Evaluation on HTTP/3 Protocol Under Network Congestion

Yujia Zhang
yuz232@ucsd.edu

Mingjie Song
m8song@ucsd.edu

1 Abstract

Since the beginnings of the internet, HTTP (Hyperlink Text Transfer Protocol), the one of the main methods of data transfer across the internet, has attracted significant interest. In the days since the early internet, web services have undergone significant changes, however, rendering the initial HTTP implementation inadequate for efficient data delivery under various congested internet situations. Google has taken the lead in seeking improved solutions for downloading web pages and, in 2013, introduced a new protocol called Quick UDP Internet Connections (QUIC). QUIC was experimentally developed in Google Chrome, and Google claimed to have seen about a three percent improvement in mean page load times with QUIC on Google Search. HTTP/3, as the latest version of HTTP, leverages the QUIC protocol as its underlying transport mechanism. Since HTTP/3 is a relatively new protocol, we sought to enhance our understanding of its functionality and performance across various network scenarios. We utilized CloudLab to set up a server with real-world web content containing different versions of HTTP. We then set up a client machine at the next hop to request data and simulate different congestion levels. We present an analysis of the performance of HTTP/3 and HTTP/1, with a particular focus on their performance under different congestion levels. Our experiments indicate that HTTP/3 performs better than HTTP/1 under network congestion, but the overall improvement is not significant for a small amount of non-streaming web contents. However, based on our analysis of QUIC, we anticipate that HTTP/3 would demonstrate even more significant benefits and reveal more meaningful data when tested with larger number of resource files with larger file size.

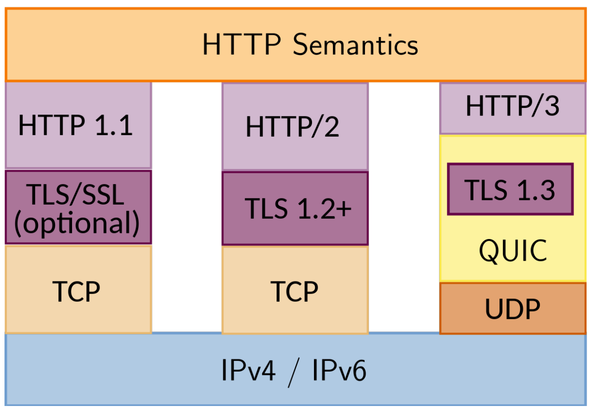
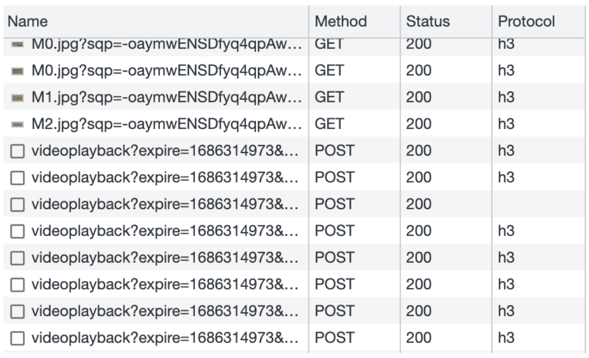


Figure 1: Evolution of HTTP protocol

2 Introduction

In 1989, the emergence of the World Wide Web (WWW), commonly known as the web, is considered as a milestone event in the history of the Internet. It greatly lowered the barrier for people to navigate information across the world with its user friendly interfaces and hyperlinks. HTTP is inseparable from this success. As an application level protocol, HTTP/1 uses TCP as its underlying protocol to ensure reliable data delivery and ordering of data. TCP incorporates a congestion window and a congestion policy to avoid congestion. This adaptively adjusts to sending rates based on network congestion levels. As wide area networks have expanded their capacity, web traffic has also increased. This shift encompasses more than just static HTML content, now encompassing high-resolution images, dynamic scripts, and streaming content. These types of traffic have different requirements compared to traditional static web pages. Additionally, the growing usage of HTTP over SSL (HTTPS) for encrypting web traffic means that servers now spend additional time establishing a secure channel over a TCP connection before transmitting data.



Name	Method	Status	Protocol
M0.jpg?sqp=-oaymwENSDfyq4qpAw...	GET	200	h3
M0.jpg?sqp=-oaymwENSDfyq4qpAw...	GET	200	h3
M1.jpg?sqp=-oaymwENSDfyq4qpAw...	GET	200	h3
M2.jpg?sqp=-oaymwENSDfyq4qpAw...	GET	200	h3
videoplayback?expire=1686314973&...	POST	200	h3
videoplayback?expire=1686314973&...	POST	200	h3
videoplayback?expire=1686314973&...	POST	200	
videoplayback?expire=1686314973&...	POST	200	h3
videoplayback?expire=1686314973&...	POST	200	h3
videoplayback?expire=1686314973&...	POST	200	h3
videoplayback?expire=1686314973&...	POST	200	h3
videoplayback?expire=1686314973&...	POST	200	h3

Figure 2: Google's Youtube Utilize HTTP/3 to deliver streaming content

TCP's congestion control mechanisms, which involve re-transmission of packets and reducing the sending rate after packet loss, aim to ensure reliability and prevent unfair allocation of network resources. However, these mechanisms can hinder the optimal performance of applications like web browsers in congested networks. In 2013, Google introduced Quick UDP Internet Connections (QUIC) as part of its Chrome browser. Unlike traditional HTTP that relies on TCP, QUIC/HTTP3 was developed with its own congestion control mechanism and utilizes UDP in the transport layer. In June 2015, an Internet Draft of the QUIC specification was submitted to the IETF, and in October 2018, the IETF's HTTP and QUIC Working Groups decided to call the HTTP mapping over QUIC "HTTP/3" in preparation for its worldwide standardization. These developments led to increased research interest in the performance of QUIC/HTTP/3.

Surprisingly, there is a lack of publicly available data on how HTTP/3 behaves in congested networks when compared to HTTP/1 and HTTP2. Although HTTP/3 was standardized in August 2020 by the Internet Engineering Task Force (IETF) as RFC 8446, most of the application web frameworks nowadays still utilize HTTP/1.1 or HTTP/2 as a default protocol, and have limited support for HTTP/3. Therefore, there is a need to gather data on HTTP/3's congestion response. Considering the increasing adoption of QUIC-based applications, exploring this topic can provide some insights to the direction of future HTTP evolution.

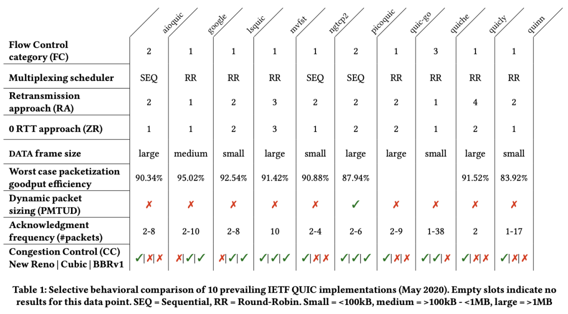
3 Related Work

When Google initially introduced QUIC in 2013, Adam Langley and his team conducted an evaluation of the original gQUIC protocol by implementing it in Google Chrome. They focused their evaluation on Google Search and YouTube mobile apps. The evaluation included metrics such as search latency, video playback latency, and video rebuffer rate. According to Langley and his team, QUIC demonstrated significant improvements, including a 3 percent increase in client desktop throughput, a 2 percent reduction in search latency, and a 9 percent decrease in video rebuffer rates.

Alexander Yu and Theophilus A. Benson conducted studies to evaluate the performance of QUIC in production compared to TCP, focusing on endpoints hosted by Google, Facebook, and Cloudflare. Their evaluation considered various dimensions, including network conditions, workloads, and client implementations. They used different types of workloads, including single-object and multi-object web pages, and tested QUIC clients such as Google Chrome, Facebook Proxygen, and Ngtcp2. Their findings highlighted the influence of the server's choice of congestion control algorithm and the importance of configuration tuning for optimal QUIC performance. They also emphasized that QUIC's removal of head-of-line blocking had minimal impact on web-page performance. Their observations underscored that QUIC's performance is closely tied to implementation design choices, bugs, and configurations, indicating that measurements of QUIC may not always reflect the protocol itself and may not generalize across different deployments.

Similarly, Peter Megyesi and his team conducted a comprehensive study comparing the performance of QUIC, SPDY, and HTTP, specifically focusing on page load time. They used a regular laptop with the Chrome browser and automated the download process using the Chrome HAR capturer. Their experiments showed that QUIC performed poorly under high bandwidth conditions when dealing with large amounts of data, but outperformed the other protocols under high round-trip time (RTT) values, particularly with low bandwidth. They concluded that none of the protocols consistently outperformed the others, highlighting the significant role of actual network conditions in determining the best-performing protocol.

In another study, Robin Marx and his team compared fifteen IETF QUIC+HTTP/3 implementations, evaluating advanced features such as Flow



	aioquic	google	lsquic	mvfst	ngtcp2	picoquic	quic-go	quiche	quicly	quinn
Flow Control category (FC)	2	1	1	1	1	2	1	3	1	1
Multiplexing scheduler	SEQ	RR	RR	RR	SEQ	SEQ	RR	RR	RR	RR
Retransmission approach (RA)	2	1	2	3	2	2	2	1	4	2
0 RTT approach (ZR)	1	1	2	3	1	2	2	1	2	1
DATA frame size	large	medium	small	large	small	large	large	small	large	small
Worst case packetization goodput efficiency	90.34%	95.02%	92.54%	91.42%	90.88%	87.94%			91.52%	83.92%
Dynamic packet sizing (PMTUD)	✗	✗	✗	✗	✗	✓	✗	✗	✗	✗
Acknowledgment frequency (#packets)	2-8	2-10	2-8	10	2-4	2-6	2-9	1-38	2	1-17
Congestion Control (CC) New Reno | Cubic | BBRv1	✓|✗|✗	✗|✓|✓	✗|✓|✓	✓|✓|✓	✓|✗|✗	✓|✓|✓	✓|✗|✗	✓|✓|✗	✓|✗|✗	✓|✗|✗

Table 1: Selective behavioral comparison of 10 prevailing IETF QUIC implementations (May 2020). Empty slots indicate no results for this data point. SEQ = Sequential, RR = Round-Robin. Small = <100kB, medium = >100kB - <1MB, large = >1MB

Figure 3: Different IETF QUIC implementations presented by Robin Marx et al.

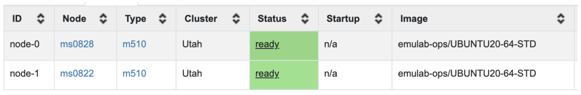
and Congestion Control, 0-RTT, Multiplexing, and Packetization. They used the structured qlog format for their evaluation. Their analysis showed that QUIC incorporates several TCP concepts and best practices for loss detection and congestion control. While many implementations adopted QUIC's New Reno variant, larger companies were exploring more modern congestion control algorithms. However, congestion control implementation was still a work in progress for most, lacking extensive validation. Some developers without sufficient resources expressed limitations in congestion control expertise and the ability to adopt advanced algorithms. The team emphasized the need for future work to demonstrate scientific rigor through root-cause analysis of observed behaviors and comparison of multiple QUIC implementations.

Overall, these studies highlight the importance of implementation design choices, configuration tuning, and specific network conditions in determining the performance of QUIC. They contribute to our understanding of QUIC's strengths and limitations and provide insights for further research in this field.

4 Experimental Setup

In this section we provide insights into the methodology we used to compare the performance of HTTP/3 and HTTP/1 under congestion, which includes the details of the experimental setup such as equipment, experimental design, and variables.

Our goal is to test and compare the network performance between HTTP/3 and HTTP/1 under different network congestion environments. We expect that HTTP/3 would take the advantage of the QUIC protocol and lead to a better performance (Throughput, RRT, etc) compared to the



ID	Node	Type	Cluster	Status	Startup	Image
node-0	ms0828	m510	Utah	ready	n/a	emulab-ops/UBUNTU20-64-STD
node-1	ms0822	m510	Utah	ready	n/a	emulab-ops/UBUNTU20-64-STD

Figure 4: Machines Information

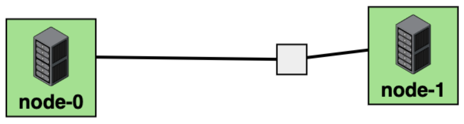


Figure 5: Network Topology

HTTP1 at high levels of congestion. To test this, we set up two machines on the cloudLab and connected them using an Ethernet cable. Both machines ran on the hardware type m510 and used UBUNTU20-64-STD disk image as shown in the Figure3, which allows us to use the Linux traffic control to mimic different network environments.

As shown in Figure 4, we setup two machines and used one machine(node1) as the server side and another one as the client side(node 0). On the server side, we pulled the resource files from https://ucsd.edu/ and https://www.nyu.edu/ websites using the Linux wget command and hosted them locally. We applied aioquic, a Python-based QUIC protocol implementation, in our server infrastructure to efficiently handle user requests using the HTTP3 protocol. Its GitHub repository received 1.3k stars and is regularly tested for interoperability against other QUIC implementations. The aioquic features a minimal TLS 1.3 implementation, a QUIC and an HTTP/3 stack that were standardized in RFC 9000 and RFC 9114 respectively. The local-hosted UCSD website contains 13 files and NYU website contains 43 files. We expect that the improvements of the QUIC protocol would be seen more prominently in websites with multiple numbers of objects. On the client side, we utilized the Chrome Headless tool to send requests to our server and Chrome DevTools Protocol library to collect timing information and analyze the network performance.

To control the server's egress traffic and mimic different congestion conditions, we utilized two types of TC commands. The first command was to apply the Token Bucket Filter queuing discipline as shown in the Figure 5 to control the data rate, where we collected the network per-

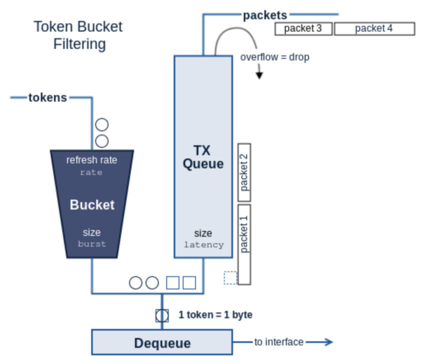
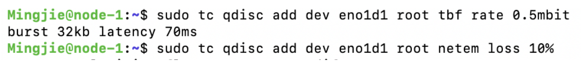


Figure 6: TBF Queuing Discipline



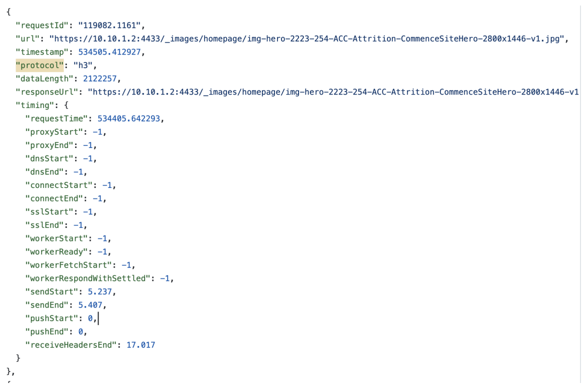
```
Mingjie@node-1:~$ sudo tc qdisc add dev eno1d1 root tbf rate 0.5mbit
burst 32kb latency 70ms
Mingjie@node-1:~$ sudo tc qdisc add dev eno1d1 root netem loss 10%
```

Figure 7: Example TC Commands Using TBF and Netem

formance information of 7 different data rates (50kbit, 100kbit, 0.5mbit, 1mbit, 1gbit, 5gbit, 10gbit) from both of the locally-hosted websites. A higher congestion level is simulated by reducing the data rates. The two mandatory parameters of TBF queuing discipline, bucket size and latency values, were determined using iperf3 to ensure that they were at the right size to support the data rate we defined. The second method was to use the network emulator called netem to introduce various network impairments and simulate different network conditions. It allows the emulation of delays, packet loss, reordering, and other network characteristics. In our experiment, we setup 5 packet loss rates (5, 10, 20, 50) as we expect both HTTP/1 and HTTP/3 would adjust their sending rates based on the number of packets lost. Figure 6 shows an example of two types of TC commands.

5 Data Analysis

The data was tracked and collected using the Chrome DevTools Protocol library. Each request's information was stored in a json file with the format shown in figure 7, where it contains the valuable fields such as url, protocol, dataLength and the timing information such as requestTime, sendStart and receiveHeadersEnd. We analyzed the network performance primarily using the throughput and the round trip time for each request. The throughput was calculated in bytes/s by the sum of all received data divided by the total time spent to receive the entire response and the



```
{
  "requestId": "119082.1161",
  "url": "https://10.10.1.2:4433/_images/homepage/img-hero-2223-254-ACC-Attrition-CommenceSiteHero-2800x1446-v1.jpg",
  "timestamp": 534505.412927,
  "protocol": "h3",
  "dataLength": 2122257,
  "responseUrl": "https://10.10.1.2:4433/_images/homepage/img-hero-2223-254-ACC-Attrition-CommenceSiteHero-2800x1446-v1
  "timing": {
    "requestTime": 534405.642293,
    "proxyStart": -1,
    "proxyEnd": -1,
    "dnsStart": -1,
    "dnsEnd": -1,
    "connectStart": -1,
    "connectEnd": -1,
    "sslStart": -1,
    "sslEnd": -1,
    "workerStart": -1,
    "workerReady": -1,
    "workerFetchStart": -1,
    "workerRespondWithSettled": -1,
    "sendStart": 5.237,
    "sendEnd": 5.407,
    "pushStart": 0,
    "pushEnd": 0,
    "receiveHeadersEnd": 17.017
  }
},
```

Figure 8: Collected Data Format

round trip time of each request was the value of $receiveHeadersEnd - sendStart$ in ms. Next we will compare and analyze how throughput and round trip time change based on the different protocol used and under different network environments.

The results depicted in Figure 9 exhibit the variation in throughput values for a single request to the locally hosted UCSD and NYU websites across different data rate limit ranges, spanning from 50 kbit to 10 Gbit, measured in bytes per second. Overall we can conclude that HTTP/3 shows a better performance when handling congestion compared to the HTTP/1 protocol. Notably, a network bottleneck became apparent when the rate limit ranges from 50 kbit to 0.5 Mbit, resulting from congestion arising due to the limited rate capacity. In response, the throughput value exhibited a rapid increase as the data rate increases. Upon surpassing the 0.5 Mbit rate limit, a trend emerged where the throughput experienced a relatively sluggish growth relative to the rate capacity. This phenomenon can likely be attributed to the ample rate capacity being adequately equipped to handle the existing volume of data transmission. We next examined the impact of packet losses on throughput values. As seen in figure 10, despite observing some fluctuations in throughput values associated with lower packet loss percentages, as it was quite challenging maintaining a consistent packet loss during the experiment, the data analysis demonstrated that in the majority of cases HTTP/3 matched or outperformed HTTP/1 in terms of throughput, even in the presence of varying packet loss percentages.

Figure 11 focuses on the round trip time of retrieving separate resource files from the locally hosted UCSD webpage under a data rate of 0.5

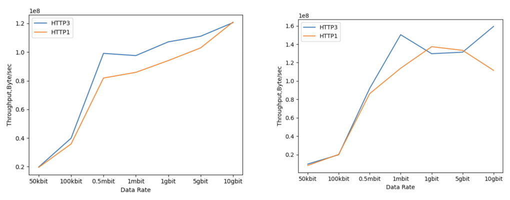


Figure 9: Throughput Under Different Data Rates When Requesting UCSD(left) And NYU(right) Webpage

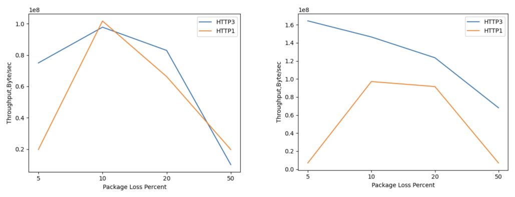


Figure 10: Throughput Under Different Packet Loss Percentages When Requesting UCSD(left) And NYU(right) Webpage

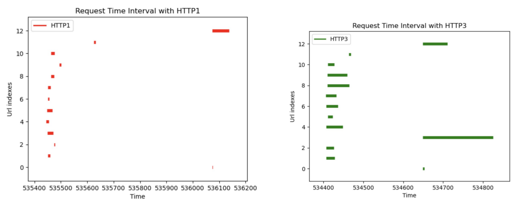


Figure 11: RTT for each resource file under HTTP/1(left) and HTTP/3(right)

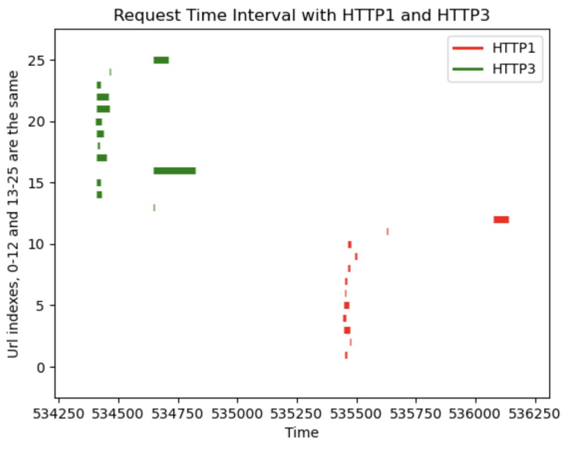


Figure 12: RTT Comparison between HTTP/3 and HTTP/1

megabits per second. Notably, in the case of HTTP/3, all the retrieval processes commenced nearly simultaneously, whereas in HTTP/1, the start times vary. This observation highlights the multiplexing capability of the QUIC protocol, which can efficiently handle multiple concurrent data streams within a single connection. In contrast, HTTP/1 establishes separate TCP connections for each data resource. Figure 12 consolidates the findings from Figure 11, presenting a more comprehensive view of the time dynamics. It became apparent that HTTP/3 exhibited a more condensed time frame in handling all the resources, while HTTP/1 demonstrated a more scattered distribution. The overall outcome underscores the advantageous performance of HTTP/3 in terms of resource retrieval efficiency compared to HTTP/1.

Due to the time and resource limitations in our testing, we were unable to explore the full potential of QUIC in terms of its congestion control and multiplexing abilities with different types and sizes of resource files. However, based on our understanding of QUIC's design, we anticipate that QUIC may demonstrate even more significant benefits and reveal more meaningful data when tested with larger number of resource files with larger file size.

6 Conclusion

Our experiment showed a minor overall performance improvement of HTTP/3 compared to HTTP/1 and revealed the advantages of applying QUIC as the underlaying transport layer protocol. QUIC's multiplexing ability can greatly reduce the redundant time needed for traditional TCP to build connections and also alleviate the head-of-line (HOL) blocking problem where a lost packet containing data from one stream prevents the receiver from processing subsequent packets containing data from other streams. By incorporating security and stream multiplexing directly into its design, we noticed a faster and smoother time span for QUIC to establish its network connection and a better overall performance under congestion environment.

References

Gaetano Carlucci, Luca De Cicco, and Saverio Mascolo. HTTP over UDP: an Experimental Investigation of QUIC. In Proceedings of the 30th Annual ACM Symposium on Applied Computing, pages 609–614. ACM, 2015.

P. Megyesi, Z. Krämer and S. Molnár, ”How quick is QUIC?,” 2016 IEEE International Conference on Communications (ICC), Kuala Lumpur, Malaysia, 2016, pp. 1-6, doi: 10.1109/ICC.2016.7510788.

QUIC, a multiplexed stream transport over UDP - The Chromium Projects. https://www.chromium.org/quic/, 2017. [Online; accessed 01-January-2017].

Marx, R., Herbots, J., Lamotte, W., Quax, P. (2020). Same Standards, Different Decisions: A Study of QUIC and HTTP/3 Implementation Diversity. In Proceedings of the Workshop on the Evolution, Performance, and Interoperability of QUIC (pp. 14–20). Association for Computing Machinery.

Adam Langley, Al Riddoch, Alyssa Wilk, Antonio Vicente, Charles 'Buck' Krasic, Cherie Shi, Dan Zhang, Fan Yang, Feodor Kouranov, Ian Swett, Janardhan Iyengar, Jeff Bailey, Jeremy Christopher Dorfman, Jim Roskind, Joanna Kulik, Patrik Göran Westin, Raman Tenneti, Robbie Shade, Ryan Hamilton, Victor Vasiliev, Wan-Teh Chang (2017). The QUIC Transport Protocol: Design and Internet-Scale Deployment.

Yu, A. and A. Benson, T. Dissecting Performance of Production QUIC. Available at: ”https://cs.brown.edu/ tab/papers/QUIC$_{W}WW21.pdf$”$(Accessed:10June2023)$.

R. Hamilton, J. Iyengar, I. Swett, and A. Wilk, “QUIC: A UDP-Based Secure and Reliable Transport for HTTP/2,” [Online]. Available: https://tools.ietf.org/html/draft-tsvwg-quic-protocol-02

Playing with QUIC. https://www.chromium.org/quic/ playing-with-quic/, 2017. [Online; accessed 01-January-2017].

John Dellaverson, Tianxiang Li, Yanrong Wang, Jana Iyengar, Alexander Afanasyev, Lixia Zhang. (n.d.). A Quick Look at QUIC*. https://web.cs.ucla.edu/ lixia/papers/UnderstandQUIC.pdf

Alexander Yu and Theophilus A. Benson. 2021. Dissecting Performance of Production QUIC. In Proceedings of the Web Conference 2021 (WWW '21). Association for Computing Machinery, New York, NY, USA, 1157–1168. https://doi.org/10.1145/3442381.3450103

A. Afanasyev, N. Tilley, P. Reiher, and L. Kleinrock, “Host-to-host Con- gestion Control for TCP,” IEEE Communications Surveys Tutorials, vol. 12, no. 3, pp. 304–342, 2010.

Dong, M., et al.: PCC: Re-Architecting Congestion Control for Consistent High Performance, vol. 1.2. NSDI (2015)

Hassan, M., Jain, R.: High Performance TCP/IP Networking, vol. 29. Prentice Hall, Englewood Cliffs (2003)