The ISP Column
A column on various things Internet

June 2025
Geoff Huston

A QUIC Progress Report

There has been a major change in the landscape of the internet over the past few years with the progressive introduction of the QUIC transport protocol. Here I'd like to look at where we are up to with the deployment of QUIC on the public Internet. But first, a review of the QUIC protocol.

QUIC

QUIC is not exactly a recent protocol, as the protocol was developed by Google in around 2012, and initial public releases of this protocol were included in Chromium version 29, released in August 2013. QUIC is one of many transport layer network protocols that have attempted to refine the basic operation of IP's *Transmission Control Protocol* (TCP).

Why were we even thinking about refining TCP?

TCP is now used in billions of devices and is perhaps the most widely adopted network transport protocol that we've ever seen. Presumably if this protocol wasn't fit for general use, then we would have moved on and adopted some other protocol or protocols instead. But the fact is that TCP is not only good enough for a broad diversity of use cases, but in many cases, TCP is incredibly good at its job.

Part of the reason for TCP's longevity and broad adoption is TCP's incredible flexibility. The protocol can support a diverse variety of uses, from micro-exchanges to gigabyte data movement, transmission speeds that vary from tens of bits per second to tens and possibly hundreds of gigabits per second. TCP is undoubtedly the workhorse of the Internet. But even so, there is always room for refinement. TCP is put to many different uses, and the design of TCP represents a set of trade-offs that attempt to be a reasonable fit for many purposes but not necessarily a truly ideal fit for any particular purpose.

One of the aspects of the original design of the Internet Protocol suite was that of elegant brevity and simplicity. The specification of TCP (RFC 793) is not a single profile of behavior that has been cast into a fixed form that was chiseled into the granite slab of a rigid Internet Standard specification. TCP is malleable in a number of important ways. Numerous efforts over the years have shown that it is possible to stay within the standard definition of TCP, in that all the packets in a session use the standard TCP header fields in mostly conventional ways, but also to create TCP implementations that behave radically differently from each other.

TCP is an example of a conventional sliding window positive acknowledgement data transfer protocol. But while this is what the standard defines, there are some variable aspects of the protocol. Critically, the TCP standard does not strictly define how the sender can control the amount of data in flight across the network. The intention is to strike a fair balance between this data flow across the network and all other flows that coincide across common network path elements (*fairness*) while at the same time not leaving unused network capacity if it can be avoided (*efficiency).* Informally, the objective of each TCP session is to go as fast as possible while not crowding out other concurrent TCP sessions that use common transmission elements. There is a long-standing convention in TCP control algorithms to adopt an approach of slowly increasing the amount of data in flight while there are no visible problems in the data transfer (as shown by the stream of received acknowledgement packets) and quickly responding to signals of network congestion (interpreted from instances of network packet drop, as shown by duplicate acknowledgements received by the sender) by rapidly decreasing the sending rate. Variants of TCP use different controls to manage this *slow increase* and *rapid drop* behavior and may also use different network-derived signals to control this data flow, including measurements of end-to-end delay, inter-packet jitter or packet loss, or explicit signals in the IP and TCP packets headers (ECN). All of these protocol variants still manage to fit with the broad parameters of we collectively call "TCP".

It is also useful to understand that most of these rate control variants of TCP need only be implemented on the data sender (the *server* in a client/server environment). The common assumption of all TCP implementations is that clients will send a TCP ACK packet on both successful receipt of in-sequence data and on receipt of out-of-sequence data. It is left to the server's TCP engine to determine how the received ACK stream will be applied to refine the sending TCP's internal model of network capability and how it will modify its subsequent sending rate accordingly. This implies that deployment of new variants of TCP flow control can be achieved by deployment within service delivery platforms and does not necessarily imply changing the TCP implementations in all the billions of client-side implementations of TCP. This factor of server-side control of TCP behavior also contributes to the viability of TCP evolution.

But despite this considerable flexibility, TCP has its problems, particularly with 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 web "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 attendant 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 one stream blocks all other fetch streams. It appears that while it makes some logical 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, we need to look beyond the existing control profile of TCP.

Why not just start afresh and define a new transport protocol that addresses these shortcomings of TCP? The problem is that there is a significant pool of deployed devices within the network that peek into the transport protocol headers of IP packets in flight. Such devices include load balancers, NATs of various forms, packet "shapers", and security firewalls. In many cases, a common reaction to an unrecognised transport protocol in an IP packet is to simply drop the packet. This mode of behaviour acts as a point of ossification in the network, effectively freezing the generally available protocol set to just TCP and UDP.

If the aim is to deploy a new transport protocol but not trigger active network elements that are expecting to see a conventional TCP or UDP header, then how can this be achieved? This was the challenge faced by the developers of QUIC.

The solution chosen by QUIC was a UDP-based approach.

UDP is a minimal framing protocol that allows an application to access the basic datagram services offered by IP. Apart from the source and destination port numbers, the UDP header adds a length header and a checksum that covers the UDP header and UDP payload. It essentially an abstraction of the underlying datagram IP model with just enough additional information to allow an IP protocol stack to direct an incoming packet to an application that has bound itself to a nominated UDP port address. If TCP is an overlay across the underlying IP datagram service, then it's a small step to think about positioning TCP as an overlay on top of an underlying UDP datagram service.

Using our standard Internet protocol model QUIC is, strictly speaking, a datagram transport application. An application that uses the QUIC protocol sends and receives packets using UDP port 443.

Not only does the use of a UDP *shim* element provide an essential protection of QUIC against network middleware's enforcement of TCP and UDP as the only viable transport protocols, but there is also another aspect of QUIC which is equally important in today's Internet. The internal architecture of host systems has changed little from the 1970's. An operating system provides a consistent abstraction of a number of functions for an application. The operation system not only looks after scheduling the processor and managing memory, but also manages the local file store, and critically for QUIC, provides the networking protocol stack. The operating system contains the device drivers that implementation a consistent view of i/o devices and also contains the implementations of the network protocol stack up to and including the transport protocol. The abstracted interface provided to the application may not be completely identical for all operating systems, but its sufficiently similar that with a few cosmetic changes an application can be readily ported to any platform with an expectation that not only will the application drive the network as expected, but do so in a standard manner such that it will seamlessly interoperate with any other application instance that is a standard-compliant implementation of the same function. Operating System network libraries not only relieve applications of the need to re-implement the network protocol stack but assist in the overall task of ensuring seamless interoperation within a diverse environment. However, such consistent functionality comes at a cost, and in this case the cost is resistance to change. Adding a new transport protocol to all operating systems, and to all package variants of all operating systems is an incredibly daunting task these days. As we have learned from the massive efforts to clean up various security vulnerabilities in operating systems, getting the installed base of systems to implement change suffers from an incredibly static and resistant tail of laggards!

Applications may not completely solve this issue, but they appear to have a far greater level of agility to self-apply upgrades. This means that if an application chooses to use its own implementation of the end-to-end transport protocol it has a greater level of control over that implementation and need not await the upgrade cycle of third-party operating system upgrades to apply changes to the code base. Web browser clients are a good example of this, where many functions are folded into the application in order to provide the desired behavior.

In the case of QUIC the transport protocol code is lifted into the application and executed in user space rather than an operating system kernel function. This may not be as efficient as a kernel implementation of the function, but the gain lies in greater flexibility and control by the application.

A second major aspect of the QUIC design is the tight integration of the TLS functions of server authentication and transport encryption. The entire QUIC payload, including framing, control, and signaling functions in addition to the payload, is encrypted. This encryption of the transport control fields is one of the critical differences between TCP and QUIC. With TCP the control parts of the transport 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. Collection of a sequence of such TCP packets, even if only looking at the packets flowing in one direction within the connection would allow a 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. Placing all 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 QUIC's connection flow. Also, TLS over TCP entails longer connection delays, as the connection sequence entails a TCP handshake and then a TLS handshake. A tighter integration of TLS with the transport protocol can remove a number of round-trip-time delays in connection establishment.

Another major change is the internal structure of the QUIC *connection* using the concept of a *stream,* with multiple streams (of various types) supported in a single connection. There are conventional *bidirectional streams*, *unidirectional streams* and *control streams* supported within QUIC. Each stream has its own flow state, so that the TCP issue of *head of line blocking* when multiple logical data streams are multiplexed into a single TCP stream is avoided. QUIC pushes the security association to the end-to-end state that is implementation as a UDP data flow, so that streams can be started in a very lightweight manner because they essentially reuse the established secure session state. It also uses a single TLS association for the QUIC connection, avoiding the overhead of forming a new TLS association for each stream.

So that's QUIC.

Invoking QUIC

A conventional way to deploy this kind of service is to nominate a transport protocol and a reserved service port (see the "IANA Service Name and Transport Protocol Port Number Registry") and let applications initiate a connection using the nominated protocol and port if they which to access a service using this protocol. If the server is unresponsive, then the application can use an alternative protocol and port selection. In the case of QUIC, this would be a QUIC connection request made via a UDP packet using destination port 443, and the fallback would be a TCP plus TLS connection to destination port 443.

While it's conventional, this connection process can be slow, particularly if the client does not know in advance that the server does not support. In this case the QUIC-enable client would send an opening QUIC packet to UDP port 443 but then must wait for a response. As the client is not necessarily aware of the round-trip time to the server, the client would normally err on the side of caution and wait at this juncture for at least one or two seconds.

QUIC was designed to increase the responsiveness of services over the network, not to introduce new sources of delay. One way to achieve this without paying a time penalty is for the server to signal the capability to use QUIC to the client in the first (non-QUIC) connection, allowing the client to initiate a QUIC session on the second and subsequent connections. This *second-use* approach is defined in an Internet standard for *Alternative Services* (RFC 7838), which is a means to list alternative ways to access the same resources, using the HTTP header fields.

For example,

> **alt-svc:** quic=":443"; ma=2592000;

This indicates that the same material is accessible using QUIC over port 443. The "ma" field is the time to keep this information cached in the local client (in seconds), which in this case is 30 days.

It is not quite as simple as it may sound. The HTTP protocol introduced the concept of session persistence in HTTP/1.0 so that the underlying transport session is kept open for some keepalive interval, allowing reuse of the TCP and TLS state without the session open overheads. The implication for the *second-use* signaling approach was that the original HTTP session was kept open across multiple requests, indefinitely deferring any subsequent connection that would use the QUIC transport protocol. This form of triggering the use of QUIC has been bundled into the Chrome browser for more than a decade.

When Apple introduced support for QUIC in its Safari browser they used a different trigger, namely the DNS. The IETF has introduced the SVCB record in the DNS, allowing a number of connection profile items to be stashed in the DNS (RFC 9460). SVCB records facilitate the discovery of alternative endpoints for a service, allowing clients to choose the most suitable endpoint based on their needs and capabilities. There is the HTTPS DNS record, which is the SVCB record for use for HTTPS service connections. Relevant here is the Application-Layer Protocol Negotiation (ALPN) field where the token **`h3`** denotes a server capability to support QUIC (HTTP/3) (RFC 9114).

Safari browsers that support the use of QUIC will query the DNS for an HTTPS record, and if there is an **`alpn`** field that specifies **`h3`** in this record, then the browser can proceed with opening a QUIC connection to the server.

The current situation is that we have two triggers for the use of QUIC, one as a "second use" in-band trigger, used by the Chrome browser, and the DNS HTTPS trigger, used by Safari

Who uses QUIC?

We are now in a position to provide some answers to this question.

The measurement technique we use for this exercise starts with DNS data for the web object that contains A, AAAA and HTTPS records, and where the HTTPS record data contains an **`alpn`** field value of **`h3`**. The object is a conventional measurement *blot*, namely a 1x1 image in png format, with the HTTP directive of **`alt-svc h3=":443"`**. The measurement script repeats this fetch a further 7 times, at 2 second intervals. The rationale for this repetition is to trigger a connection termination and a second connection which would use the **`alt-svc`** directive, if the client was using this directive.

This measurement has been in operation since mid-2022, and the daily data for the past three years is shown in Figure 1. The data shows a percentage of tested end users, and across this period the average number of measurement tests numbers between 28M to 35M per day.

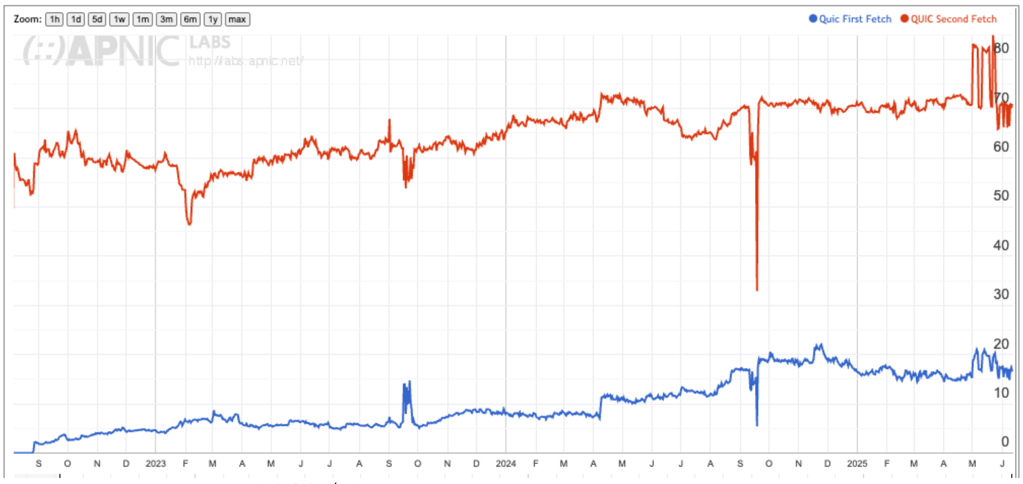

Figure 1 – Use of QUIC (HTTP/3)

Figure 1 shows the relative level of use of QUIC for the initial fetch (blue trace in Figure 1, 15% to 20% of users) and the relative level of the use of QUIC during the subsequent fetches (red trace, 70% of users).

Figure 2 looks at the geographic distribution of QUIC use. QUIC is widely used (more than 75% of users) in most countries across the Internet with the exception of China (15%). There is also partial support in Morocco (40%), Iran (40%) and Nigeria (54%).

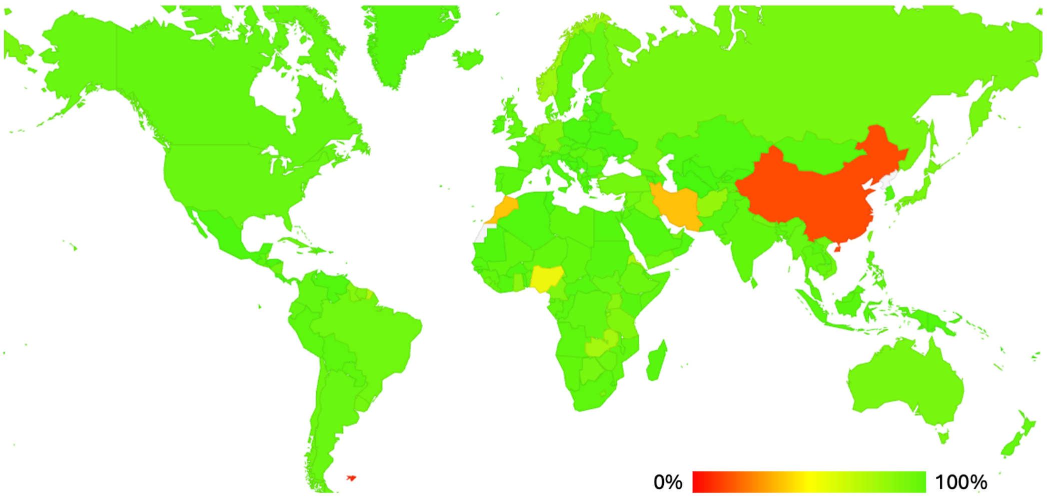

Figure 2 – Geographic Distribution of QUIC support

We can use the browser's user agent string to look at QUIC behaviour by browser – the results for one day of measurement are shown in Table 1.

Browser	Count	HTTPS DNS Query	QUIC 1st Fetch	QUIC Sub Fetc h	NO QUIC
Chrome	9,258,909	174,193	96,824	8,139,214	1,022,871
Safari	5,085,639	4,727,457	2,761,376	20,246	2,304,017
Edge	67,009	47,441	16,776	37,957	12,576
Firefox	12,989	3,689	4,783	5,789	2,417
Opera	4,054	2,218	1,715	1,544	795
Others	29,540	9,828	3,427	15,975	9,838

Table 1 – Browser QUIC Behaviour

The Chrome browser does not perform a DNS HTTPS query and relies on the **`alt-svc`** directive to trigger the use of QUIC on second and subsequent fetches. The Safari browser uses the DNS HTTP query. It is interesting to present the measurements for these browsers in percentage terms (Table 2).

Browser	Count	HTTPS DNS Query	QUIC 1st Fetch	QUIC Sub Fetch	NO QUIC
Chrome	9,258,909	1.9%	1.0%	87.9%	11.0%
Safari	5,085,639	93.0%	54.3%	0.4%	45.3%
Edge	67,009	70.8%	25.0%	56.6%	18.8%
Firefox	12,989	28.4%	36.8%	44.6%	18.6%
Opera	4,054	54.7%	42.3%	38.1%	19.6%
Others	29,540	33.3%	11.6%	54.1%	33.3%

Table 2 – Browser QUIC Behaviour in % terms

It is clear from Table 2 that the Chrome browser does not query for the DNS HTTPS record and relies on the **`alt-svc`** directive to trigger QUIC, and the predominant behaviour of the Safari browser is to query for the HTTPS record, any rely on the data in this record and not the **`alt-svc`** directive. Of interest in Table 2 is the 45% rate of Safari clients who do not use QUIC, as compared to the 11% rate of Chrome clients who do not use QUIC. Is this variance of behaviour between Safari and Chrome browsers an issue with the local network conditions or some issue with the behaviour of the Safari browser?

As an example, let's look at the data collected for users located in the Macao Special Administrative Region of China. On this day our measurement system observed 14,328 individual measurements using the Chrome browser, where 13,029 used QUIC, and the remainder did not. This is a 9.1% failure rate for QUIC. On the same day, we observed 24,700 measurements using the Safari browser, of which only 3,411 individual measurements used QUIC, and the remainder failed. This is a failure rate of 86.2%. The major retail service provider there is AS 4609, which is registered to "Companhia de Telecomunicacoes de Macau SARL". There were 21,022 Safari users who used this provider as their ISP and the QUIC failure rate for these users was 85.2% There were 8,669 Chrome users of this ISP, with a failure rate of 7.4%. A similar relative failure profile between the two browsers is visible for the other two local ISPs "MTel" and "China Telecom Macau". It seems reasonable to conclude that it's not the network itself that is generating this different failure profile between the two browser families. There are many other locales where we observe a similar discrepancy in behaviour between Chrome and Safari. A list of such economies where the difference in QUIC failure rates for these two browsers is greater than 25% is shown in Table 3.

CC	Safari Fail	Chrome Fail	Difference	Name
SR	89.2%	11.5%	77.7%	Suriname
GY	88.3%	11.1%	77.2%	Guyana
MO	86.2%	9.1%	77.1%	Macao, SAR China
EC	78.2%	6.0%	72.2%	Ecuador
BO	76.7%	6.9%	69.8%	Bolivia
PE	75.3%	6.3%	69.0%	Peru
CL	73.7%	5.8%	67.8%	Chile
VE	72.9%	7.3%	65.7%	Venezuela
CO	71.9%	8.4%	63.5%	Colombia
AR	73.0%	9.6%	63.4%	Argentina
CN	92.5%	29.8%	62.7%	China
BR	68.8%	8.2%	60.6%	Brazil
NG	71.9%	13.2%	58.7%	Nigeria
UY	61.0%	6.0%	55.1%	Uruguay

PY	64.0%	12.5%	51.5%	Paraguay
ZM	60.3%	10.7%	49.6%	Zambia
TR	50.9%	7.5%	43.3%	Turkey
NA	48.4%	6.4%	42.0%	Namibia
ID	47.0%	8.8%	38.2%	Indonesia
IN	48.8%	11.6%	37.2%	India
BN	39.1%	3.8%	35.3%	Brunei Darussalam
CU	54.3%	20.3%	34.0%	Cuba
RW	41.2%	8.0%	33.2%	Rwanda
HK	62.4%	32.3%	30.1%	Hong Kong, SAR China
CM	45.1%	16.9%	28.2%	Cameroon
CY	40.1%	12.4%	27.7%	Cyprus
NP	39.5%	11.9%	27.6%	Nepal
PK	37.4%	10.7%	26.7%	Pakistan
SO	39.2%	13.1%	26.2%	Somalia
PA	31.4%	5.5%	25.9%	Panama
SY	40.1%	14.4%	25.7%	Syrian Arab Republic
LB	33.5%	8.2%	25.3%	Lebanon
AO	35.9%	10.6%	25.3%	Angola

Table 3 – Economies where the Safari and Chrome browser QUIC failure rates differ by more than 25%

The significant difference between Safari and Chrome failure rates here tends to (largely) absolve the network infrastructure from blame (with, potentially the exception of China). If the local network infrastructure was blocking UDP port 443, then Safari and Chrome browsers would both be blocked from performing QUIC transactions. The difference in behaviour here between the browsers lies in the use of this new HTTPS query type by Safari. In this measurement we observe the HTTPS query being passed to the authoritative nameservers (As the DNS name contains a uniquely generated component each measurement generates an HTTPS query with a unique query name, conventional DNS caching is bypassed.) In the manner of the measurement we use here we cannot discern if the DNS response is passed all the way back to the querying browser application on the user's device, but in QUIC we are observing a widespread behaviour that is consistent with query-type DNS response filtering being performed in some fashion within the local DNS resolution environment. The HTTPS response appears not to be getting back to the Safari browser.

DNS Ossification?

The introduction of new DNS query types into the end user application environment is extremely uncommon. This HTTPS DNS resource record type is perhaps the first new type that is intended to be used by end systems for many years. End hosts perform A, AAAA and TXT queries and little else.

Yes, I am drawing a distinction here between the query types used in the process of name resolution, used by recursive resolvers and query types used by the application environment in end systems. From this perspective, the DNSSEC validation query types, DS, DNSKEY and SOA, are effectively recursive resolver query types, as very few application environments perform DNSSEC validation directory. Similarly, NS and CNAME queries are used in the name resolution process, but these two query types are not "new" types by any means. What's left are the A, AAAA and TXT queries, which are also by no means "new" query types.

Safari has taken the step of introducing the HTTPS query type into this environment, and the data suggests that on average, about one half of the DNS responses to these HTTPS queries are getting back to the Safari application.

It would not be surprising to learn that in the same way that the Internet's transport infrastructure has ossified to just the TCP, UDP transport protocols, the end-application DNS capability set has similarly ossified, and these days DNS responses other than A, AAAA and possibly TXT records do not necessarily make it back to the querying application.

I have often heard the Internet environment as one that is characterized as enabling *permissionless innovation*. By this it's intended to describe a space that is not only accepting of change, but one that embraces change. The standards that underpin the common technology base that defines a single coherent network are generally minimal in nature, allowing a broad scope for experimentation in diverse behaviours on the network. If any such changes generate positive incentives for broader adoption, then they can be standardized and integrated into the mainstream technology base. However, preserving this flexibility in an ever-growing environment is proving to be a fundamental challenge these days. The massive numbers involved in deployment at the user edge of the Internet, in mobile devices, host computers, consumer premises equipment, NATs, security firewalls and other forms of traffic interception, all add to the inertial load of the Internet and contribute to ossification of the technology base.

If Apple's Safari maintainers were to ask me (and why should they!) I'd gently suggest that in addition to the HTTPS DNS query, the Safari application should also look for the **`alt-svc`** directive in web responses, and if this directive includes the signaling of QUIC capability then the Safari browser should use it for second and subsequent fetches from this server.

Should the same suggestion apply to Chrome, to use the HTTPS DNS query? I'm not sure. Right now, it's a noisy signal, with this measurement suggesting that up to one half of all HTTPS responses apparently don't make it all the way back to the browser application. Yes, the DNS relies on caching to gain performance and scalability but adding an additional query in addition to the A and AAAA queries seems to be adding up to a further 50% to the DNS load profile without clearly understanding the scale of the benefit that would be derived from this additional load. What incremental load are we willing to place on the entire DNS infrastructure to improve the performance of the initial fetch of web objects? I can appreciate Chrome's evident reticence to take this step right now.

And what does this data suggest about the state of the DNS right now? Considerable effort has been invested in the use of the DNS as a highly efficient service rendezvous tool, allowing a client to establish the connection profile for a named service with the use of the SVCB resource record. This is a record that is intended for use by the end use application, and it may well be that this will not be a widely usable mechanism, despite its advantages in improving the performance and flexibility of service access. Maybe all we are left with as a reliable DNS query mechanism for applications to use are the limited collection of A, AAAA and TXT queries.

Disclaimer

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

Author

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

www.potaroo.net