Evaluating Performance of HTTP/3 for Video Streaming: A Comparative Study with Previous Versions of HTTP

Thomas Daniel Galligan

Final Year Project Extended Abstract
BSc in Computer Science

Supervisor: Dr. Cormac Sreenan

Department of Computer Science
University College Cork

8th March, 2023

Contents

1 Introduction **1**
- 1.1 HTTP Versions . . . 1
- 1.2 QUIC . . . 1

2 Approach **3**
- 2.1 Shaping . . . 3
- 2.2 QUIC-Compatable Clients . . . 3
  - 2.2.1 QUIC-Go . . . 3
  - 2.2.2 Quiche + cURL . . . 3
- 2.3 Server . . . 4
- 2.4 Data Collection and Graphing . . . 4

3 Findings **5**
- 3.1 Packet Loss . . . 5
- 3.2 Latency . . . 5

4 Conclusion and Future Work **8**
- 4.1 Reflection . . . 8
- 4.2 Future Tests . . . 8
- 4.3 Points to Investigate . . . 8

1 Introduction

HTTP is a application-layer network communication protocol used for transferring data over the World Wide Web [10]. According to the IETF HTTP Working Group, it is "the core protocol of the World Wide Web" [6]. The first HTTP version was originally proposed by Tim Berniers-Lee in 1989 [10], and ever since, both it and the World Wide Web grew in popularity. On top of that, video takes up approximately 53% of global traffic share [14]. As video streaming takes up such a large amount of HTTP traffic, it is important that streaming of video is efficient and that quality of experience of users is high so as to keep users engaged with the content. This project aims to evaluate the performance of HTTP/3 with specific regard to video streaming, and compare the results with other version of HTTP. To do so, we must look at what makes each protocol different from one another.

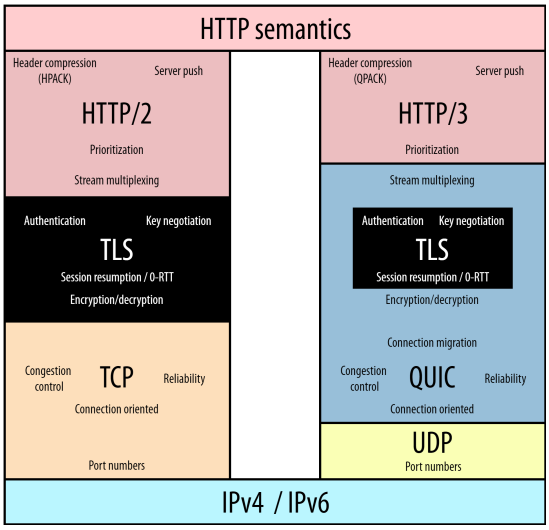
1.1 HTTP Versions

HTTP version standards aim to implement HTTP semantics [3] by way of implementing an application-layer protocol, sitting atop transport-layer protocols. HTTP/0.9, HTTP/1, HTTP/1.1 & HTTP/2 all sit atop TCP (Transmission Control Protocol). HTTP/3 is the newest iteration of the HTTP standard, and sits atop a new protocol; QUIC (QUick Internet Connections). HTTP/0.9 will not be considered for this project, as there are many security implications of adopting HTTP/0.9 into a system, as HTTP/0.9 can't be transported using TLS. As well as this, there is no HTTP header communication possible, as "headers" did not exist as we understand them in modern versions of HTTP[7]. As well as this, HTTP/1.0 also will also not be considered as almost all modern webservers make use of `vhosts`, however, HTTP/1.0 does not include a `Host` header field to allow a single IP to be used for multiple different domain names[12]. This is a problem as IP address space is limited, and IP addresses are expensive.

HTTP versions considered for testing in this project are as follows: HTTP/3, HTTP/2 & HTTP/1.1.

1.2 QUIC

QUIC is a general-purpose transport-layer protocol developed by Google since 2012 [2] and standardized in 2021. QUIC itself sits atop another trasnport-layer protocol, UDP (User Datagram Protocol), which is responsible for the physical delivery of packets over IP (Internet Protocol). QUIC, being a general purpose trasnport-layer protocol, can, in theory be used for any application-layer protocol, but one of its use-cases is for HTTP/3. It also aims to be a secure protocol by way of encapsulating TLSv1.3 into the stack of protocols that make up QUIC (see Figure 1.1.



(Marx, Smashing Magazine 2021)[9]

Figure 1.1: HTTP/2 and HTTP/3 Protocol Stack

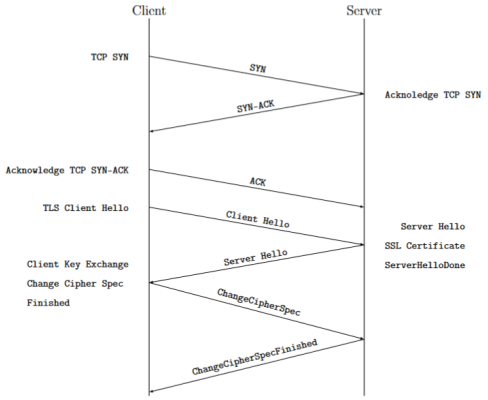


Figure 1.2: TCP and TLS Handshake

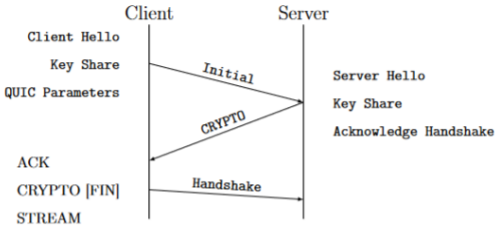


Figure 1.3: QUIC Protocol Handshake

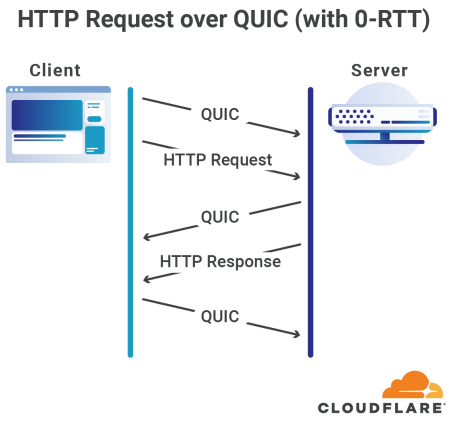


Figure 1.4: QUIC 0-RTT Connection Resumption. Source: Cloudflare Blog[4]

2 Approach

Investigating the performance of both HTTP/3 and previous versions of HTTP for video streaming was carried out in multiple different ways to conduct a fair comparison between the different versions. Experiments focused on downloading files of sizes comparable to AVC's (Advanced Video Codec) 264 encoded video segments for DASH (Dynamic Adaptive Streaming over Http), also since its the most commonly used video codec [1] and were conducted under a network connection of up to 1Gb/s download speed and approximately 100Mb/s upload speed. The ping from the client to the test server was 23.46ms with a standard deviation of 0.976ms. All experiments were ran with the same client link speeds and bandwidth, but network conditions were altered by way of 'shaping' the connection at the server.

2.1 Shaping

Network shaping is often used to introduce a variable network condition to a connection. This was done throughout the experimentation for this report by way of using the `tc` (Traffic Control) command. tc is a command-line utility that allows for the manipulation of the Linux kernel's traffic control subsystem. `tc` is used to shape the network connection by way of introducing delay, packet loss, and other network parameters. It is used to simulate potential real-world scenarios, and to isolate the effects of a single variable on the network connection, without having to change multiple variables at once.

2.2 QUIC-Compatable Clients

2.2.1 QUIC-Go

One tool used to investigate performance was the `quic-go` client. `quic-go` is an open-source QUIC server and client implementation written in the Go programming language. It is a library that can be used to send HTTP/3 or basic QUIC requests to a server. Some of the experiments use it to test the performance of HTTP/3 by way of sending HTTP requests to a server, and recording the time it takes for the server to respond and return the full response body.

2.2.2 Quiche + cURL

Another client-side tool used was Cloudflare's `Quiche` library compiled into the `cURL` command-line utility. `Quiche` is an open-source QUIC implementation written in the Rust programming language by Cloudflare. It is a library that can be used to send HTTP/3 or basic QUIC requests to a server. `cURL` is a command-line utility that can be used to send HTTP requests to a server. It is used to test the performance of HTTP/3 by way of sending HTTP requests to a server, and recording the time it takes for the server to respond and return the full response body.

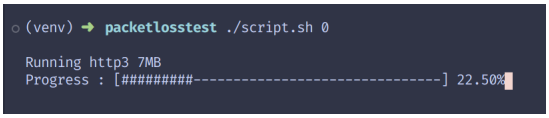


Figure 2.1: Screenshot of Quiche + CURL test environment

2.3 Server

The server implementation used for these experiments was NGINX with its new QUIC branch, which introduces support for HTTP/3 and QUIC. The student chose to use this webserver, as it is the most popular webserver in use [11]. As OpenSSL (the TLS library normally used by NGINX) does not yet support TLSv1.3, the student compiled Google's BoringSSL library that does support TLSv1.3. While compiling NGINX, the student linked the static objects from the compiled BoringSSL library to the C compiler used for compiling the NGINX source code. This allowed NGINX to use BoringSSL as its TLS library, and thus support TLSv1.3. The student then compiled NGINX with the QUIC stream and HTTP/3 module. The student left the default configuration options for the NGINX HTTP server configurations, but did disable compression, as this would likely give an unfair advantage to HTTP/1.1 and HTTP/2.

2.4 Data Collection and Graphing

The student also wrote Bash scripts to automate testing of the servers' response time, when using the `cURL` command, taking in parameters such as the magnitude of latency and/or packet loss. The script would then be used to iterate over a range of files on the server and download each, one at a time over HTTP/3, HTTP/2 & HTTP/1.1. The script would then record the time to download the full file into a file unique for that test environment and protocol. Then, a `Matplotlib` was used to graph the results of the tests, to provide a visual manner of identifying trends in the data.

3 Findings

As was to be expected, HTTP/3 is still in its infancy, and is not yet supported in the Linux kernel. This means that there are many performance issues that will hopefully be improved with time, as optimization for UDP packets is not as well implemented. This can be seen in flamegraphs [5] I generated for HTTP/2 (Figure 3.3) and HTTP/3 (Figure 3.4), where a 5 second `perf` trace picks up a HTTP request for a 12 Megabyte file. The Flamegraph for QUIC transport noticeably features a large amount of CPU time when compared to the TCP flamegraph (approximately 0.3s for UDP, and 0.01s for TCP — not including logging, making UDP take 30x longer in terms of CPU time). This can likely be explained by the fact that UDP normally does not need to carry nearly as much data as TCP does, and as such, high throughput of data has not been a priority in the Linux kernel, meaning optimizations have likely not been prioritized for UDP, but hopefully, if QUIC and HTTP/3 catch on, optimizations will come in time.

3.1 Packet Loss

QUIC has mutltiple sophisticated mechanisms to detect loss and recover lost data. These mechanisms seem to perform better than TCP's. For a packet to be deemed lost, a packet must not receive an acknowledgement from the sender, but a packet that was sent afterwards has been acknowledged. QUIC uses Ack-based loss detection, meaning that packets must be acknowledged as having been received by the client. However, this does seem to come at the cost of increased requests being sent from client to server when compared with TCP to acknowledge sent packets.

Packet loss was varied by using the `tc` utility, prior packet loss to variance was negligible. As can be seen in Figure 3.1, as loss increases, as does download time massively for HTTP/1.1 and HTTP/2. However, for HTTP/3, download time doesn't increase with loss nearly as much as for the other HTTP versions.

For each request, a new connection is created. QUIC's handshakes seem to be slower than TCP (both using TLSv1.3)

3.2 Latency

Latency is another metric that was measured during experimentation. `tc` was also used to add incremental latency to the server to shape the network conditions.

From Figure 3.2, it can be seen that latency has an effect on all file sizes and all protocols, however, for HTTP/3 the effect is large. This can likely be explained by the loss detection mechanisms in HTTP/3 requiring more acknowledgements for successful packet transmissions.

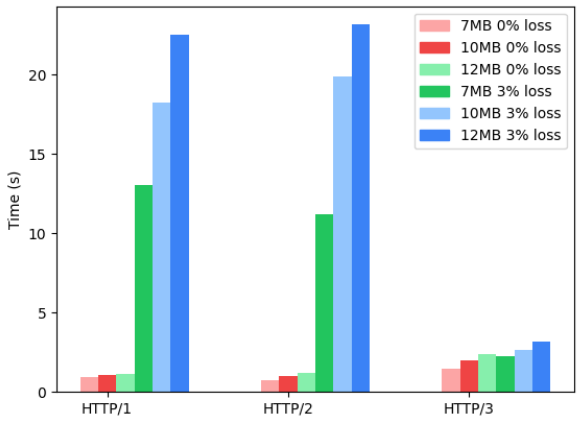


Figure 3.1: Effect of packet loss on download time

Table 3.1: Matrix of network effects on download time for different HTTP versions

Protocol	**As loss increases, download time...**	**As latency increases, download time...**
HTTP/3	...increases minorly	...increases exponentially
HTTP/2	...increases exponentially	...increases minorly
HTTP/1.1	...increases exponentially	...increases minorly

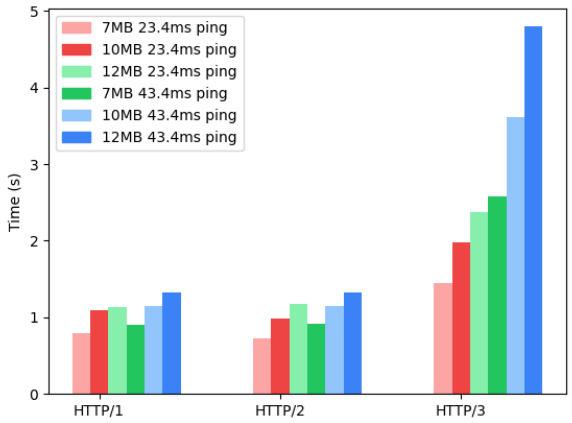


Figure 3.2: Effect of latency on download time

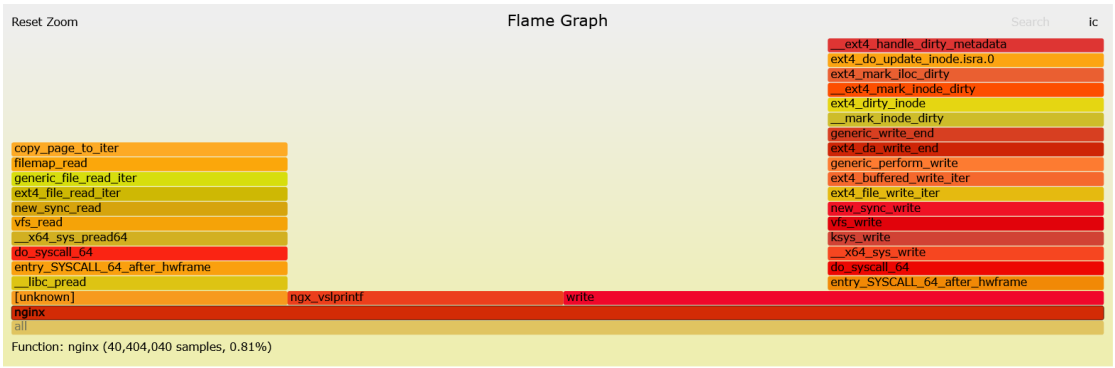


Figure 3.3: HTTP/2 download Flamegraph

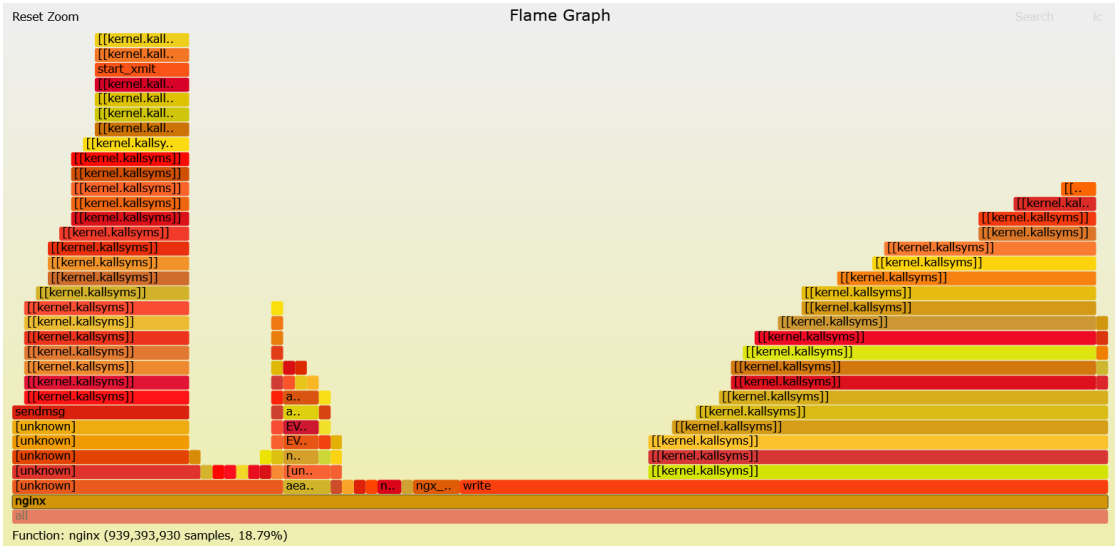


Figure 3.4: HTTP/3 download Flamegraph

4 Conclusion and Future Work

4.1 Reflection

The work carried out during this project aims to investigate the performance of HTTP/3 for video streaming, while comparing with previous versions of HTTP. To adequately investigate this topic, more specific tests will need to be carried out to better outline where HTTP/3 lies.

4.2 Future Tests

One such test would be to run multiple downloads over the same connection (for both HTTP over TCP and QUIC), so as to avoid initiating a complete connection handshake and evaluate performance of the protcols. This would be a likely scenario as with TCP Keepalive normally will outlast the time between fetching one video segment and another. As well as this, the test carried out all negotiated a full QUIC handhshake (see Figure 1.3), which in theory can be reduced to a 0-RTT connection resumption on a returning client, who has already established a TLS connection.

Another relevant test would be to download multiple files over the same connection at roughly the same time. For HTTP/2 and HTTP/3 this will allow the use of streams, and will create an interesting comparison, and perhaps variables such as packet loss can be added to compare the protocols' capabilities of handling such network scenarios.

There are also plans to make use of real-world data collected by researchers conducted in Amsterdam [13]. This would give a good simulation of how HTTP/3 could perform in real life, so as to get a good understanding of how it matches up against HTTP/1.1 and HTTP/2 in a non-ideal network environment.

4.3 Points to Investigate

One of the main problems seen in the project so far is that HTTP/3 seems to perform poorly in ideal scenarios when compared to the other HTTP protocols on a single download. This looks like it may be caused by CPU bottleneck, which is unlikely to be fixed until the Linux kernel upgrades the UDP subsytem [8]. However, there is merit to investigating exactly why, which will be looked at later in this project.

Bibliography

[1] Inc. Bitmovin. Video developer report 2019.

[2] Google Chromium. Quic, a multiplexed transport over udp.

[3] Roy T. Fielding, Mark Nottingham, and Julian Reschke. HTTP Semantics. RFC 9110, June 2022.

[4] Alessandro Ghedini. Even faster connection establishment with quic 0-rtt resumption, Nov 2019.

[5] Brendan Gregg. Cpu flame graphs.

[6] IETF HTTP Working Group. Ietf http working group.

[7] Tim Berners Lee. The original http as defined in 1991, 1992.

[8] Litspeed.

[9] Robin Marx. *Protocol Stack H2, H3.* Smashing Magazine, Aug 2021.

[10] Mozilla. Evolution of http - http: Mdn.

[11] Netcraft, Feb 2023.

[12] Henrik Nielsen, Roy T. Fielding, and Tim Berners-Lee. Hypertext Transfer Protocol – HTTP/1.0. RFC 1945, May 1996.

[13] Darijo Raca, Jason J. Quinlan, Ahmed H. Zahran, and Cormac J. Sreenan. Beyond throughput: A 4g lte dataset with channel and context metrics. In *Proceedings of the 9th ACM Multimedia Systems Conference*, MMSys '18, page 460–465, New York, NY, USA, 2018. Association for Computing Machinery.

[14] Sandvine. 2022 global internet phenomena report. *Internet Phenomena Report*, page 12–13, Jan 2022.