Performance Evaluation of QUIC vs TCP for Cloud Control Systems

Peng, Haorui; Tärneberg, William; Fitzgerald, Emma; Kihl, Maria

Published in:
The 31st International Conference on Software, Telecommunications and Computer Networks - SoftCOM 2023

DOI:
10.23919/SoftCOM58365.2023.10271592

2023

Document Version:
Early version, also known as pre-print

Link to publication

Citation for published version (APA):
Peng, H., Tärneberg, W., Fitzgerald, E., & Kihl, M. (2023). Performance Evaluation of QUIC vs TCP for Cloud Control Systems. In *The 31st International Conference on Software, Telecommunications and Computer Networks - SoftCOM 2023* IEEE - Institute of Electrical and Electronics Engineers Inc.. https://doi.org/10.23919/SoftCOM58365.2023.10271592

Total number of authors:
4

Performance Evaluation of QUIC Vs. TCP for Cloud Control Systems

Haorui Peng, William Tärneberg, Emma Fitzgerald, Maria Kihl
Department of Electrical and Information Technology, Lund University, Lund, Sweden
{haorui.peng, william.tarneberg, emma.fitzgerald, maria.kihl}@eit.lth.se

***Abstract*—QUIC is a UDP-based transport layer protocol developed by Google that is designed to deliver lower latency performance than TCP. However, its performance has been found to vary across different studies due to discrepancies in targeted applications and experimental configurations. Therefore, the protocol evaluation is heavily dependent on the QoS requirements of the target application. In this paper, we evaluate the performance of QUIC for cloud control systems by comparing it with TCP and TCP-based protocols. We performed the evaluation under two different cloud environments: a private edge cluster and public cloud. Different networks were utilized under multiple scenarios, including 5G, Ethernet LAN and Internet. Our objective is to determine if QUIC can meet the QoS requirements of a cloud control system. By focusing on this specific use case, we aim to provide a more comprehensive understanding of the potential benefits and limitations of QUIC on cloud control systems. The results of our evaluation suggest that the performance of different protocols varies based on the characteristics of different control systems. However, QUIC offers more advantages in deploying and enhancing time-critical systems.**

***Index Terms*—Cloud control systems, Transport layer protocols, QUIC**

I. Introduction

Quick UDP Internet Connections (QUIC) [1] is a transport layer protocol that was initially proposed and developed by Google since 2012 and aims to provide an alternative to TCP for deploying today's web application. It has been globally enabled on Google services such as YouTube and Gmail [2], as well as Meta services such as Facebook and Instagram. It is also supported by today's major browsers such as Chrome, Microsoft Edge, Firefox and Safari.

QUIC was standardized by The Internet Engineering Task Force (IETF) at 2021 [1], and later, HTTP3, an HTTP protocol that relies on QUIC was standardized in 2022 [3]. According to the report from Google, QUIC has achieved 10% reduction in page loading time for their properties and 30% reduction in rebuffers for video services [4]. Meta also reported that, with QUIC, Facebook users have experienced a 6% reduction in request errors and 20% tail latency reduction [5].

Considering the low latency benefits, we believe that QUIC could be a more appropriate protocol for Cloud Control Systems (CCSs) [6]. CCSs are cyber-physical systems that deploy the controller in a public or edge cloud, and the controller communicates with the control plant through a network. By utilizing QUIC, CCSs could potentially benefit from reduced latency and improved network performance.

Although there are conflicting results regarding the performance of QUIC and its potential to replace TCP/TLS in web applications [7]–[11], most studies have shown that QUIC outperforms TLS/TCP in less reliable networks and with smaller packet sizes [7]–[9]. According to [12], these discrepancies may be due to differences in application settings and protocol implementation used in the experiments. Overall, while QUIC has shown promise in reducing latency and improving performance in certain applications and network conditions, its benefits and limitations are highly dependent on the specific use case and network environment. Further research and evaluation are necessary to fully understand the potential benefits and drawbacks of QUIC in various contexts. An application-oriented evaluation of transport layer protocols is crucial for critical applications like CCSs and their developers, as such a system requires low-latency and reliable end-to-end communication.

This paper presents an application specific evaluation of QUIC for CCSs, which has not been previously explored. We compare QUIC with TCP and evaluate the performance of HTTP3 (based on QUIC) versus HTTP1.1 (with TLS implemented) for a CCS while considering its specific Quality of Service (QoS) requirements. We conduct experiments in scenarios with different cloud environments and communication networks to investigate whether HTTP3/QUIC is a better choice than legacy TCP-based protocols for CCS with specific QoS requirements.

II. Targeted System

This section provides an overview of the system targeted for our protocol evaluation, namely CCS, and highlights the technical differences among the protocols that have an impact on system performance.

A. QUIC, TCP and HTTP

QUIC has evolved through multiple versions over the past decade, and the current major QUIC implementations available in various programming languages conform to the IETF standard, which is also the QUIC version we address in this paper. The application layer protocol based on QUIC is standardized as HTTP3. In contrast to secured HTTP1.1, which is built on top of a TLS layer and TCP transport protocol, HTTP3 relies on QUIC for all functionalities, including cryptographic and transport handshake, and multiplexing. The multistreaming mechanism of QUIC and HTTP3 facilitates concurrent data streams over a singular QUIC/UDP connection. This feature mitigates head-of-line blocking and reduces connection establishment and management latency, which contrasts with HTTP1.1 that requires multiple concurrent TCP connections for multiplexing.

Based on the targeted cloud control system in our work, which involves a cloud application that requires full network stack implementation, this paper focuses on comparing the two HTTP protocols, namely HTTP3 based on QUIC/UDP and HTTP1.1 based on TLS/TCP. Our evaluation mainly revolves around the performance and behavior of these protocols in the context of a cloud control system, and highlights their potential advantages.

B. Cloud Control System

A cloud control system is a cyber-physical system where the plant is a physical device such as robotic arms, autonomous vehicles and production lines, but the the controller of the system is implemented in the cloud. In a CCS, the control loop is separated by a network. In combination with mobile communication such as 5G, this type of systems are evolving rapidly in the Industry 4.0 era to enable automation in manufacture, mining, etcetera [13], [14]. The control process operates at a frequency appropriate for the specific plant, where in each period, the plant sends its current state as a request to the cloud-based controller, which computes a control signal and synchronously replies with it. The control signal is then applied to the plant for actuation. The length of the period is also called the *sampling rate* of a cloud control system, deciding the interval of subsequent requests made by the plant.

The key difference between a CCS and a local control system is the presence of network latency, which can result in outdated control signals being applied to the plant. This can negatively impact the performance and safety of the process. The frequency of requests from the plant to the controller depends on the plant model, and affects the design of the controller. Processes that require higher request frequencies for updating plant states and receiving control signals require lower network latency and jitter to meet their dynamics.

Based on our previous study on CCSs [15], [16], where HTTP1.1 was mainly implemented in the systems, the payload size containing the plant state or control signal is typically small, ranging from 200-300 bytes. This is much smaller compared to payload sizes of web page or video downloads. In most cases of CCSs, there is no need for segmentation of the payload as both QUIC and TCP support payload sizes up to 1350 bytes and 1500 bytes.

III. Experiment Setup

In this section, we describe our experimental setup for evaluating the performance of different protocols in CCSs, as well as the performance metrics we consider. Specifically, we compare QUIC with TCP as a transport protocol. Meanwhile, since HTTP is still the most prevalent application protocol used in cloud applications, we also evaluate the performance of HTTP3 and HTTP1.1 over TLS/TCP. Thus, in this paper, four servers are deployed using different protocols: (i) single stream QUIC (ii) TCP only (without TLS) (iii) HTTP over QUIC (HTTP3) and (iv) HTTP1.1 over TLS/TCP. Note that in the first two cases, no application protocols are implemented in the communication, so the packet size and network overhead would be smaller than case (iii) and (iv).

A. Evaluation Setup

To evaluate the performance of the protocols, we created several client-server pairs using Golang, communicating over different protocols. The client is the plant of a control system that sends its current state periodically to the server, which is the remote cloud controller. We implemented QUIC and HTTP3 using the `quic-go` [17] library, which provides an implementation of the IETF-standardized QUIC protocol.

The client runs on an Ubuntu-jammy machine with a 5.19.0-38-generic Linux kernel, while the servers are containerized using Docker and deployed on either an edge cluster or a public cloud. We evaluate the performance of the protocols under three different network scenarios, depending on where the servers are deployed: (i) Ethernet (wired local area network), (ii) Private 5G mobile network, (iii) Internet.

In order to evaluate how different protocols perform in various cloud environments and under different network scenarios, we intentionally did not control or tune the networks or network configurations specifically for the experiments.

The edge cloud environment used in the experiments consists of a seven-node bare-metal cluster with Kubernetes used for container orchestration. Each server is running in a Kubernetes Pod with BestEffort QoS class, and each node of the cluster is running on Ubuntu-jammy operating system with Linux kernel version 5.15.0-69-generic. For the evaluation under the edge scenario, two types of networks were used. The first is a wired network where the client and the edge cluster are only connected via Ethernet. The second is a private 5G mobile network hosted in the lab where we set up all the experiments, and the mobile network configuration and the base station setup are detailed in [16]. The client machine is connected to a 5G WNC modem to enable mobile communication, and the core functions of the 5G network are also deployed as Kubernetes services in the edge cluster, making the servers mobile edge applications under this scenario.

For the public cloud environment, we deployed each server as an Elastic Container Service (ECS) provided by AWS. The host AWS data center is located in Stockholm, Sweden, and each service is allocated 1 vCPU and 3GB memory with the compute engine provided by AWS Fargate. Under this scenario, the data path between the client and the servers traverses the Internet, which is much more unpredictable than the edge scenarios. Table I summarizes the three scenarios used in our evaluation. Additionally, to provide an intuitive view of the networks involved in our measurements, we also included the Round Trip Time (RTT) value via PING measurements to the servers through the underlying networks.

TABLE I
THREE EVALUATION SCENARIOS IN THE EXPERIMENT

Scenario Name	Ethernet	5G	AWS
Deployment Env.	Edge cluster	Edge cluster	AWS Public Cloud
Network	Ethernet LAN	5G	Internet
RTT	0.520ms	10.688ms	13.553ms

For the evaluation, we mimic the behavior of a CCS by designing a communication pattern between each pair of client and server in our experiment, since we don't focus on the performance related to control processes. Specifically, the client sends a request to the server every $d$ms, which corresponds to the sampling rate of a control system. Each request contains a payload of p bytes. Upon receiving each request, the server sends back a response that also contains p bytes payload. In this way, the communication pattern mimics a control system with a sampling time of $d$ms, where each message exchange between the plant and controller involves a payload of p bytes. Following are the values of d and p we have considered in our evaluation, aiming to imitate different types of control systems. The evaluation experiment was running for one hour under each pair of the parameters for collecting sufficient amount of data for analysis.

- d (ms): 5, 10, 25, 50, 75, 100
- p (bytes): 128, 256, 512, 1024

B. Performance Metrics

Considering the QoS requirements of a CCS, we preformed the evaluation on different protocols from two different perspectives: 1) *the response delay and jitter of the delay*, and 2) *data volume*.

The response delay in a CCS is the application response time measured at the plant, reflecting how long it takes to receive a server response after a request is made. This delay impacts the plant's ability to timely receive and act upon control signals, as responses are expected before the plant sends subsequent state updates. The response delay, typically longer than network latency, accounts for transport and application protocols processing time and the controller application's execution time in the cloud [18].

Based on [19], we calculate and express response delay jitter as a percentage by dividing the mean jitter value by the mean response delay. Jitter is vital to a CCS, particularly when network latency matches or exceeds the control process' sampling rate, necessitating delay compensation like plant state predictions. High jitter signifies unpredictable network and plant behavior, complicating delay compensation, while lower jitter suggests predictability, simplifying compensation.

We evaluate the data volume produced by different protocols in a CCS, crucial when the controller is cloud-based and service charges depend on traffic. Instead of throughput, which relates to network bandwidth and data rate determined by control plant frequency and payload size, we use the average packet size per request and response (Equation (1)) to represent overall data volume. This measure clearly shows each protocol's network overhead, assisting in selecting a protocol that minimizes data volume without affecting performances.

$$\begin{aligned}\text{data volume per request} &= \frac{\text{Total sent data by the plant}}{\text{Number of requests}}\\ \text{data volume per response} &= \frac{\text{Total received by the plant}}{\text{Number of responses}}\end{aligned} \tag{1}$$

IV. Evaluation Results

In this section, we present the results from our evaluation and our analysis on the results.

A. Response Time and Jitters

Based on the chosen values for the parameters d and network latency given in various scenarios in Table I, we have two different cases for evaluating the response delay and its jitter: 1) the *low-frequency case* and 2) the *high-frequency case*. In the low-frequency case, d is greater than the network latency, such as when $d = 25$ms under AWS and 5G scenarios. In this case, although the average response delay is longer than the network latency, it is usually shorter than one sampling interval d. The response delay and jitter performance shows the reliability of the network in the system. In the high-frequency case, when the sampling interval d is short than the network latency, delay compensation will be needed in the control system. However, a relatively lower response delay and smaller jitter has more advantages in designing the compensation algorithms.

1) Low-frequency case: Figure 1 displays the box plots of response delays for various protocols in three scenarios. From the bottom to the top, the box lines and whiskers indicate the 5th, 25th, 50th, 75th, and 95th percentile of the measurement data. The evaluation results are based on increasing payload size and a fixed sampling rate of $d = 25$ms, results in the cases with sampling rate $d \geq 25$ms are similar to this case.

Under all scenarios , single stream QUIC has around 0.3ms higher delays than TCP, since we don't have cryptographic implementation in our deployed TCP communication, which requires less handshake latency and processing time compared to QUIC. Meanwhile, the response delays generally increase with increasing payload size due to the higher processing time on larger size packets. In the Ethernet scenario, which provides the most reliable and lossless network among all our evaluated scenarios, HTTP3 exhibits a worse performance than HTTP1.1 in terms of response delay, which is consistent with the observations from previous research [10], [11]. Although 5G gives higher network latency, is considered as a reliable mobile network with little loss, and HTTP3 has higher response delay than HTTP1.1 under the 5G scenario, which shows the same behavior as the Ethernet scenario. Furthermore, we have observed that the payload size has larger impact on QUIC and HTTP3 in this case, especially on the median value of response delay.

The performance of the protocols under the AWS scenario exhibits differences from the other scenarios, where HTTP1.1

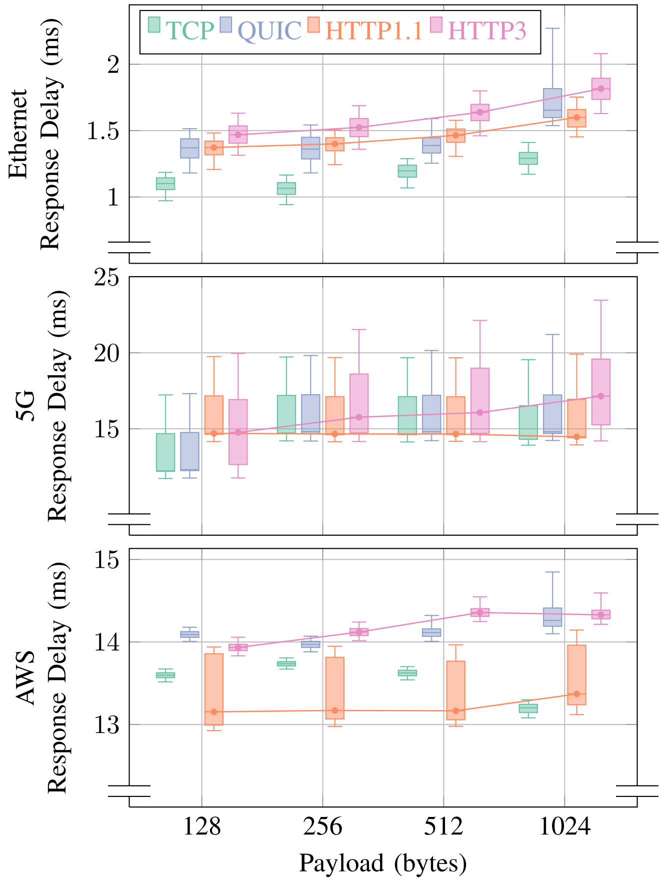

Fig. 1. Box-plots of response delay of different protocols under Ethernet, 5G and AWS scenarios, with increasing payload size p and sampling rate $d =$ 25ms. Due to different scales of observed network latency in each scenario, the plots are presented in different scales, and the y-axes are broken from origin start.

shows almost 1ms lower median values on response delays compared to TCP, and it also has much wider ranges between the first and third quartile compared to other protocols. This behavior may be attributed to the possibility of cloud providers restricting or deprioritizing certain traffic such as pure TCP and QUIC, which have no application protocol implemented, and this leads to an impact on the performance of the protocols. Other observation in the experiments under the AWS scenario is that the HTTP1.1 packets with TLS/TCP can not be decrypted, but the HTTP3 and QUIC can be. This may be due to certain security rules implemented in the data center, which also affect the delays introduced by HTTP1.1 under the AWS scenario.

Additionally, the response delays show smaller variances under the AWS scenario compared to the other scenarios, which is also revealed by the jitter performance illustrated in Figure 2. Under this scenario, since the ECSs hosting different server applications are not under the control of the user, the performance of each protocol may be influenced by the cloud execution environment and virtual network environment,

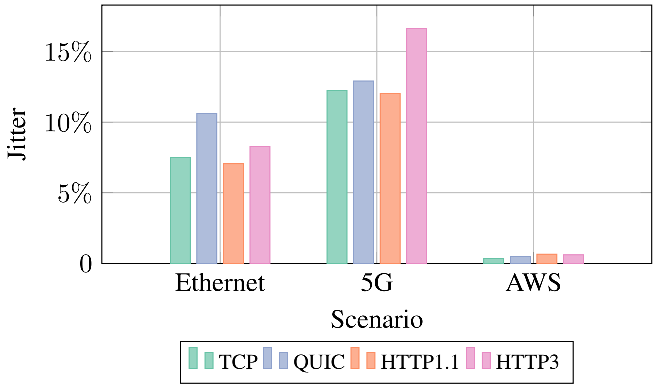

Fig. 2. The jitter of different protocols under three scenarios, with payload size $p = 256$bytes and sampling rate $d = 25$ms.

which also remain unknown to the user. Under the Ethernet and 5G scenarios, TCP shows less jitter than QUIC, and HTTP1.1 has less jitter than HTTP3.

In this case, both HTTP1.1 and HTTP3 require only one connection at the transport layer (TCP or UDP). With HTTP1.1, the TCP connection is persistent and reused by each request, meaning that only one connection establishment and transport/cryptographic handshake are needed. After the connection is established, the client can simply send requests and receive responses from the server. The acknowledgment for each request and response is usually contained in the response and the next request.

In the case of HTTP3, only one UDP connection is established, but each request-response pair is carried by a separate bidirectional stream that is initiated by the client. In order to allow new streams to open for new requests, the server must keep sending MAX_STREAMS QUIC frames to permit the streams to be opened cumulatively. Furthermore, as the next request establishes a new stream, the acknowledgment of the previous response cannot use the new stream and must be sent separately. This behavior adds more overhead in HTTP3 compared to HTTP1.1 in the low-frequency case and results in longer response delays and larger jitter.

2) high-frequency cases: In scenarios where the network latency exceeds the plant sampling interval d, as in the high-frequency case, multiplexing functionality is required for the protocols to function properly. Pure TCP or signal stream QUIC cannot achieve this, so the comparison in this case is limited to HTTP3 and HTTP1.1. The response delay and jitter performance with different payload sizes are shown in Figure 3 for a plant sampling interval of $d = 5$ms and an average network latency of 10.688ms under the 5G scenario. We choose this scenario as show case because the 5G network gives a relatively larger network latency than Ethernet, which may also longer than the sampling rate of certain control systems. The computational environment is also more controllable by us than the AWS scenario. The results indicate that, in this case, HTTP3 outperforms HTTP1.1 with respect to both response delay and jitter, which is in contrast to the low-frequency case.

When the network latency exceeds the plant sampling

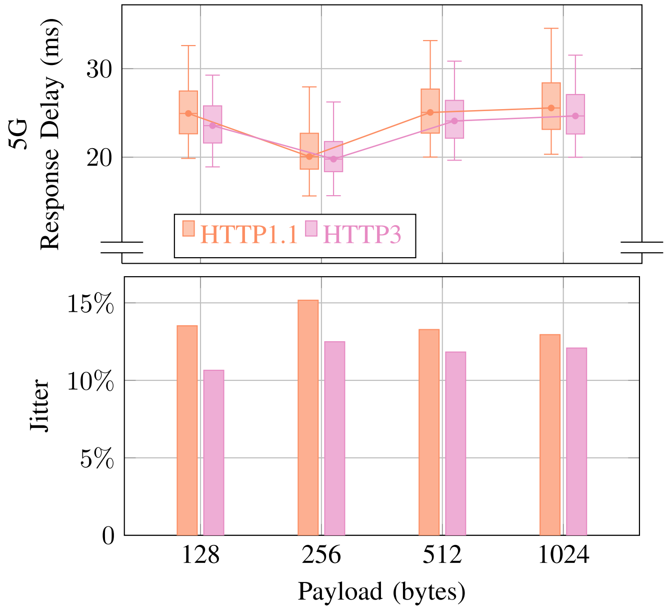

Fig. 3. Comparison of response delay (top) and jitter (bottom) performance between HTTP1.1 and HTTP3 under the 5G scenario with increasing payload size and a sampling rate of $d = 5$ms. The y-axis of the response delay performance is broken from origin start.

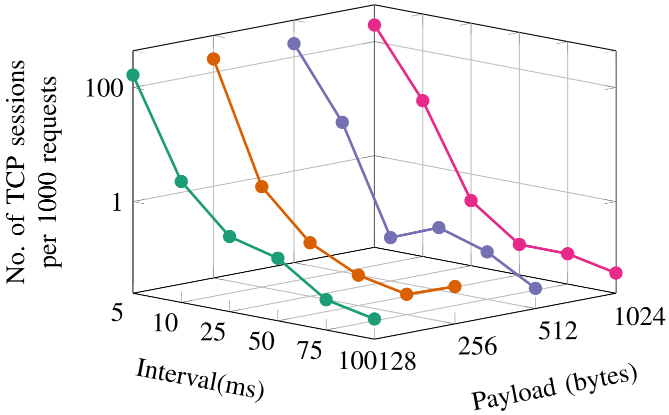

Fig. 4. Average number of TCP sessions established in HTTP1.1 experiments for 1000 requests under different plant sampling intervals and payload sizes. The z-axis value indicates the number of sessions per 1000 requests. In the higher frequency case with a sampling interval of $d = 5$ms, approximately 100-200 sessions were established for every 1000 requests. This number decreased to less than 15 sessions per 1000 requests when the sampling interval was $d = 10$ms. When d was greater than 50ms, typically only 1 or 2 sessions were used during each experiment.

interval d, the persistent TCP connection used by HTTP1.1 can become a bottleneck due to head-of-line blocking. This occurs when a slow response blocks the entire connection, leading to longer response delays and higher jitter. However, the control system must send the plant state periodically, regardless of whether the response to the previous request has arrived or not. An application using HTTP1.1 can handle this head-of-line blocking by using a new TCP connection for the next request before the response to the previous one has arrived. If a session is not available between the client and the server, a new session is established with new transport and cryptographic handshake. If a session is established for a new request, but is not reused by a subsequent request in a short time, it will be closed. The average number of TCP sessions established per 1000 requests in a series of experiments under the 5G scenario is shown in Figure 4, with the number of sessions increasing with higher frequencies. The established session number varies for different experiments even under the same parameter values, due to the highly dependent network latency and jitter. Even when the sampling rate d is larger than the network latency, more than one TCP session may be established due to jitter.

In contrast, HTTP3 allows multiple streams to be established simultaneously for multiple requests without extra handshake effort, and it has the same behavior as the low-frequency case. Therefore, in this case, HTTP3 outperforms HTTP1.1 in terms of response delay and jitter.

B. Data Volume

In Figure 5, we present the data volume evaluation results of a set of experiments under the 5G scenario, where the payload size of each request and response is 128 bytes. We chose this scenario for analysis because it gives a relatively larger network latency, meanwhile the edge cloud execution environment is more controllable. The figure shows that HTTP1.1 has a larger data volume per request than HTTP3 in all cases. This is primarily because TCP has a larger header size than UDP and QUIC, resulting in HTTP1.1 having a larger packet size than HTTP3 with the same payload amount. Comparing the data volume per request to the actual HTTP packet sizes in the figure, we can observe that the data volume per request with HTTP1.1 is much closer to its actual packet size compared to HTTP3. This implies that the overhead of HTTP1.1 is mainly due to the large header size in each protocol, while the overhead in HTTP3 is mainly introduced by the stream control frames and ACK packets as discussed earlier.

It is worth noting that in the high-frequency case with a sampling rate of $d = 5$ms, the data volume of HTTP1.1 is much higher than in other cases. This is because of the large number of TCP session establishments in this case, which is also evident in Figure 4, and the handshake packets causing a lot of network overhead. In general, considering the cost of deploying the controller in public cloud, HTTP3 is a more economical choice than HTTP1.1.

V. Conclusions

This paper presents our evaluation of transport protocols for CCSs, with the aim of exploring the potential advantages of deploying QUIC/HTTP3 over legacy TCP-based protocols. We conducted experiments under three different network scenarios and evaluated protocol performance based on two key factors: system response delay and jitter, and data volume created by the system. Our results show that when a control system operates at a relatively low frequency and network latency is not significant, QUIC and HTTP3 exhibit worse performance

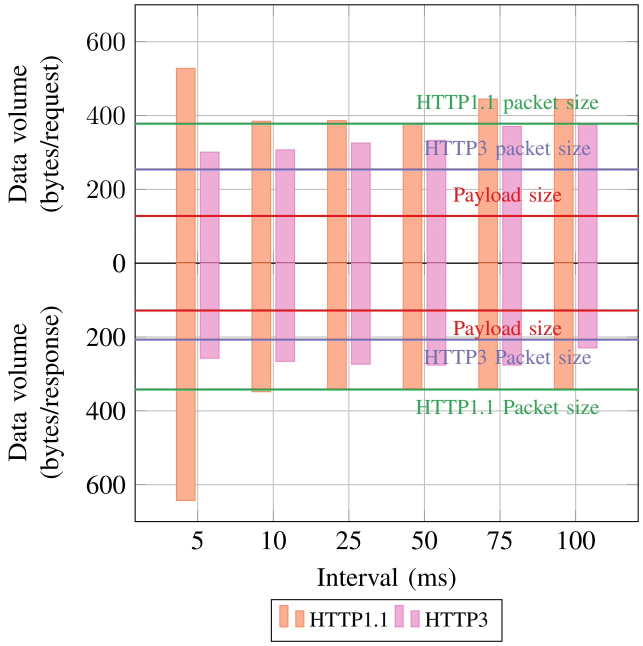

Fig. 5. Data volume per request (top) and per response (bottom) under the 5G scenario with different sampling intervals and payload size of 128 bytes. The bar charts depict the data volume generated by different protocols, where the red lines represent the payload size of 128 bytes. Additionally, the orange and blue lines denote the packet size of a single HTTP1.1 and HTTP3 request and response.

than TCP and HTTP1.1, consistent with previous research. However, under higher frequency with significant network latency, HTTP3 demonstrates shorter and less variant response delay, which is beneficial for more time-critical applications and advanced control designs. In terms of data volume, HTTP3 generates less data per request and less network overhead, indicating that it is a more cost-effective choice for deploying controllers in public clouds. Going forward, we plan to deploy actual time-critical control processes and examine the effectiveness of delay compensation with different protocols.

VI. Acknowledgment

This work was partially supported by the Wallenberg AI, Autonomous Systems and Software Program (WASP) funded by the Knut and Alice Wallenberg Foundation, the SEC4FACTORY project, funded by the Swedish Foundation for Strategic Research (SSF), and the IMMINENCE project funded by Sweden's Innovation Agency (VINNOVA). The authors are part of the Excellence Center at Linköping-Lund on Information Technology (ELLIIT), and the Nordic University Hub on Industrial IoT (HI2OT) funded by NordForsk.

References

[1] J. Iyengar and M. Thomson, "QUIC: A UDP-based multiplexed and secure transport," RFC 9000, May 2021, last Accessed: July 4, 2023. [Online]. Available: https://www.rfc-editor.org/info/rfc9000

[2] A. Langley, A. Riddoch, A. Wilk, A. Vicente, C. Krasic, D. Zhang, F. Yang, F. Kouranov, I. Swett, J. Iyengar, J. Bailey, J. Dorfman, J. Roskind, J. Kulik, P. Westin, R. Tenneti, R. Shade, R. Hamilton, V. Vasiliev, W.-T. Chang, and Z. Shi, "The QUIC transport protocol," in *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*. ACM, 2017.

[3] M. Bishop, "HTTP/3," RFC 9114, Jun. 2022, last Accessed: July 4, 2023. [Online]. Available: https://www.rfc-editor.org/info/rfc9114

[4] "Google Edge Network," last Accessed: July 4, 2023. [Online]. Available: https://peering.google.com/#/learn-more/quic

[5] M. Joras and Y. Chi, "How Facebook is bringing QUIC to billions," last Accessed: July 4, 2023. [Online]. Available: https://engineering.fb.com/2020/10/21/networking-traffic/how-facebook-is-bringing-quic-to-billions/

[6] P. Skarin, "Control over the cloud: Offloading, elastic computing, and predictive control," Ph.D. dissertation, Department of Automatic Control, Lund University, Nov. 2021.

[7] B. V. D. Cunha, X. Li, W. Wilson, and K. Harfoush, "Performance benchmarking of the QUIC transport protocol," in *2023 IEEE 20th Consumer Communications & Networking Conference (CCNC)*. IEEE.

[8] S. Cook, B. Mathieu, P. Truong, and I. Hamchaoui, "QUIC: Better for what and for whom?" in *2017 IEEE International Conference on Communications (ICC)*. IEEE, 2017.

[9] P. Megyesi, Z. Kramer, and S. Molnar, "How quick is quic?" in *2016 IEEE International Conference on Communications (ICC)*. IEEE, 2016.

[10] M. Seufert, R. Schatz, N. Wehner, and P. Casas, "QUICker or not? -an empirical analysis of QUIC vs TCP for video streaming QoE provisioning," in *2019 22nd Conference on Innovation in Clouds, Internet and Networks and Workshops (ICIN)*. IEEE, 2019.

[11] D. Saif, C.-H. Lung, and A. Matrawy, "An early benchmark of quality of experience between HTTP/2 and HTTP/3 using lighthouse," *IEEE International Conference on Communications*, 2020.

[12] M. Moulay, F. D. Munoz, and V. Mancuso, "On the experimental assessment of QUIC and congestion control schemes in cellular networks," in *19th Mediterranean Communication and Computer Networking Conference, MedComNet 2021*. IEEE, 2021.

[13] Nokia, "5G at the Sandvik underground test mine: A private wireless network enabling mining automation," last Accessed: July 4, 2023. [Online]. Available: https://go.rocktechnology.sandvik/l/490131/2021-09-10/99yx1z

[14] B. K. Christer Boberg, Malgorzata Svensson, "Distributed cloud – a key enabler of automotive and industry 4.0 use cases," last Accessed: July 4, 2023. [Online]. Available: https://www.ericsson.com/en/reports-and-papers/ericsson-technology-review/articles/distributed-cloud

[15] F. Akbarian, W. Tärneberg, E. Fitzgerald, and M. Kihl, "Attack resilient cloud-based control systems for industry 4.0," *IEEE Access*, vol. 11, 2023.

[16] H. Peng, W. Tarneberg, E. Fitzgerald, F. Tufvesson, and M. Kihl, "Evaluation of control over the edge of a configurable mid-band 5g base station," in *6th IEEE International Conference on Fog and Edge Computing, ICFEC 2022*. IEEE, 2022.

[17] "quic-go: A quic implementation in pure go," last Accessed: July 4, 2023. [Online]. Available: https://github.com/quic-go/quic-go

[18] H. Peng, W. Tarneberg, E. Fitzgerald, and M. Kihl, "Punctual cloud: Unbinding real-time applications from cloud-induced delays," in *2021 International Symposium on Networks, Computers and Communications*. IEEE.

[19] S. Poretsky, S. Erramilli, J. Perser, and S. Khurana, "Terminology for Benchmarking Network-layer Traffic Control Mechanisms," RFC 4689, Oct. 2006, last Accessed: July 4, 2023. [Online]. Available: https://www.rfc-editor.org/info/rfc4689