July 2022
Geoff Huston

A Look at QUIC Use

Quick UDP Internet Connection (QUIC) is a network protocol initially developed and deployed by Google, and recently (May 2021) standardized in the Internet Engineering Task Force (IETF) (RFC 9000). In this article we'll take a quick tour of QUIC and then look at the extent to which QUIC is being used on today's Internet.

QUIC is not exactly a recent protocol, as the protocol was developed by Google a decade ago in 2012, and initial public releases of this protocol were included in Chromium version 29, released in August 2013. QUIC is an attempt to refine the basic operation of IP's *Transmission Control Protocol* (TCP), not by fundamentally changing the flow control procedures and stream management per se, but by changing where the transport function is implemented within the end host and, consequently, who has change control over this function.

IP's TCP protocol has been implemented as an operating system function, and applications interact with TCP by an interface that implements the basic I/O functions of open/close and read/write. The details of stream data integrity and flow control are largely hidden from the application. This certainly makes for simpler applications, but this simplicity comes with its own attendant issues. TCP has its problems, particularly so in relation to web-based services. These days most web pages are not simple monolithic objects. They typically contain many separate components, including images, scripts, customized frames, style sheets and similar. Each of these is a separate object and if you are using a browser that is equipped the original implementation of HTTP each object will be loaded in a new TCP session, even if they are served from the same IP address. The overheads of setting up both a new TCP session and a new *Transport Layer Security* (TLS) session for each distinct web object within a compound web resource can become quite significant, and the temptations to re-use an already established TLS session for multiple fetches from the same server are close to overwhelming. But this approach of multiplexing a number of data streams within a single TCP session also has its issues. Multiplexing multiple logical data flows across a single session can generate unwanted inter-dependencies between the flow processors and may lead to *head of line blocking* situations, where a stall in the transfer of the currently active stream blocks all queued fetch streams in the same TCP session. While it makes some sense to share a single end-to-end security association and a single rate-controlled data flow state across a network across multiple logical data flows, TCP represents a rather poor way of achieving this outcome. The conclusion is that if we want to improve the efficiency of such compound transactions by introducing parallel behaviors into the protocol, then we need to look beyond current behaviours of TCP.

Of course, this sounds a whole lot easier than it actually is. Some classes of applications (and services) who would like to see some changes in TCP's behaviour, but it's up to the maintainers of operating systems platforms to implement such changes. This displacement of cost and benefit can often result in an impasse where such changes are inadequately motivated for those who need to make such changes. A more direct outcome can be achieved by allowing the application to exercise direct control over how it wants its transport service to behave.

This is where the UDP protocol comes into play. UDP is a narrow "shim" protocol that exposes the application to the basic datagram behaviour of IP. An application could implement its own implementation of an end-to-end transport protocol and load the control structures of this transport protocol, together with the payload itself and load the combination of the two as the UDP payload. At this point the application has complete control over the transport protocol and can customize the transport protocol behaviour as it wishes without waiting for any third party.

A Brief Summary of QUIC

QUIC is an implementation of an end-to-end transport protocol implemented as an overlay on a UDP datagram flow (Figure 1).

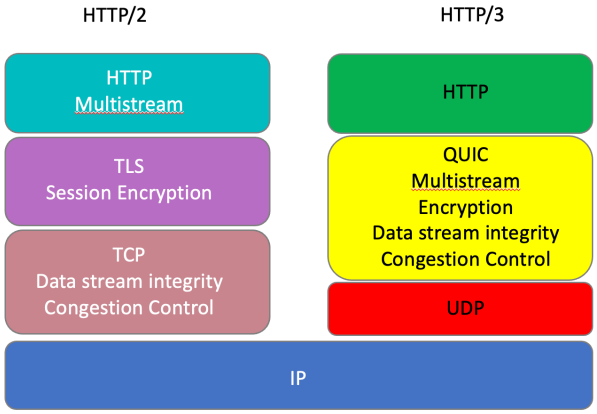

Figure 1 – Comparison of TCP and QUIC within the HTTP architecture

QUIC takes the combinations of TCP's stream integrity and flow control functions, combines it with the session encryption functions of TLS, adds a more flexible version of multi-stream handling, and also adds better support for address agility to tolerate a broad variety of NAT address translation behaviours.

QUIC Connection Identifiers

For flexible NAT traversal QUIC uses the concept of *connection identifiers* (connection IDs). Each endpoint generates connection IDs that will allow received packets with that connection ID to be routed to the process that is using that connection ID. During QUIC version negotiation these connection IDs are exchanged, and thereafter each sent QUIC packet includes the current connection ID of the remote party.

This form of semantic distinction between the identity of a connection to an endpoint and the current IP address and port number that is used by QUIC is similar to the *Host Identity Protocol* (HIP). QUIC'S constant endpoint identifier allows a session to survive changes in the endpoint IP addresses and ports. An incoming QUIC packet can be recognized as part of an existing stream if it uses the same connection ID, even if the source IP address and UDP port numbers may have changed.

QUIC Multi-Stream Support

A single QUIC session can support multiple streams profiles. *Bidirectional streams* place the client and server transactions into a matched context, as is required for the conventional request/response transactions of HTTP/1, for example. A client would be expected to open a bidirectional stream with a server and then issue a request in a stream which would generate a matching response from the server. It is possible for a server to initiate a bidirectional *push stream* to a client, which contains a response without an initial request. Control information is supported using unidirectional *control streams*, where one side can pass a message to the other as soon as they are able. An underlying *unidirectional stream* interface, used to support control streams, is also exposed to the application.

Not only can QUIC support a number of different stream profiles, but QUIC can support different stream profiles within a single end-to-end QUIC session. This is not a novel concept of course, and the

HTTP/2 protocol is a good example of an application-level protocol adding multiplexing and stream framing in order to carry multiple data flows across a single transport data stream. However, a single TCP transport stream as used by HTTP/2 may encounter *head of line blocking* where all overlay data streams fate-share across a single TCP session. If one of the streams stalls, then it's possible that all overlay data streams will be affected and may stall as well.

QUIC allows for a slightly different form of multiplexing where each overlay data stream can use its own end-to-end flow state, and a pause in one overlay stream does not imply that any other simultaneous stream is affected.

Part of the reason to multiplex multiple data flows between the same two endpoints in HTTP/2 was to reduce the overhead of setting up a TLS security association for each TCP session. This can be a major issue when the individual streams are each sending a small object, and it's possible to encounter a situation where the TCP and TLS handshake component of a compound web object fetch dominates both the total download time and the data volume.

QUIC pushes the security association to the end-to-end state that is implemented as a UDP data flow, so that individual streams can be started in a very lightweight manner because they essentially reuse the established secure session state.

QUIC Encryption

As is probably clear from the references to TLS already, QUIC uses end-to-end encryption. This encryption is performed on the UDP payload, so once the TLS handshake is complete very little of the subsequent QUIC packet exchange is in the clear (Figure 2).

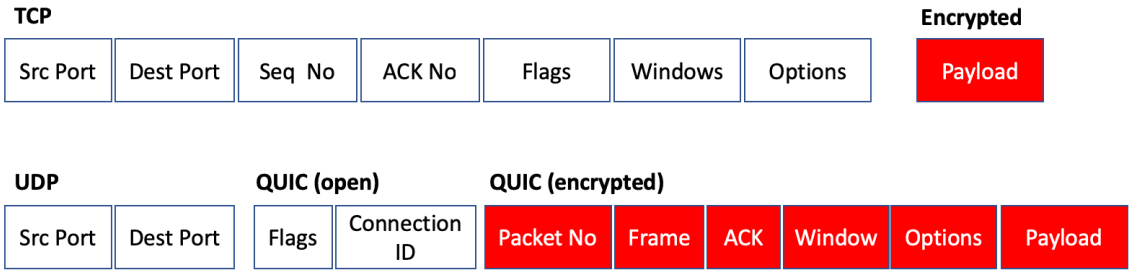

Figure 2 – Comparison of TCP and TLS with QUIC

What is exposed in QUIC are the *public flags*. This initial part of a QUIC packet consists of the connection ID, allowing the receiver to associate the packet with an endpoint without decrypting the entire packet. The QUIC version is also part of the public flag set. This is used in the initial QUIC session establishment and can be omitted thereafter.

The remainder of the QUIC packet are *private flags* and the payload. These are encrypted and are not directly visible to an eavesdropper. This private section includes the packet sequence number. This field used to detect duplicate and missing packets. It also includes all the flow control parameters, including window advertisements.

This is one of the critical differences between TCP and QUIC. With TCP the control parts of the protocol are in the clear, so that a network element would be able to inspect the port addresses (and infer the application type), as well as the flow state of the connection. Connection of a sequence of such TCP packets, even if only looking at the packets flowing in one direction within the connection would allow the network element to infer the round-trip time and the data transmission rate. And, like a NAT, manipulation of the receive window in the ACK stream would allow a network element to apply a throttle to a connection and reduce the transfer rate in a manner that would be invisible to both endpoints. By

placing all of this control information inside the encrypted part of the QUIC packet ensures that no network element has direct visibility to this information, and no network element can manipulate the connection flow.

One could take the view that QUIC enforces a perspective that was assumed in the 1980's. This is that the end-to-end transport protocol is not shared with the network. All the network 'sees' are stateless datagrams, and the endpoints can safely assume that the information contained in the end-to-end transport control fields is carried over the network in a manner that protects it from third party inspection and alteration.

QUIC and IP Fragmentation

The short answer is "no!" QUIC packets cannot be fragmented. The way this is achieved is by having the QUIC HELLO packet be padded out to the maximal packet size, and not completing the initial HELLO exchange if the maximally-sized packet is fragmented. The smallest allowed maximal packet size in 1,200 bytes. The endpoints are permitted to use a larger maximal packet size if they have confirmed the viability of this setting via some Path MTU Discovery procedure.

Configuration for Measuring QUIC Use

Now let's turn to the task of measuring the use of QUIC (and HTTP/3) in today's Internet. To achieve this, we've used APNIC Lab's Measurement platform where the measurement is embedded in a script within an online advertisement. The advertisement script directs the user to perform a number of URL fetches, and the servers that serve the referenced objects are instrumented to allow client capabilities and behaviours to be inferred from the server's actions.

In this case the client is directed to load a basic URL object (a minimal 1x1 pixel 'blot') where the domain name part of the URL is unique to each individual measurement. To set up a QUIC measurement we've taken the following steps:

- used the `nginx` server v1.12.7 with QUIC support included,
- used a URL domain name with a defined HTTPS RR Type whose value is `alpn="h3", and`
- used an Alternative Service directive on the content, namely `Alt-Svc: h3=":443"`, which is intended to direct the client to use HTTP/3 for subsequent retrievals.

> Normally this last measure, the Alternative Service directive, would be largely ineffectual. Each client receives the ad as a single event and the script directs each ad to load once, so a client should not be performing a second load of the URL. In this case we've used a variant of the script that directs the client to wait for 2 seconds and then repeat the load of this URL. It is assumed that this delayed repeat would be sufficient for the client to act on the Alternative Service directive.
>
> We perform this repeated URL fetch one fifth of the time in this experiment.

QUIC Measurements

The QUIC measurement commenced at the start of June 2022. In this measurement we measure both the number of users who query for an HTTPS record and the number of users who use HTTP/3 (QUIC) to retrieve the URL. The results of this measurement for June 2022 is shown in Figure 3.

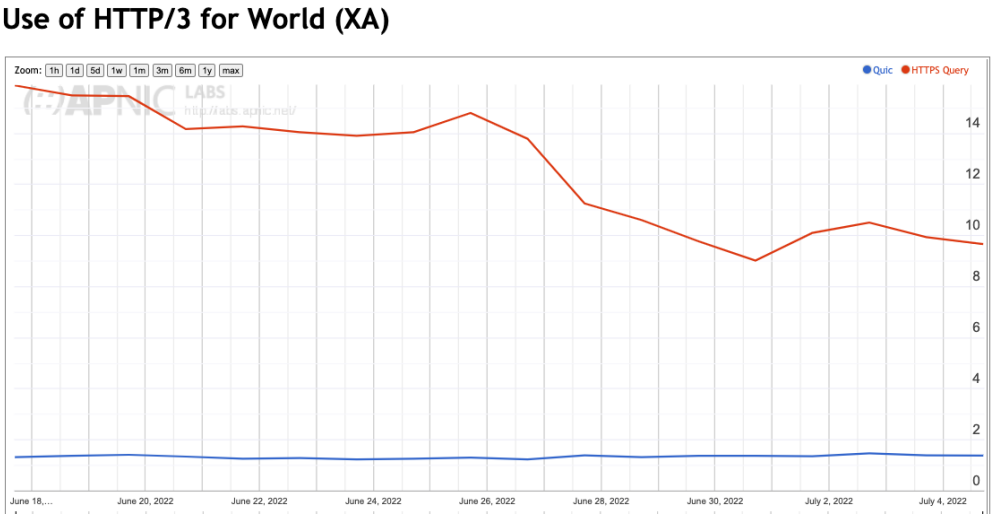

Figure 3 – QUIC measurements for June 2022

It is interesting that the number of users who query for the HTTPS DNS type is between 10% and 15% of users, while the number of users who use HTTP/3 to perform the object retrieval is far lower, at 1.5% of users. If the client browser is aware of HTTP/3 to the extent that it is configured to query for this DNS record type, then why is the subsequent retrieval using HTTP/3 must lower? We'll return to this question later.

It is also interesting to look at the distribution of the use of HTTP/3 for object retrieval on a per-country distribution (Figure 4).

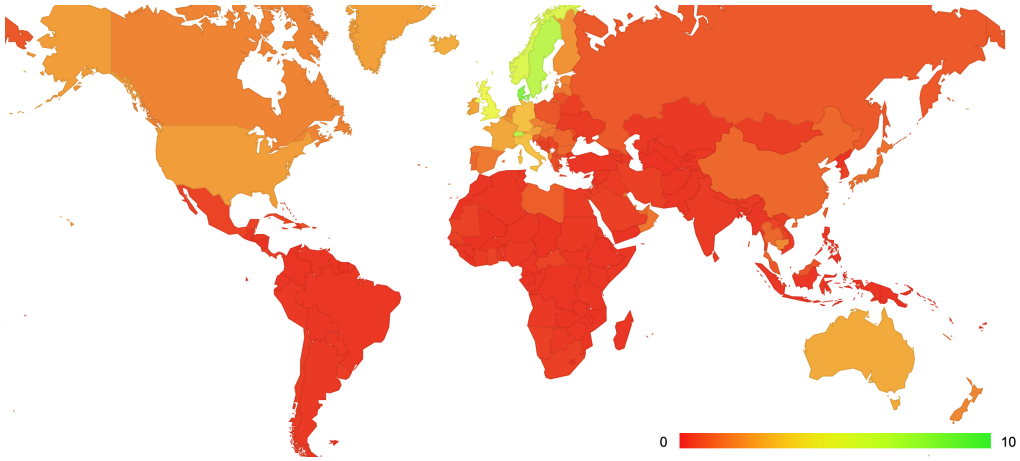

Figure 4 – QUIC use per country

QUIC use is higher in Denmark, Sweden, Norway and Switzerland, while lower in most parts of Asia, South America and Africa. (The full report of data on QUIC use can be found at https://stats.labs.apnic.net/quic)

Which Clients use HTTP/3?

By using the browser string provided in every HTTP retrieval and matching the retrieval to the protocol used (HTTP/2 or HTTP/3) we can produce a profile of which platforms and which browsers are using HTTP/3 for object retrieval. This is shown in Table 1 for platforms. It should be noted that the browser string is not completely reliable as a data source and there is a margin of misattribution here, but the larger numbers provide a reasonable indication of which browsers and which platforms are being used to perform HTTP/3 retrievals.

	HTTP/3	Non-HTTP/3
Android	47.7%	84.5%
iOS	**44.5%**	**5.5%**
Win	10.0%	5.5%
Mac OS X	1.5%	1.0%
Win7/8	1.0%	1.0%
Linux	0.3%	0.4%

Table 1 – Platforms vs HTTP/3 use

Table 1 also provides for comparison a distribution of browser use for non-QUIC use. It's evident that the major shift here is the use of HTTP/3 in Apple's iOS devices (iPhones and iPads) in conjunction with HTTP/3 retrievals.

A similar view, this time for browsers, is shown in Table 2.

	HTTP/3	Non-HTTP/3
Chrome	52.2%	91.7%
Safari	**44.6%**	**4.3%**
Firefox	2.2%	0.8%
Edge	0.6%	0.7%
Opera	0.3%	0.2%

Table 2 – Browsers vs HTTP/3 use

Again, the major variance in Table 2 from the non-QUIC picture is the use of the Safari browser for QUIC use, which correlates well with the iOS platform data in Table 1.

At this stage it appears that a little under half of the observed use of HTTP/3 and QUIC in today's Internet is attributable to iOS platforms and the Safari browser.

QUIC Packet Sizes

As noted in RFC 9000 "UDP datagrams MUST NOT be fragmented at the IP layer." The same specification also states that "Clients MUST ensure that UDP datagrams containing Initial packets have UDP payloads of at least 1200 bytes, adding PADDING frames as necessary." This implies that QUIC implementations must constrain the maximum packet size to a conservative size that comfortable avoids path fragmentation scenarios.

One response to these constraints is to perform some form of Path MTU discovery and use that value. The other approach is to select a packet size that has a relatively high level of assurance that it will not be fragmented.

To answer the question of what maximum packet sizes are being used for QUIC at present we looked at each QUIC session and recorded the maximum packet size. The distribution of observed packet sizes is shown in Figure 5.

The most common session packet size was 1,200 bytes, observed in 46% of sessions. The next most common sizes were 1,250 bytes (18.5% of sessions) and 1,252 bytes (16.4% of sessions). No session had a larger packet than 1,357 bytes in size.

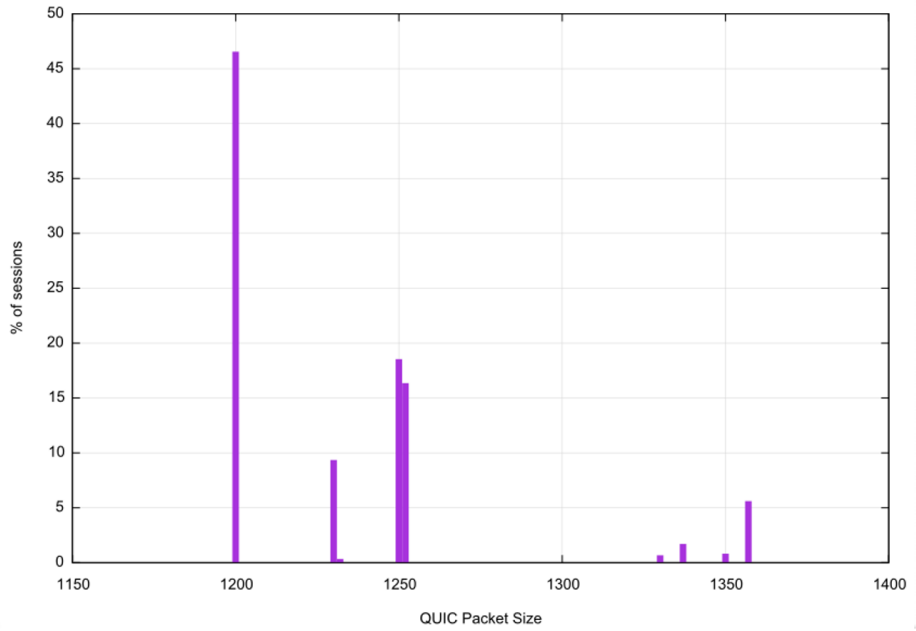

Figure 5 – QUIC maximum packet size distribution

QUIC Connection Reliability

Every new protocol on the Internet has a potential issue with existing middleware. Many packet filters operate within the constraint of a narrow aperture of accepted protocols and ports, and the use of UDP port 443 with an encrypted payload is no exception here. It is reasonable to understand the level of robustness of the QUIC protocol in the Internet.

If we were able to reliably instrument both ends of the QUIC connection we could look at both the reliability of the initial QUIC packet sent from the client to the server, and the responding QUIC packet that is sent back from the server. Unfortunately, we are not in a position to instrument the client so we can only look at the second class of reliability, namely the response packets that are sent back from the server to the client, and whether there is a subsequent packet received from the client.

Table 3 shows the measurements over a 24-hour period for these QUIC response packets. Here we are looking at connections at the server where we see the initial connection packet, but no subsequent packet, indicating a failed QUIC connection. Given that we cannot see failure in the sending of the initial QUIC packet this observed failure rate is a lower bound

Initial QUIC Connections:	19,211,357
Failed Connections:	46,645
Failure Rate:	0.24%

Table 3 – QUIC Connection Failure Rate

Triggering QUIC

There are two mechanisms that allow a server to signal to a client that it is capable of supporting an HTTP/3 session using QUIC, as previous nogtd. To recap these mechanisms are:

- An Alternative Service directive in the content header, namely: `Alt-Svc: h3=":443"`
- A URL domain name with a defined HTTPS RR Type whose value is: `alpn="h3"`

Which one is used by clients?

The way we can measure this is to note that the service directive is only used on the second retrieval, while the DNS-based capability can be established on first use. In this experiment we use two different tests. For one fifth (22%) of the clients who execute the measurement script the client performs two fetches, spaced 2 seconds apart, where the second fetch is distinguished only by an additional field added to the arguments of the URL retrieval. The remainder of the clients perform a single fetch. The two fetches allow the server to inform the client that it can support HTTP/3 in the Alt-Svc header in the first retrieval and then to use HTTP/3 in the second retrieval.

Table 4 compares the HTTP/3 retrieval rate using these two triggering processes. If the client does not use HTTP/3 in the first retrieval, but changes to use HTTP/3 in the second retrieval we assume it used the Alt-Svc. If the client used HTTP/3 in a single retrieval scenario we assume it used the DNS HTTPS mechanism.

Used DNS HTTPS Query	1%
Used Alt-Svc Header	5%

Table 4 – QUIC Connection Trigger Rates

It's evident that the Alt-Svc trigger mechanism is four times more prevalent than the DNS-based mechanism. The reason for this disparity is the use of the DNS in recent releases of the iOS platform using the Safari browser, as compared to the use of the Chrome browser's use of the Alt-Svc directive. Given that Chrome is used by some 90% of the observed clients, then the 5% trigger rate across the total client base corresponds to a 5.5% trigger rate across Chrome clients. The 1% DNS-based trigger rate seen across the entire measurement sample set corresponds to a 20-25% trigger rate across Safari iOS clients.

Is QUIC Faster?

One of the motivations behind the use of QUIC is that the overheads of setting up a QUIC session are lower than the serial delays in setting up a TCP session and then a TLS session. Its useful to ask in the observed practice correlates with this theory.

We will compare the client-measured time to load an object (a 1x1 pixel gif over a secured session) using HTTP/2 and the same client's measured time to load the same object using HTTP/3. There are a number of variables in the client's time measurement, including variance relating to the internal task scheduling within the browser, but these individual factors should be cancelled out over a large enough sample set.

Figure 6 shows the count of time differences in the centre point of +/- 500ms.

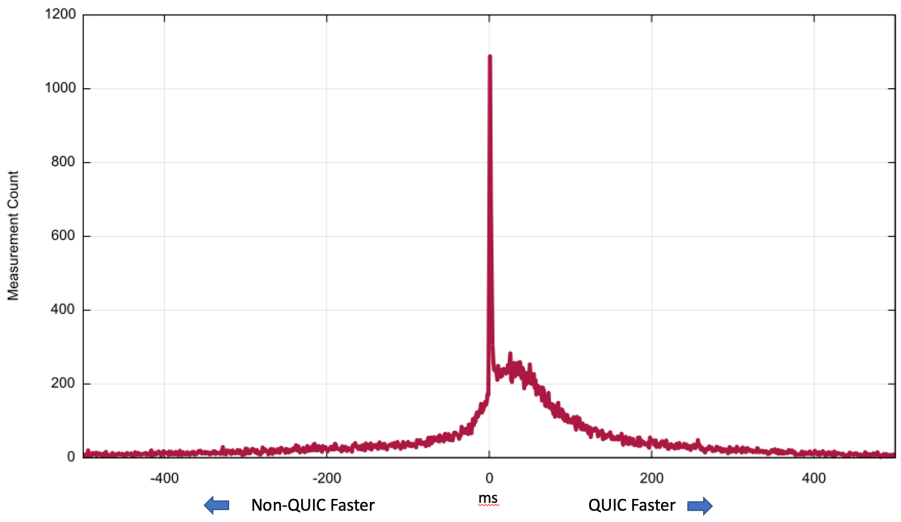

Figure 6 – Comparison of retrieval time differences: QUIC vs non-QUIC

A cumulative distribution shows that QUIC is faster in 2/3 of the samples (Figure 7)

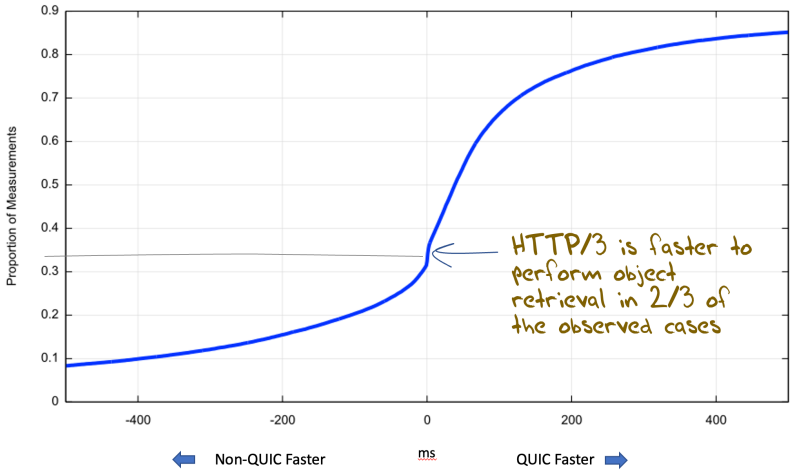

Figure 7 – Cumulative distribution of retrieval time differences: QUIC vs non-QUIC

Summary

It's clear that HTTP/3 and QUIC is gathering some visible deployment momentum in today's Internet, due largely to the impact of Apple's Safari in recent releases of iOS and MAC OS.

QUIC is performing a conservative response to IP packet fragmentation and maximum packet sizes are being held within the range of 1,200 – 1,360 octets.

Thanks to the widespread use of the Chrome browser in the Internet the predominant trigger mechanism for QUIC remains the Alt-Svc directive, although the DNS HTTPS mechanism is being used by Apple clients.

Importantly, HTTP/3 and QUIC is generally faster than TCP and TLS.

We will be continuing this experiment to track the deployment of QUIC over the coming months. The reports are available at https://stats.labs.apnic.net/quic.

Acknowledgements

My thanks to João Damas, senior researcher at APNIC Labs, for his efforts in setting up the QUIC measurement rig, and to Google for their ongoing generous support of the Ad campaigns used to conduct these measurements.

Disclaimer

The above views do not necessarily represent the views or positions of the Asia Pacific Network Information Centre.

Author

Geoff Huston AM, B.Sc., M.Sc., is the Chief Scientist at APNIC, the Regional Internet Registry serving the Asia Pacific region.

www.potaroo.net