How Quick is Quic

Preetham Katta **Sai Kaushal Sadhu** **Venkatesh Venugopal**

GitHub Repository: ***https://github.com/reddy-preetham/fcn-project.git***

Abstract

The HTTP protocol is used to transfer the majority of internet traffic. Because the internet is constantly changing, the amount of data sent across the network is also growing. This necessitates the development of a better protocol capable of transferring large amounts of data at a reasonable speed. QUIC is a Google-developed protocol that aims to solve some of the current protocol's issues while simultaneously improving performance, primarily by lowering its time for a web page to load. The project's goal is to analyze the performance of QUIC against HTTP, SPDY and find out the difference in performances. The performance is measured based on the page load time across different network scenarios. This is important because the objects in the websites have become more complex, thus slowing the page load times. QUIC helps reduce the page load times in case of a large number of complex/large web objects in some scenarios. With the recent standardization of QUIC, we want to explore the impact of QUIC on different web applications and network security.

Introduction & Background

HTTP is an application layer protocol used for transferring web pages over the internet. HTTP protocol uses the TCP as its core in the transport layer. Since TCP is being used, the order is maintained while sending the packets. HTTP protocol requires a three-way handshake before exchanging data between the server and client. Almost all web pages nowadays use the HTTP/2 protocol to transfer web pages over the internet. One of the main problems present in the HTTP2 is the Head of Line Blocking, which hampers performance by a considerable factor.

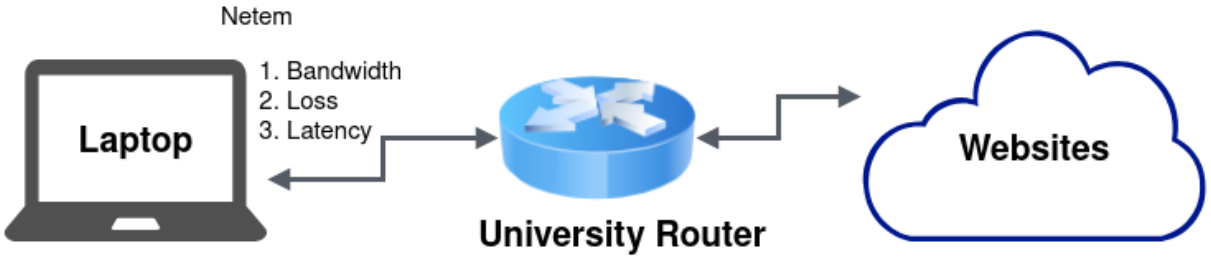
The key difference between QUIC and other protocols is that it minimizes web traffic latency by experimenting with UDP for downloading web applications instead of the usual TCP method. Unlike TCP, QUIC incorporates stream multiplexing directly into the design of its transport protocol. This feature relaxes the ordering constraints on packet delivery because

data only needs to be delivered in order at the stream level rather than at the connection level, thus avoiding HOL blocking problems present in TCP. QUIC also has a new encryption technique that replaces SSL/TLS and reduces round trips between the client and server during connection setup. QUIC uses TCP Cubic as the congestion control algorithm. In addition, it uses a feature called packet pacing where QUIC monitors the inter-packet spacing and uses this to estimate available bandwidth and control packet pacing. Although this is done to reduce congestion related packet loss, it can be seen that this hinders performance significantly.

When compared to previous versions, QUIC introduces numerous changes. The project's primary goal is to compare and evaluate the scenarios in which QUIC performs better and determine where it fails. Finally, we assess whether it is worthwhile to switch from the current protocol to QUIC.

While evaluating performance metrics, we have considered time to the first byte along with page load time. We have also considered object MimeType in our analysis to check the load time according to the objects present on the web page. So, we believe these are the additional analyses from our part compared to the original paper.

Test Setup



Client Setup:

All of our tests were carried out on a Mac(M1, 16GB), Ubuntu(NVidia RTX2070, 32GB), Ubuntu(NVidia GTX1660, 8GB), and University wifi. The experimental data from all of these devices were combined and analyzed. In order to capture data while doing our testing for both QUIC (HTTP 3) and HTTP2, we used the chromium headless version. The HAR (HTTP archive format) files from the chromium-browser were generated and used for our investigation.

Server Setup:

Over 20 websites that supported QUIC were used in the experiment. Because only Google, Facebook, and Cloudflare currently support QUIC, the websites chosen all belong to these domains. To the best of our knowledge, these were the only companies that provided production services to QUIC. We ran tests on two types of workloads: single-object resources, which allowed us to look at raw protocol behavior, and multi-object web pages, which allowed us to look at protocol behavior while the page was loading. For both types of workloads, we used static resources.

Network Emulation:

We have emulated different network environments using the tc command, and the following are the list of bandwidths, latencies, and loss percentages used:

Latencies = [1ms, 100ms]

Loss Rates = [0%,2%]

Bandwidths =[2Mbps, 10Mbps, 50Mbps]

Therefore, there would be 12 different scenarios for each website to be run. Since we have a total of 20 websites, we have 240 different test cases each for TCP and QUIC. We have generated har files using a python script and a chromium browser. For each possible scenario, we try to analyze how QUIC compares with the HTTP2 protocol.

Evaluation

We change the following parameters based on the setup mentioned above and perform our analysis.

Parameter Type	Parameter	Values tested
Network	Bandwidth	[2Mbps, 10Mbps, 50Mbps]
	Loss rate	[0%, 2%]
	Latency	[1ms, 100ms]
Webpage	Webpage Size	[0.5 MB, 1.5 MB, 2.5 MB] *{small, medium, large}*
	Object Type	CSS, HTML, IMAGE, JS

Performance Metrics:

1. Page Load Time: Indicates the time taken for the specific webpage to load along with all of the resources present in the webpage. This is the cardinal metric in estimating the performance of the underlying protocol used.

2. Time to First Byte: This metric is used to determine how fast the connection is established and how soon the communication can start between the server and the client.

Measurement Tools:

1. Chromium Browser: Headless version of the chromium browser is spawned from the python code, this acts as the client for the QUIC and HTTP2 protocols

2. HAR Capturer: This is used to capture all the logs from the chromium browser when it interacts with sites. Using these logs, we perform our analysis.

Results:

_____We use H2 for HTTP2 and H3 for QUIC interchangeable across the results.

1. Comparing Page Load time for varying bandwidth and loss

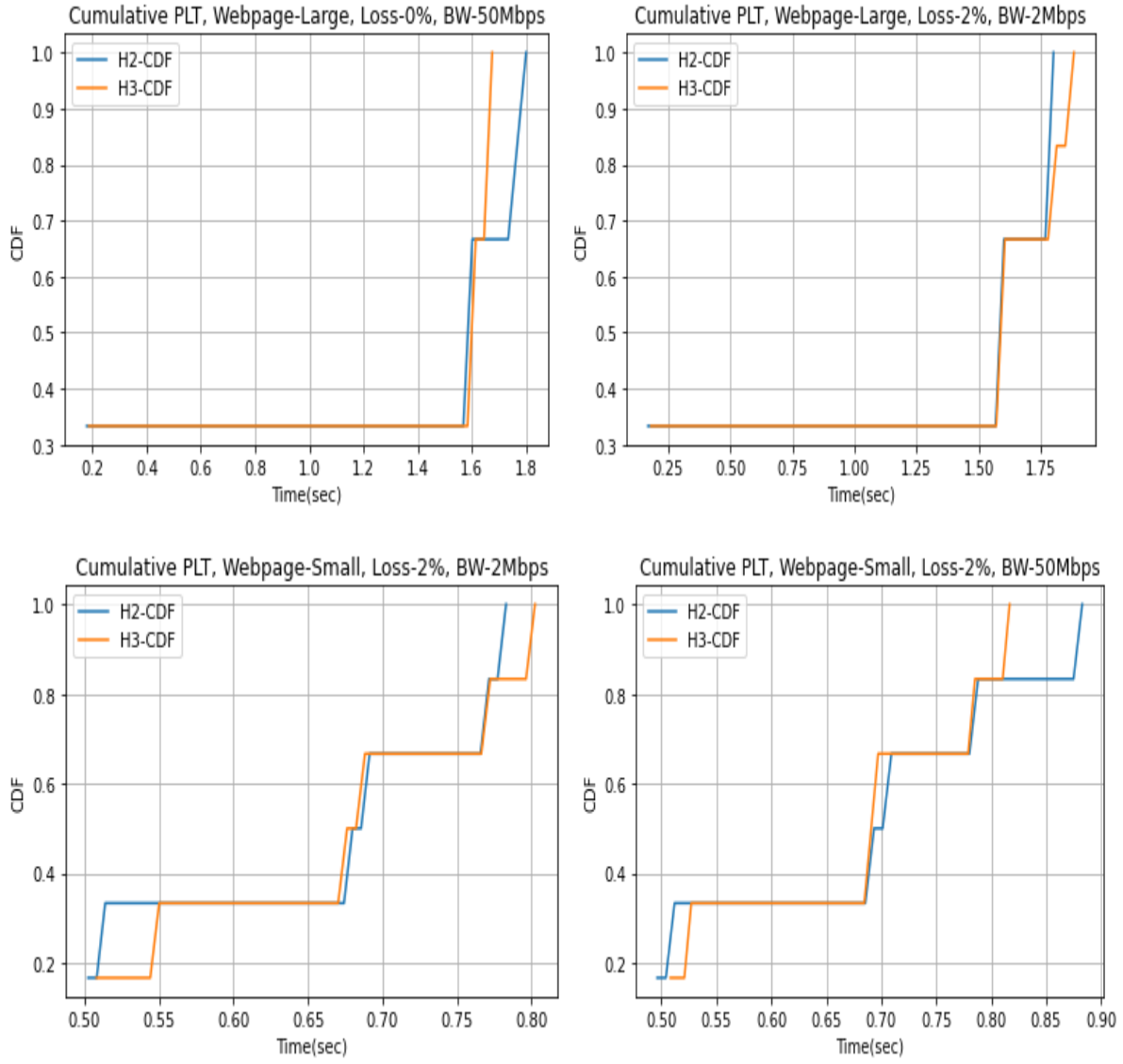


Figure 1

Figure 1 shows the web page load time of QUIC and HTTP2 for small and large web pages under different network conditions. It is expected that QUIC provides better performance under a high loss rate and when the Bandwidth is low. However, our experiments reveal that both QUIC and HTTP2 provide comparable performance.

2. Comparing Object load time for loss = 0 and varying bandwidth

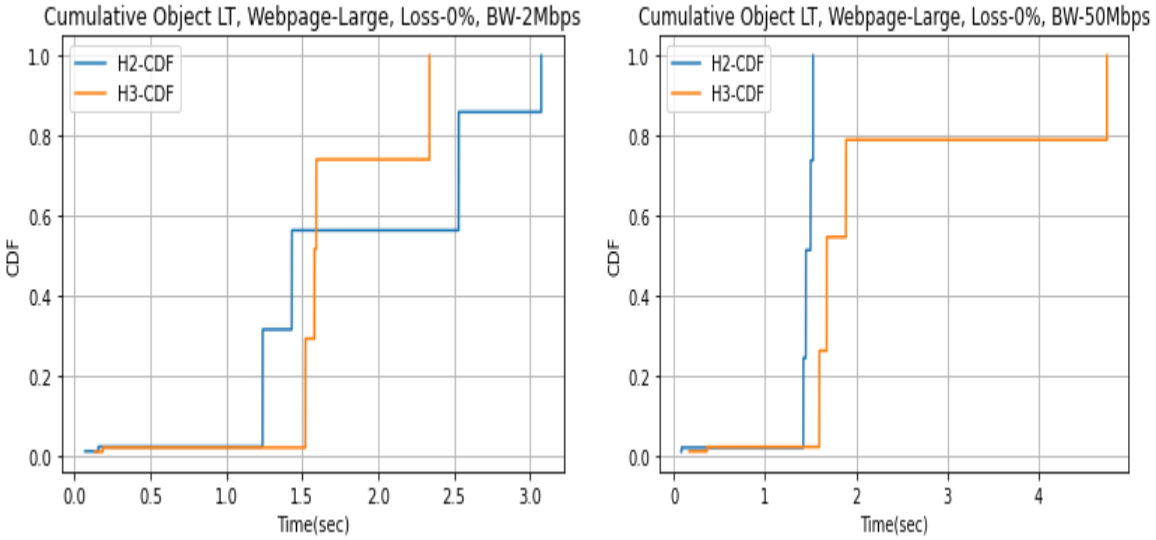
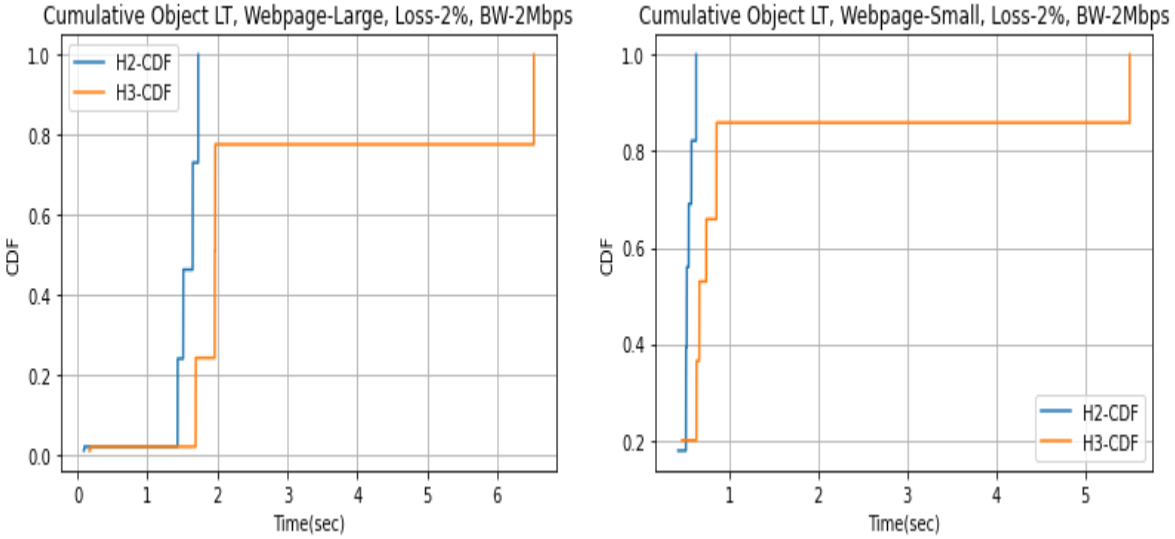


Figure 2

Figure 2 shows the Object load time of QUIC and HTTP2 when the loss rate is 0 and with the varying Bandwidth. It can be inferred that under lower bandwidth, QUIC provides better results and when the bandwidth increases, HTTP2 provides better results than QUIC.

3. Comparing Object load time for loss = 2% and varying bandwidth



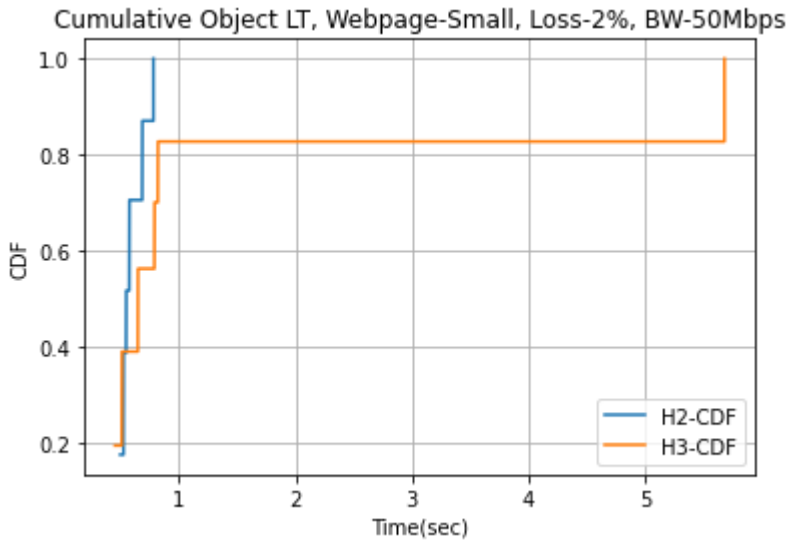


Figure 3

Figure 3 shows the object load times for when the loss is 2% and with the varying Bandwidth. We see that both QUIC and HTTP2 provide comparable results from the above experiments.

4. **Comparing Time to get the first byte for varying loss, bandwidth, and web page size**

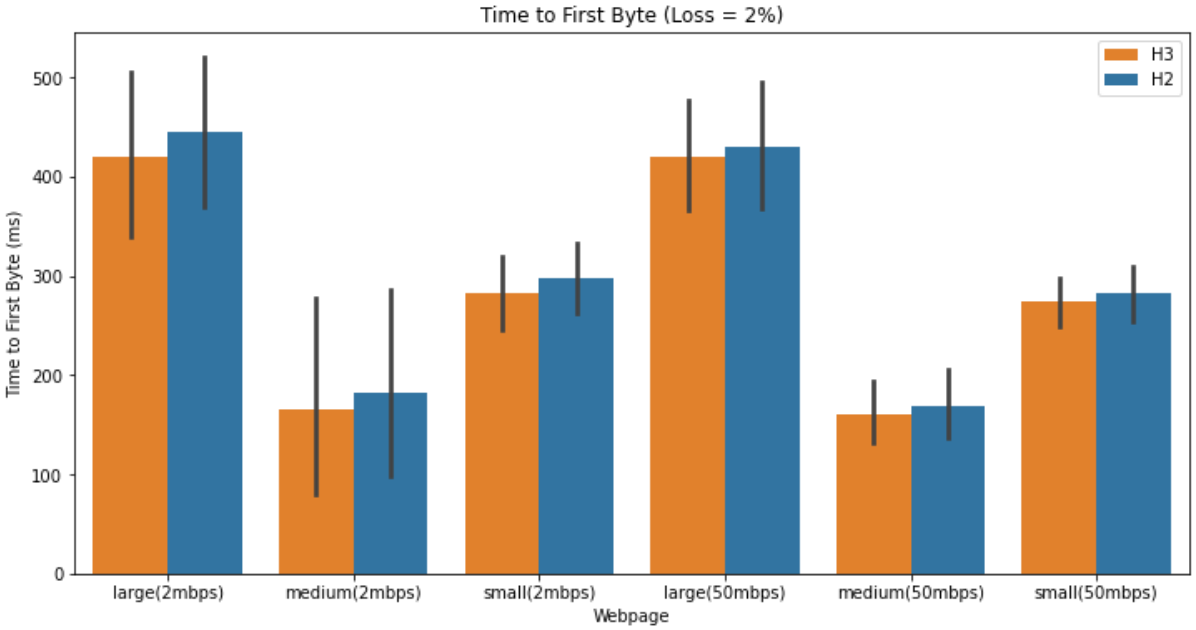


Figure 4

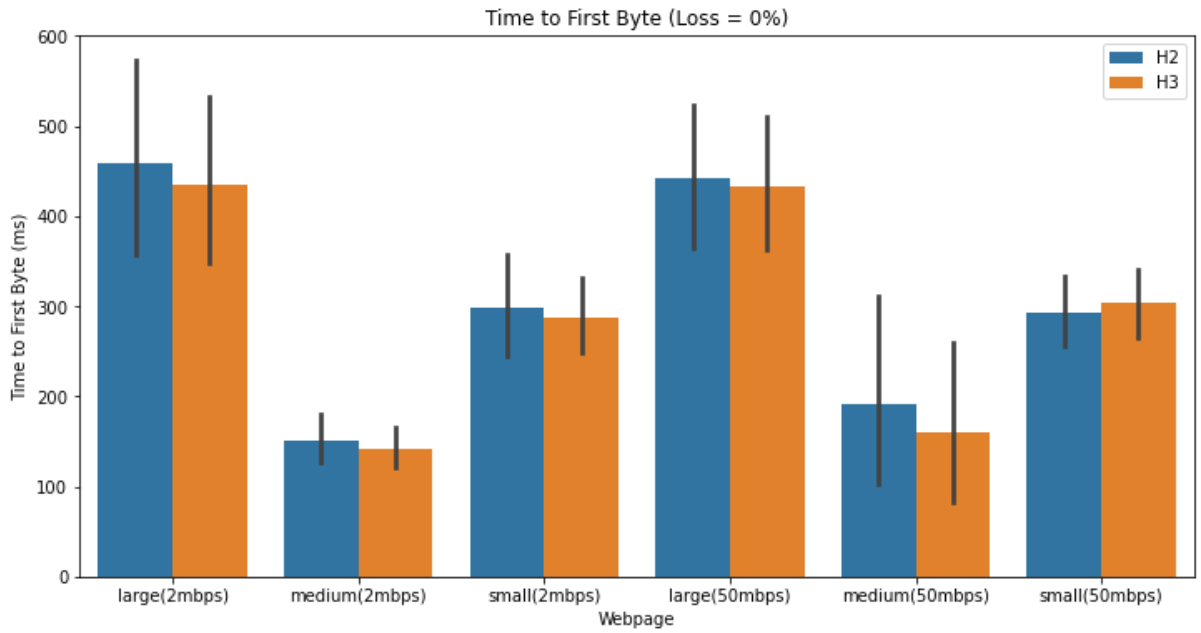
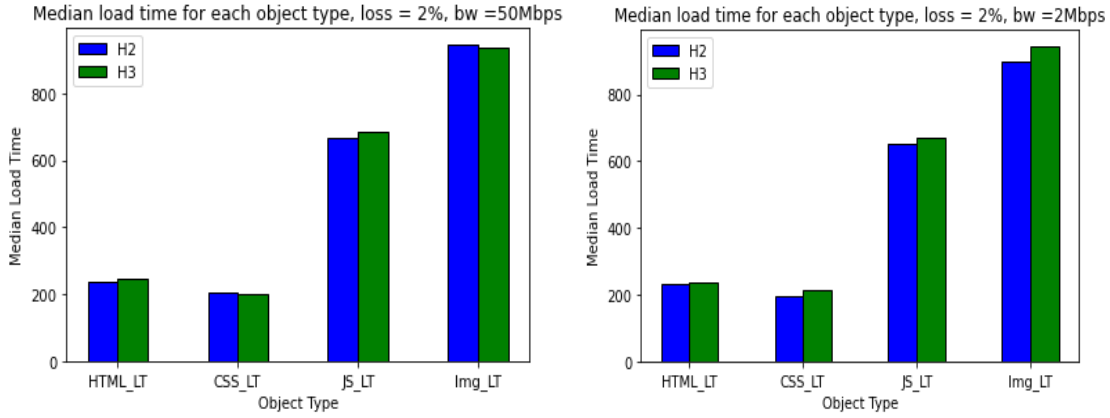


Figure 5

From figure 4 and 5, we see that there is a slight improvement for the connection time in the case of QUIC, but this is almost negligible which is not significant as claimed by QUIC protocol.

5. Comparing Load time of each object type w.r.t mentioned network scenarios



We also compared the fetch times of different object types under both QUIC and HTTP2 protocol. As there are more number of image and javascript objects than HTML and CSS objects they take more Load Time to fetch. It is seen that HTTP2, QUIC provide comparable results.

Conclusion:

From the above analysis, not all our hypotheses were proven to be true. Most of the hypotheses we assumed had HTTP3 to perform better for its lesser RTT in connection setup, but the analysis showed similar performances for H2 and H3, sometimes worse. The main problem for this result might be the inconsistencies in the number of websites using QUIC. Perhaps a year from now, performing the same analysis would give us better results considering the month of QUIC standardization, i.e., May 2021. QUIC appears to be a poor choice when a web page contains many considerable resources because QUIC multiplexes the requests for all of these resources, leading to a significant latency for one of those resources to finish. Having higher loss percentages than 2% might work better because the analysis and the performance showed negligible effect by introducing the mentioned loss.

References:

1.*How quick is QUIC*

2. *How speedy is SPDY*

3.*Dissecting Performance of Production QUIC*

4. *A First Look at QUIC in the Wild*

4.*Chrome HAR Capturer*

5. *Miscellaneous git repositories referenced for analysis:*

https://chromium.googlesource.com/chromium/src/+/master/third_party/WebKit/Source/core/inspector/browser_protocol.json%20

https://github.com/cyrus-and/chrome-har-capturer%20

https://github.com/triplewy/quic-benchmarks%20

https://github.com/Shenggan/quic_vs_tcp%20

https://github.com/arashmolavi/quic