

A Pure HTTP/3 Alternative to MQTT-over-QUIC in Resource-Constrained IoT

Darius Saif*, Ashraf Matrawy†

Carleton University, Department of Systems and Computer Engineering*, School of Information Technology†

Email: Dariussaif*, Amatrawy†@sce.carleton.ca

Abstract—In this paper, we address the issue of scalable, interoperable, and timely dissemination of information in resource-constrained IoT. Scalability is addressed by adopting a publish-subscribe architecture. To address interoperable and timely dissemination, we propose an HTTP/3 (H3) solution that exploits the wide-ranging improvements made over H2. We evaluated our solution by comparing it to a state-of-the-art work: MQTT-over-QUIC. Because QUIC and H3 have undergone standardization in tandem, we hypothesized that H3 would take better advantage of QUIC transport than an MQTT mapping would. Performance, network overhead, and device overhead were investigated for both protocols. Our H3-based solution satisfied our timely dissemination requirement by offering a key performance savings of 1 RoundTrip Time (RTT) for publish messages to arrive at the broker. In IoT networks, with typically high RTT, this savings is significant. On the other hand, we found that MQTT-over-QUIC put marginally less strain over the network.

Index Terms—IoT, QUIC, HTTP/3, MQTT, Standards

I. INTRODUCTION

To realize an Internet of Things (IoT), many low-cost wireless devices must be deployed in a variety of locations. These sensory devices are often limited in computing resources, storage, and battery. Because of these characteristics, it is imperative that lightweight and energy efficient communication standards be used between sensors themselves, gateways, and fog networking nodes [1]. The heterogeneous nature of IoT makes interoperability ever-more important, giving rise to the Web of Things (WoT) [2] for protocols like MQTT and HTTP.

QUIC, an emerging protocol, has become an area of interest in IoT. It is a cross-layer standard that offers integrated security, reduced connection establishment time, and advanced stream multiplexing. It has been found that QUIC can outperform TCP+TLS in networks with higher loss and Round-Trip Time (RTT) [3]. Such conditions are present in resource-constrained IoT. Researchers [4], [5] have thus mapped MQTT to run over QUIC and evaluated it against MQTT-over-TCP.

QUIC, however, was designed and tuned to carry HTTP traffic, forming the basis of HTTP/3 (H3). Given that, our research objective is to determine whether a pure H3 publish-subscribe (pub-sub) architecture can provide even further gains than MQTT-over-QUIC. We hypothesize that, because of the vertical integration of H3 and QUIC, H3 is better positioned to take full advantage of QUIC transport, instead of MQTT.

To this end, the contributions of this paper are i.) addressing scalable, interoperable, and timely dissemination of information in resource-constrained IoT networks, and ii.) comparison in three categories between MQTT-over-QUIC and against an H3-enabled pub-sub application of our creation.

II. RELATED WORK

Saif *et al.* [6] compared web-page performance of H3 draft 27 and H2 in more traditional networks. A suite of metrics was used to shed light on Quality of Experience (QoE). It was shown that H3 was more impervious to network impairment, even though H3's QoE scoring was marginally worse than H2.

Lars Eggert [7] showed that IoT devices are capable of running QUIC. Two development boards with 32-bit microprocessors were examined running QUIC via *Quant* and *picoTLS*. Storage space, battery consumption, as well as memory and CPU usage on the boards were monitored while various QUIC transactions (downloads) took place. It was found that QUIC required about 63KB of flash, 16KB of heap memory, 4KB of stack memory, and 0.9J energy per transaction. This was deemed sufficient and that, with further optimizations, QUIC could run on 16-bit processors as well. This work, unlike ours, did not include H3, however.

Kumar and Dezfouli [4] outfitted Chromium's GQUIC to carry MQTT. Because MQTT and QUIC both run in user-space, they had written inter-process communication APIs and redesigned various data structures to make their solution functional. Raspberry Pi 3 Model B's were used as the subscribers, publishers, and broker. Three network categories were used in their testing: wired, wireless, and long distance. During connection establishment, MQTT-over-QUIC reduced packets exchanged with the broker by up to 56.25%. To test Head-of-Line Blocking (HoLB), packets were randomly dropped. MQTT (over TCP) was shown to have higher latency in every network configuration. Memory and processor utilization of the broker were also monitored for half-open connections. Although MQTT-over-QUIC used slightly more resources, it relinquished up to 83.24% and 50.32% more processor and memory usage respectively compared to MQTT. Lastly, MQTT-over-QUIC was found to be more resilient to when the broker's IP changed mid-transaction.

Fernandez *et al.* [5] had also ported MQTT over to QUIC and compared it against TCP. A pure GO implementation [8] of QUIC was integrated with Eclipse Paho and VolantMQ. These packages were used as the publisher and broker, respectively. Their implementation was made available open-source. They used NS-3 to emulate different network profiles: Wi-Fi, 4G, and satellite. On each profile, testing of single and multiple MQTT connections were investigated. In the single connection tests, the time delta starting from 1000 packets being published to the last of such messages being pushed to a subscriber

was recorded. They found that QUIC outperformed TCP in every configuration, by up to 40%. Connection establishment time was the focus of the multiple connection testing. After publishing one message, the connection to the publisher and broker was closed. Afterwards, another message was published on a new connection. QUIC proved to perform slightly better than TCP and also exhibited less variation.

Rather than creating specific conversion code and changing data structures to support MQTT over QUIC transport – which can introduce sub-optimal performance – our work considers a pure H3 pub-sub implementation. Our implementation was compared against Fernandez *et al.*'s [5] MQTT-over-QUIC.

III. H3 PUBLISH-SUBSCRIBE ARCHITECTURE

Previous versions of HTTP were deemed to be inappropriate for IoT due to the nature of its connection management [9]. Connection setup times were lengthy and subject to HoLB. Both of these factors were greatly reduced in H3 because of its 0 or 1-RTT establishment times in addition to QUIC's knowledge of streams in the transport layer. Because of these significant gains, we explore the viability of H3 pub-sub.

HTTP was not built for pub-sub, but its APIs are versatile enough to handle it and is also highly interoperable. For the purposes of this paper, an HTTP pub-sub architecture was adapted from Drift [10]. The way Drift handles pub-sub functions over HTTP is shown in Table I. In each case below, the client encodes the *topic* name into the URL.

Topic Exists	HEAD	Server returns 200 if it exists, else 404.
Create Topic	PUT	Server sanity checks the topic name and creates the topic if checks pass.
Delete Topic	DELETE	Server cleans up topic data and unsubscribes all parties from the topic.
Publish to Topic	POST	Client encodes topic name in URL and includes data in message body, server stores information and pushes to subscribers.
Subscribe to Topic	GET	Client spawns a listener for incoming events, server appends node to the list of nodes subscribed to the topic.

TABLE I
H3 PUBLISH-SUBSCRIBE SEMANTIC MAPPINGS

We retooled Drift to run over H3. Drift was chosen due to its few dependencies and open-source code written in GO. Since Fernandez *et al.*'s [5] implementation was also executed with GO, this made for a more fair comparison. In our testing, the underlying QUIC code [8] was common to H3 pub-sub and MQTT-over-QUIC.

Firstly, we scrapped the entirety of Drift's *transport* package, because it was based on a GO module of H2 [11]. We fully replaced and upgraded this functionality with the equivalents from QUIC-GO's H3 stack package. These changes posed consequences in Drift's *client* package. This package housed the methods used for mapping pub-sub concepts to HTTP APIs, as outlined in Table I. Minor amendments to these methods were necessary in order for us to better accommodate H3 APIs, due to differences in implementation between the H3 and original H2 modules.

Likewise, we also upgraded the *server* package code to run H3. This was taken care of by adding an H3 listener call from QUIC-GO into the *server* package along with a *Cookoo* [12] based HTTP handler. Cookoo, a chain-of-command GO framework, was used for the management of topics and publishers/subscribers on the broker entity.

IV. NETWORK MODELLING

NetEm [13] was leveraged to meet this paper's focus on realistic resource-constrained IoT network conditions. Because Eggert established the validity of running QUIC on IoT devices in [7], a proof-of-concept approach is used in this paper through the means of emulation. Delay and rate throttling rules were applied to outgoing packets on the publisher and broker network interfaces, as shown in Figure 1:

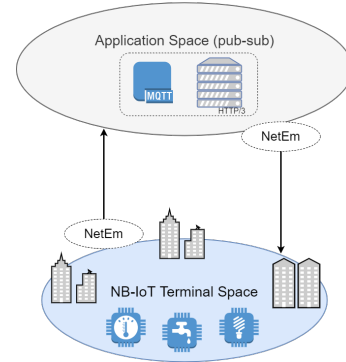


Fig. 1. Emulated Network Topology

The 3GPP's Narrowband-IoT (NB-IoT) network parameters were used in this paper. Given NB-IoT's high RTT and modest data rates [14], such attributes make it a suitable choice for resource-constrained IoT. The network parameters [14] of NB-IoT's second generation standard were incorporated into our environment. These parameters are summarized in Table II:

Downlink Rate	127 kbit/s
Uplink Rate	159 kbit/s
Round-Trip Time	2 seconds

TABLE II
NB-IoT CAT NB2 NETWORK PARAMETERS

V. EXPERIMENTAL SYSTEM SETUP

Ubuntu 18.04.4 (kernel 4.15.0-88) Virtual Machines (VMs) were deployed in Oracle's VirtualBox for the roles of publishers, subscribers, and the broker. All VMs were allocated 4 processors and 6GB of memory. To allow for VM-to-VM communications, they were configured using a Host-only Ethernet Adapter. The PC hosting the VMs was a 64-bit Windows 10 machine (24GB RAM and Intel i5-836U CPU).

In our setup, default tuning of QUIC-GO [8] release v0.20.1 powered *both* the H3 and MQTT-over-QUIC implementations. Transport specific factors, like 0-RTT, are expected to equally affect both and are therefore not considered. Thus, we focus on the application layer differences between H3 and MQTT.

The setup adhered to the IETF draft 29 QUIC standard [15] and the *quic_transport_parameters* in each protocol's

Client Hello were identical. Both also used CUBIC congestion control [16] and path MTU discovery. MQTT-over-QUIC used basic authentication. Dynamic table QPACK was not used for H3; only the static table and encoded string literals were employed. QPACK [17] is H3's method of header compression.

NetEm rules provisioned on the broker accounted for the downlink rate and half the RTT value, in accordance to the NB-IoT parameters of Table II. Conversely, the uplink rate and half the RTT value from Table II were imposed with NetEm on the publisher/subscriber's network interface card.

VI. RESULTS

A. Performance Indicators

1) *Time to First Data Frame*: This was the delta between the publisher's initial Client Hello and its first data frame (H3 DATA FRAME or MQTT PUBLISH packet, respectively). As shown in Figure 2, this measurement was independent of the size of the message. What was striking, however, was that MQTT-over-QUIC required one additional RTT for the publish message content to be sent – delaying the message by seconds.

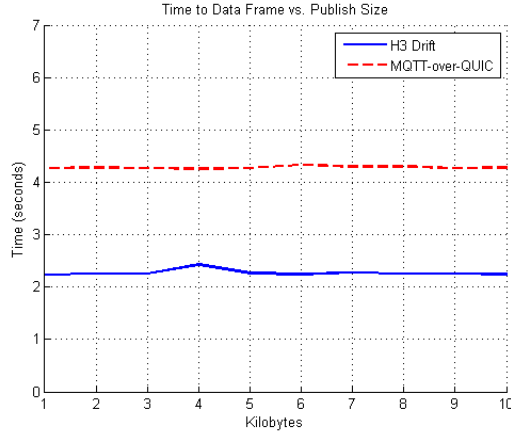


Fig. 2. Time to First Data Frame with Increasing Message Size

This was due to MQTT's application level connection setup overhead when any form of authentication is used. The first packet a publisher must send to the broker is a CONNECT packet. As per MQTT's specification "If a Client sets an Authentication Method in the CONNECT, the Client MUST NOT send any packets other than AUTH or DISCONNECT packets until it has received a CONNACK packet" [18]. This added an additional RTT which was not necessary in our pure H3 pub-sub implementation.

2) *Time to Connection Close & Throughput*: Five publisher clients, staggered by one second, each sent one 1KB message to the broker. Packet loss falling within the typical bounds for NB-IoT [19] was applied with a uniform distribution.

All of the H3 transactions completed by the 8 second mark, whereas MQTT-over-QUIC lagged roughly 1.4 seconds behind. Also, H3 achieved a 24% higher peak throughput at the connection level but MQTT-over-QUIC's peak throughput for the aggregate of connections was 2.88% higher. Figure 3 shows the connection-level plots for each implementation, with data points aggregated over 200ms intervals.

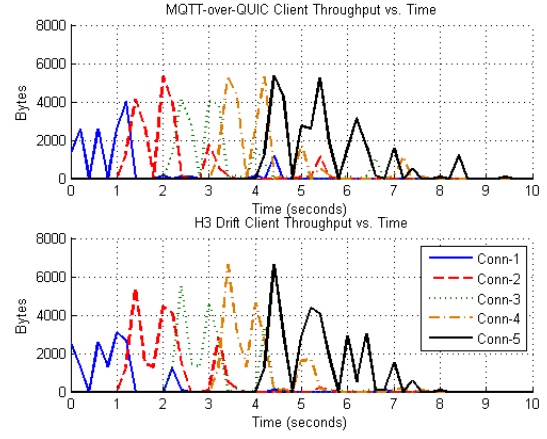


Fig. 3. Interleaved Clients Publishing via MQTT and H3

B. Network Overhead

The total amount of bytes and packets exchanged between the broker and publisher was examined for messages of increasing size (1 to 10KB). Figure 4 shows that H3-based transactions required more data exchange than MQTT-over-QUIC – meaning higher network overhead. Still, the two implementations were quite competitive: the average increase of bytes transmitted was found as 3.23%. Similarly, the number of packets transmitted for H3 transactions were, on average, 11% more than MQTT-over-QUIC.

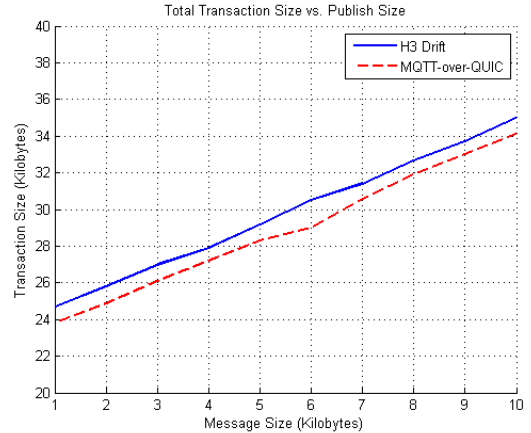


Fig. 4. Transaction Size vs. Increasing Message Size

Because of NB-IoT's large RTT, the default tuned QUIC transport was rather aggressive. By the time the broker's response to the publisher's initial establishment request arrived, six additional retry *Client Hello* packets were sent by the publisher. This elicited further server responses, attributing to unnecessary data exchange. Such occurred in every transaction for both implementations (nullifying its effects), which led to the somewhat inflated numbers in Figure 4.

C. Device Overhead

Keeping in mind the intended IoT environment, resource consumption was given careful consideration. To this end, the Linux command *ps* was employed in order to give further insights. For the duration of each publish process, *ps* was sampled every 100ms for CPU utilization.

The CPU consumption is reported as a percentage – that is, the ratio of CPU time used to the process’ duration. Figure 5 shows that the CPU usage of H3 was not only unpredictable, but also much higher than that of MQTT-over-QUIC.

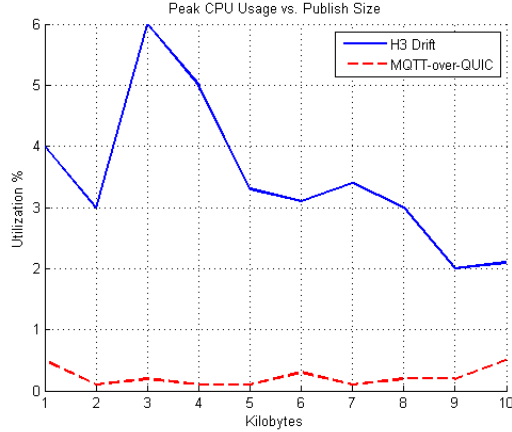


Fig. 5. Peak CPU Usage with Increasing Message Size

A GO profiler, *pprof*, was used to investigate memory. The total number of bytes allocated in heap memory were investigated for both H3 and MQTT-over-QUIC. The profiler’s *MemProfileRate* was driven to 0 in order to record all allocated blocks. Results were collected from 10KB publish messages and the *top* command was used to identify the 5 most memory-hungry functions. These results are shown in Figures 6 and 7:

```
Showing nodes accounting for 1415.75kB, 48.94% of 2893.06kB total
Dropped 123 nodes (cum <= 14.46kB)
Showing top 5 nodes out of 60
```

flat	flat%	sum%	cum	cum%	
361.78kB	12.51%	12.51%	1193.60kB	41.26%	crypto/x509.parseCertificate
313.75kB	10.84%	23.35%	313.75kB	10.84%	bytes.makeSlice
265.48kB	9.18%	32.53%	265.48kB	9.18%	reflect.unsafe_NewArray
244.89kB	8.46%	40.99%	1319.42kB	45.61%	encoding/asn1.parseField
229.84kB	7.94%	48.94%	263.97kB	9.12%	encoding/pem.Decode

Fig. 6. Top 5 Memory Consumers for H3

The *flat* columns represent memory that was allocated, and held, by that particular function whereas *cum* columns are inclusive of the function’s children. The *sum%* column is the summation of the current, and all previous, *flat%* values.

```
Showing nodes accounting for 86.19kB, 31.58% of 272.94kB total
Dropped 357 nodes (cum <= 1.36kB)
Showing top 5 nodes out of 214
```

flat	flat%	sum%	cum	cum%	
36.03kB	13.20%	13.20%	36.03kB	13.20%	regexp.(*bitState).reset
15.47kB	5.67%	18.87%	15.47kB	5.67%	github.com/lucas-clemente/quic-go/internal/wire.int.0.func1
13.50kB	4.95%	23.81%	13.50kB	4.95%	crypto/sha256.New
11.10kB	4.09%	27.90%	23.91kB	8.76%	crypto/hmac.New
10.03kB	3.68%	31.58%	10.47kB	3.84%	github.com/eclipse/paho.mqtt.golang.(*client).Publish

Fig. 7. Top 5 Memory Consumers for MQTT-over-QUIC

H3 used approximately 10 times the memory as MQTT-over-QUIC. The cause of this disparity is due to H3’s TLS configuration. Functions for decoding and parsing the x509 certificate and its fields account for the vast majority of this gap. The x509 certificate overhead also largely attributed to the extra CPU. *pprof* was run in CPU profiling mode for H3 and it uncovered that 37.5% of H3’s time occupying the CPU went towards dealing with the x509 certificate.

VII. CONCLUSIONS

In this paper, an H3 pub-sub alternative to MQTT-over-QUIC for resource-constrained IoT was designed and ex-

plored. This implementation retained features akin to MQTT and was quite competitive in terms of network overhead. A savings of 1-RTT during publishing went to H3’s favor because of its cross-layer integration with QUIC. The significance of this result is that, in IoT networks, messages can be received (and pushed to subscribers) seconds faster.

H3 was more taxing than MQTT-over-QUIC on the end device – for resource-constrained IoT, this poses a clear trade-off. It dipped further into the device’s CPU and memory. Code profiling with *pprof* in our test environment found this was primarily due to implementation-specific factors.

QUIC/H3 have proven themselves as having strong potential for scalable, interoperable, and timely communication for IoT. In that vein, points of our future work include i.) extending our test environment for more realistic conditions, ii.) non-default stack tuning, and iii.) exploring alternatives to x509 for H3.

REFERENCES

- [1] J. Dizdarević, F. Carpio, A. Jukan, and X. Masip-Bruin, “A Survey of Communication Protocols for Internet of Things and Related Challenges of Fog and Cloud Computing Integration,” *ACM Computing Surveys (CSUR)*, vol. 51, no. 6, pp. 1–29, 2019.
- [2] D. Raggett, “The Web of Things: Challenges and Opportunities,” *Computer*, vol. 48, no. 5, pp. 26–32, 2015.
- [3] A. Langley, A. Riddoch, A. Wilk, A. Vicente, C. Krasic, D. Zhang, F. Yang, *et al.*, “The QUIC Transport Protocol: Design and Internet-Scale Deployment,” in *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, pp. 183–196, 2017.
- [4] P. Kumar and B. Dezfouli, “Implementation and Analysis of QUIC for MQTT,” *Computer Networks*, vol. 150, pp. 28–45, 2019.
- [5] F. Fernández, M. Zverev, P. Garrido, J. R. Juárez, J. Bilbao, *et al.*, “And QUIC meets IoT: Performance Assessment of MQTT over QUIC,” in *International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob)*, pp. 1–6, IEEE, 2020.
- [6] D. Saif, C.-H. Lung, and A. Matrawy, “An Early Benchmark of Quality of Experience Between HTTP/2 and HTTP/3 using Lighthouse,” in *International Conference on Communications (ICC)*, IEEE, 2021.
- [7] L. Eggert, “Towards Securing the Internet of Things with QUIC,” in *Workshop on Decentralized IoT Systems and Security (DISS)*, 2020.
- [8] “A QUIC Implementation in Pure Go,” <https://github.com/lucas-clemente/quic-go>. Accessed: 2019-12-03.
- [9] T. Yokotani and Y. Sasaki, “Comparison with HTTP and MQTT on Required Network Resources for IoT,” in *International Conference on Control, Electronics, Renewable Energy and Communications (IC-CEREC)*, pp. 1–6, IEEE, 2016.
- [10] “Drift: An HTTP/2 Pub/Sub service,” <https://github.com/technosophos/drift>. Accessed: 2020-12-14.
- [11] “HTTP/2 Package,” <https://pkg.go.dev/golang.org/x/net/http2>. Accessed: 2021-09-11.
- [12] “Cookoo: A Chain of Command Framework Written in GO,” <https://go.gosourc.com/net+/master/http2/>. Accessed: 2021-09-11.
- [13] S. Hemminger *et al.*, “Network Emulation with NetEm,” in *Linux Conference Australia*, vol. 844, Citeseer, 2005.
- [14] A. D. Zayas and P. Merino, “The 3GPP NB-IoT system architecture for the Internet of Things,” in *International Conference on Communications Workshops (ICC Workshops)*, pp. 277–282, IEEE, 2017.
- [15] M. Bishop, “Hypertext Transfer Protocol Version 3 (HTTP/3),” Internet-Draft draft-ietf-quic-http-29, Internet Engineering Task Force, 2020.
- [16] S. Ha, I. Rhee, and L. Xu, “CUBIC: a new TCP-friendly high-speed TCP variant,” *ACM SIGOPS operating systems review*, vol. 42, no. 5, pp. 64–74, 2008.
- [17] C. B. Krasic, M. Bishop, and A. Frindell, “QPACK: Header Compression for HTTP/3,” Internet-Draft draft-ietf-quic-qpack-21, Internet Engineering Task Force, Feb. 2021.
- [18] “MQTT Version 5.0,” Standard, OASIS, <https://docs.oasis-open.org/mqtt/mqtt/v5.0/mqtt-v5.0.pdf>, 2019.
- [19] “NB-IoT, LoRaWAN, Sigfox: An up-to-date Comparison,” Whitepaper version 1.3, Deutsche Telekom AG, Friedrich-Ebert-Allee 140, 53113 Bonn, Germany, April 2021.