CoMI had little use because its specification was still under draft, and no open-source implementations existed at the time.

Since Sinche *et al.*'s survey, CoMI was renamed to CORECONF [11] and its working group have a GitHub project[1] which does not appear to be ready for production use. Version 1.2 of LwM2M has also been released as of late 2020 [9]. In addition to the client-server model of previous versions, LwM2M v1.2 allows for a gateway to maintain client instances of devices which may not natively support LwM2M – thus, further addressing compatibility. Another key enhancement in v1.2 is support for additional transports: MQTT and HTTP.

[1]https://github.com/core-wg/comi

III. Related Work

Parmigiani and Dettmar [13] compared the performance LwM2M and MQTT in low-power wide area networks with different levels of security. The metrics monitored in their analysis were packets and bytes transmitted, packet loss, and energy consumption. The scenarios they studied were i.) initial connection, ii.) single client-to-server message, and iii.) steady-state update. Their testing took place over live LTE-M and NarrowBand-IoT (NB-IoT) networks: Orange Slovakia and Vodafone Italy, respectively. Version 1.0 of LwM2M (running over CoAP and UDP) was compared against MQTT (over TCP). A Raspberry Pi 3 connected point-to-point with a cellular modem ran both clients. Energy consumption was analyzed with a separate board. They found that LwM2M transmitted up to 82% fewer bytes than MQTT during connection setup. For larger payload sizes, MQTT used 20% less energy, but the opposite was true for smaller payloads. At steady state, LwM2M also consumed up to 40% less energy.

Eggert [14] showed that constrained devices like the Particle Argon and ESP32-DevKitC V4 development boards are capable of running QUIC, using the *Quant* and *picoTLS* libraries. Eggert analyzed storage space, battery consumption, memory, and CPU utilization of the boards by executing file downloads of 5KB. The major finding was that these boards had sufficient resources to run QUIC – and that with further optimizations, 16-bit processors could be supported as well.

Since then, other researchers have integrated QUIC with popular IoT protocols such as CoAP [15], Advanced Message Queuing Protocol (AMQP) [16], and MQTT [17]. To our knowledge, our paper is the first to consider QUIC as a transport-layer solution for IIoT device management.

IV. Proposed Architecture

Our proposed architecture for managing IIoT devices is a RESTful pub-sub system which makes use of the widely known APIs and semantics of HTTP/3. Publishing is accomplished by encoding a topic's name in an HTTP POST request's header and enclosing the message contents in the POST's body. Subscriptions are maintained through long-lived GET requests. Topics can also be explicitly created or deleted via PUT and DELETE requests, respectively.

Much akin to MQTT, clients connect to a broker which manages authentication and the dissemination of information. The broker accepts data published to a topic and forwards the information to all clients subscribed

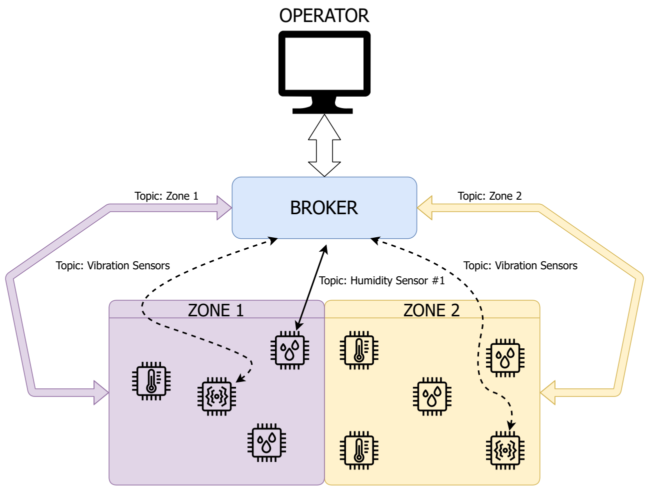

Fig. 1. Heterogeneous and Multi-Zone Network under Management

to said topic. Our solution only supports a Quality of Service (QoS) mode equivalent to MQTT's Assured Delivery [12]. We note that for the case of device management, Assured Delivery is necessary to guarantee that configurations are received reliably.

Our architecture has key features which make it robust for heterogeneous networks. QUIC transport-layer support offers integrated security, stream awareness, and proven performance gains over TCP and TLS [18]. Because QUIC operates in user-space, any device capable of UDP communication can run our solution without any dependencies. This makes our solution highly compatible and easily upgradeable. Moreover, the hierarchy of topics facilitates in logically grouping devices based on any set of features that an operator requires.

With the option to keep a specific topic's history on the broker, we are also able to streamline on-boarding of new devices entering the network. Once initialized, a new device can automatically apply previously committed configurations.

Figure 1 shows how our QUIC-based pub-sub architecture uses topics to realize a hierarchy of devices under management. Each device is capable of publishing information back to the broker node for diagnostic and monitoring purposes.

The QUIC-GO[2] v0.25.0 library powered our implementation. In our previous work [4], we tuned QUIC-GO for efficient performance and resource management within IoT. This included design of a high-watermark scheme for the transmission of QUIC MAX_STREAM frames as well as connection-level basic authentication (basicAuth). We found a key performance savings of one Round-Trip Time (RTT) in favor of our implementation when compared to MQTT-over-QUIC [17]. We also found that while our implementation was more resilient to Head-of-Line Blocking (HoLB), it used slightly more device resources.

For this paper, we adapted our implementation into a solution which can resolve many of the challenges of mass device configuration in IIoT. We also investigate scalability due to increasing network sizes, which was not previously tackled in our past works. We have extended our client-end program to treat received messages as executible configuration commands. The GOLANG `os/exec` package is leveraged to provide this functionality. Commands received over subscription channels are thus executed and the *stdout* and *stderr* are published back to the broker over a topic name unique to each device.

V. Evaluation Approach

Our evaluation goal was to compare our architecture to the industry leading IoT management protocols [2]. We focused on both quantitative (performance, overhead, and scalability) as well as qualitative (ease of use, and automation potential) factors towards configuration of IIoT devices *en masse*.

Neither CORECONF (CoMI) nor LwM2M were included in our evaluation. At the time of writing, CORECONF does not have a production-ready implementation and, thus, we were unable to consider it. Regarding LwM2M, we experimented with release 2.0.0-M11 of Eclipse's Leshan[3]: an open-source project which has been widely used in the literature [2], [13]. Leshan is written in Java and includes client and server libraries supporting LwM2M v1.1. In our experimentation, we found several factors working against this paper's goal of mass configuration of IIoT devices: i.) there is no way to logically cluster or group like devices and ii.) automation of management tasks via a script or other means does not seem to be possible. Our understanding is that these functionalities, critical to the goals of this paper, are neither explicitly supported as part of LwM2M's specification [9] nor are they offered by open-source implementations of the protocol.

In our evaluation, we have made two important assumptions which we find reasonable for an IIoT setting: devices are 1.) pre-installed with the relevant software executables, and 2.) pre-configured with the names of topics to subscribe to.

[2]https://github.com/quic-go/quic-go

[3]https://github.com/eclipse/leshan

We used version 5.8 of the net-SNMP package to provide comparison against a traditional management protocol. Commands were sent over SNMPv3 using AES encryption and SHA1 hashing. We also used MQTT-over-QUIC [17], which employs the same QUIC library as our solution. The clients and broker are adapted from Eclipse Paho[4] and VolantMQ[5], respectively. MQTT versions 3.1 and 3.1.1 are supported. We used Assured Delivery QoS in our testing. Transport layer tuning of MQTT-over-QUIC and our implementation were identical – full details of this can be found in [4].

Metrics collected from the experimentation with our solution, SNMP, and MQTT-over-QUIC are defined in Table II. Each experiment was run for 20 iterations and box and whisker plots were generated for statistical reporting. Packet captures were also taken during each iteration using tshark.

We note that, unlike SNMP, the pub-sub architectures under evaluation are connection-oriented. It is intended that such connections be long-lived for the use case of device management. Thus, we view the connection establishment as a one-time cost. For fairness, the metrics in Table II are not inclusive of connection establishment for either pub-sub solution.

VI. Experimental Setup

We believe that an industrial application would typically be more time sensitive, consist of many more nodes, and may span a larger physical area. Given these characteristics, we consider an IIoT deployment in this paper. Furthermore, envision a deployment consisting of devices located in a secure premises and operating over a private network. Ergo, our setup is realized as a controlled environment.

To model a heterogeneous network of devices, we consider three classes of devices deployed into two distinct physical zones. We envision scenarios where an operator may wish to configure devices of $type_x$ differently depending on the zone of deployment. Furthermore, the configuration of $type_x$ devices would be different from $type_y$ and $type_z$.

While we have experimented with our pub-sub system running on hardware IoT devices in a previous work [4], we have deployed our software in a Docker environment in this paper in order to scale to large network sizes. Docker version 23.0.1 ran on a Ubuntu 20.04.6 Virtual Machine (VM) which was allocated 20 cores and 32GB of memory. By default, a single Docker container can use as much host resources as the kernel scheduler allows[6] – which is why we continuously monitor CPU and memory during our experimentation and report these values in our results.

TABLE II
Evaluation Metrics

Execution Time	The time from when a configuration command is issued to each client executing it and reporting back to the broker
Bytes Transmitted	The total number of bytes transmitted to configure all devices (from the server end)
CPU Utilization	The amount of time the program occupied the container's CPU, reported as a percentage by *docker stats*
Memory Usage	The peak memory used by any one container, as reported by *docker stats*

In terms of software deployment, the broker code ran on the VM itself and Docker containers represented devices which were subject to management. Containers and the VM were all networked together using the bridged network mode.

We used Alpine[7] Linux as a base image for the Docker containers: a lightweight and secure distribution, typically used in embedded systems. The Docker container ID was used to uniquely identify each client. We used Docker Compose to describe the network topology, which dictated the topics each client would subscribe to. Docker Compose also instructed containers to run our software on boot-up.

NetEm was leveraged between the VM and containers to emulate a network profile according to the 3GPP Release 13's Extended Coverage GSM IoT (EC-GSM-IoT) standard [19]. It is a long range, low cost, and low energy technology which offers higher bandwidth and lower latency than NB-IoT. Compared to NB-IoT and LTE-M, EC-GSM-IoT is also more easily integrated with existing networks [20]. As such, rate throttling of 474 kilobits per second was applied on each container's uplink and downlink. An RTT of 700ms was also applied with a uniform distribution.

VII. Results

Figure 2 shows the execution time for each management protocol as the network size increased. One command was sent to each container, eliciting a 12-byte response. SNMP commands were sent in parallel, as

[4]https://github.com/pgOrtiz90/paho.mqtt.golang
[5]https://github.com/fatimafp95/volantmq_2
[6]https://docs.docker.com/config/containers/resource_constraints/
[7]https://www.alpinelinux.org/about/

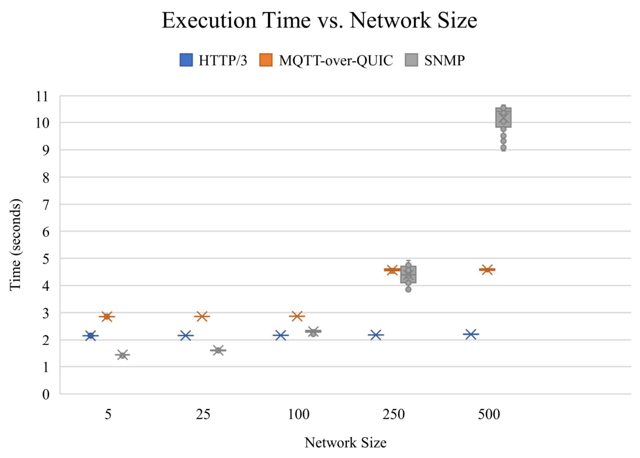

Fig. 2. Execution Time of Different Solutions

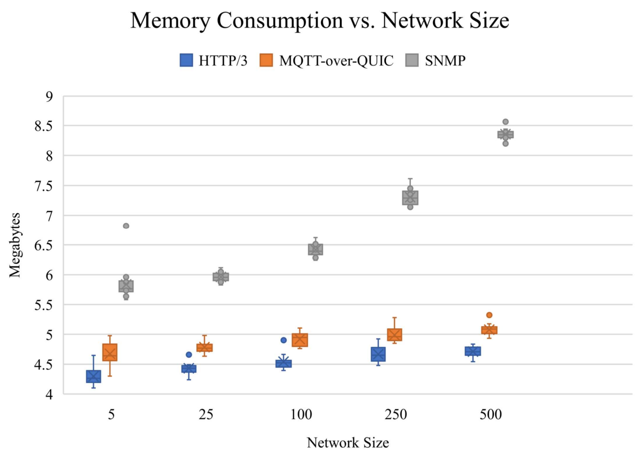

Fig. 4. Memory Consumption of Different Solutions

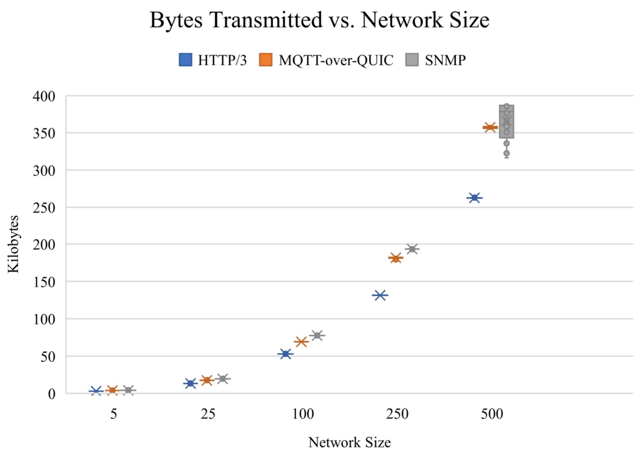

Fig. 3. Bytes Transmitted of Different Solutions

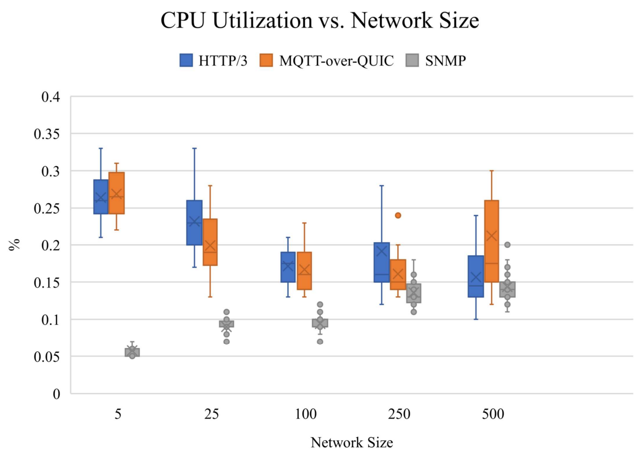

Fig. 5. CPU Utilization of Different Solutions

sequential execution took far too long. Even with parallel execution, SNMP was the least scalable, as the execution time disproportionately grew with the network size. Our HTTP/3 solution fared the best: having the most bounded performance and finishing almost 5 times faster than SNMP at the largest network size.

The bytes transmitted of each solution is shown in Figure 3. We had to tune the SNMP timeout option because we initially saw packet retransmissions before the client had a chance to respond, given the high RTT. SNMP produced the most bytes with slight variability, as indicated by the plot's whiskers and inter-quartile range. Our solution produced the fewest bytes.

Figures 4 and 5 show each management protocol's resource consumption in terms of memory and CPU, respectively. The command `docker stats` was used to collect memory and CPU data from every container in each iteration. We generated these whisker plots by taking the container with the peak utilization from each iteration. MQTT-over-QUIC and our solution were quite competitive in terms of memory and stayed relatively flat no matter the network size. Still, HTTP/3 consistently had the lowest average usage. At a network size of 500, SNMP required almost double the memory.

Although there was no discernible trend, SNMP was the most frugal in terms of CPU. We hypothesize that this disparity was due to SNMP's more primitive encryption and hashing methods: AES and SHA1. Both MQTT-over-QUIC and our solution used the `TLS_CHACHA20_POLY1305_SHA256` cipher-suite. Furthermore, we previously found static table QPACK to be an additional consumer of CPU in our solution [4]. QPACK [21] is HTTP/3's header compression mechanism, which aids in its resiliency against out-of-order data delivery. Lastly, we note that both pub-sub architectures experienced much more CPU variability.

VIII. Conclusions

In this paper, we sought out to address the challenge of scalable and extensible configuration of heterogeneous devices *en masse* in an IIoT setting. We proposed a pub-sub architecture fueled by HTTP/3 and compared our solution to industry-standard management protocols. Data collected from actual traffic with realistic network conditions shed light on performance, resource consumption, and other qualitative factors. We showed how our architecture can be used for device monitoring and onboarding as well.

In our experimentation, our proposed solution was the most scalable in terms of execution time. As the network size increased, its performance was bounded, and the broker was able to disseminate the data about 5 times faster than SNMP. Our solution also had the smallest memory and byte transfer footprint on the constrained devices, which is significant given their limited computing resources. While CPU consumption for each container didn't produce an apparent trend with increasing network sizes, HTTP/3 and MQTT-over-QUIC had higher average utilization. We noted that QUIC transport and, specifically for our HTTP/3 solution, header compression attributed towards this.

We conclude that our QUIC-based approach would be a good candidate for further exploration in IIoT configuration and management – especially for larger scale networks. The flexibility in grouping devices as an operator sees fit and ease of use are important practical factors. This is coupled with its low overhead and fast convergence time of pushing configurations to large numbers of devices. Future work items that we are interested in are i.) expounding on the device onboarding process through practical examples as well as ii.) dynamic device configuration based on device monitoring.

References

[1] E. Sisinni, A. Saifullah, S. Han, U. Jennehag, and M. Gidlund, "Industrial Internet of Things: Challenges, Opportunities, and Directions," *IEEE Transactions on Industrial Informatics*, vol. 14, no. 11, pp. 4724–4734, 2018.

[2] S. Sinche, D. Raposo, N. Armando, A. Rodrigues, F. Boavida, V. Pereira, and J. S. Silva, "A Survey of IoT Management Protocols and Frameworks," *IEEE Communications Surveys Tutorials*, vol. 22, no. 2, pp. 1168–1190, 2020.

[3] D. Saif and A. Matrawy, "A Pure HTTP/3 Alternative to MQTT-over-QUIC in Resource-Constrained IoT," in *IEEE Conference on Standards for Communications and Networking (CSCN)*, pp. 36–39, 2021.

[4] D. Saif and A. Matrawy, "An Experimental Investigation of Tuning QUIC-Based Publish-Subscribe Architectures in IoT," *IEEE Internet of Things Journal*, pp. 1–1, 2023. Early Access.

[5] M. Fedor, M. L. Schoffstall, J. R. Davin, and D. J. D. Case, "Simple Network Management Protocol (SNMP)." RFC 1157, May 1990.

[6] H. Mukhtar, K. Kang-Myo, S. A. Chaudhry, *et al.*, "LNMP-Management Architecture for IPv6 based Low-Power Wireless Personal Area Networks (6LoWPAN)," in *NOMS 2008 - IEEE Network Operations and Management Symposium*, pp. 417–424, 2008.

[7] R. Enns, M. Björklund, A. Bierman, and J. Schönwälder, "Network Configuration Protocol (NETCONF)." RFC 6241, June 2011.

[8] A. Bierman, M. Björklund, and K. Watsen, "RESTCONF Protocol." RFC 8040, Jan. 2017.

[9] "Lightweight Machine-to-Machine Technical Specification v1.2," Standard OMA-TS-LightweightM2M_Core-V1_2-20201110-A, Open Mobile Alliance, November 2020.

[10] "Device Management Protocol v2.0," Standard OMA-TS-DM_Protocol-V2_0-20160209-A, Open Mobile Alliance, February 2016.

[11] M. Veillette, P. V. der Stok, *et al.*, "CoAP Management Interface (CORECONF)," Internet-Draft draft-ietf-core-comi-12, Internet Engineering Task Force, Mar. 2023. Work in Progress.

[12] A. Banks, E. Briggs, K. Borgendale, and R. Gupta, "MQTT Version 5.0," Standard mqtt-v5.0, OASIS, March 2019.

[13] A. Parmigiani and U. Dettmar, "Comparison and Evaluation of LwM2M and MQTT in Low-Power Wide-Area Networks," in *IEEE International Conference on Internet of Things and Intelligence Systems (IoTaIS)*, pp. 8–14, 2021.

[14] L. Eggert, "Towards Securing the Internet of Things with QUIC," in *Workshop on Decentralized IoT Systems and Security (DISS)*, 2020.

[15] R. Herrero, "Analysis of QUIC Transported CoAP," *SN Computer Science*, vol. 2, no. 2, p. 62, 2021.

[16] F. Iqbal, M. Gohar, H. Karamti, W. Karamti, S.-J. Koh, and J.-G. Choi, "Use of QUIC for AMQP in IoT networks," *Computer Networks*, vol. 225, p. 109640, 2023.

[17] F. Fernández, M. Zverev, P. Garrido, J. R. Juárez, J. Bilbao, and R. Agüero, "And QUIC Meets IoT: Performance Assessment of MQTT over QUIC," in *IEEE International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob)*, pp. 1–6, 2020.

[18] J. Iyengar and M. Thomson, "QUIC: A UDP-Based Multiplexed and Secure Transport." RFC 9000, May 2021.

[19] W. Ayoub, A. E. Samhat, F. Nouvel, M. Mroue, and J.-C. Prévotet, "Internet of Mobile Things: Overview of LoRaWAN, DASH7, and NB-IoT in LPWANs Standards and Supported Mobility," *IEEE Communications Surveys Tutorials*, vol. 21, no. 2, pp. 1561–1581, 2019.

[20] Ericsson, "The cellular Internet of Things – technologies, standards and performance." https://www.ericsson.com/en/blog/2017/12/the-cellular-internet-of-things--technologies-standards-and-performance, 2017. Accessed: 2023-09-22.

[21] C. B. Krasic, M. Bishop, and A. Frindell, "QPACK: Field Compression for HTTP/3." RFC 9204, June 2022.