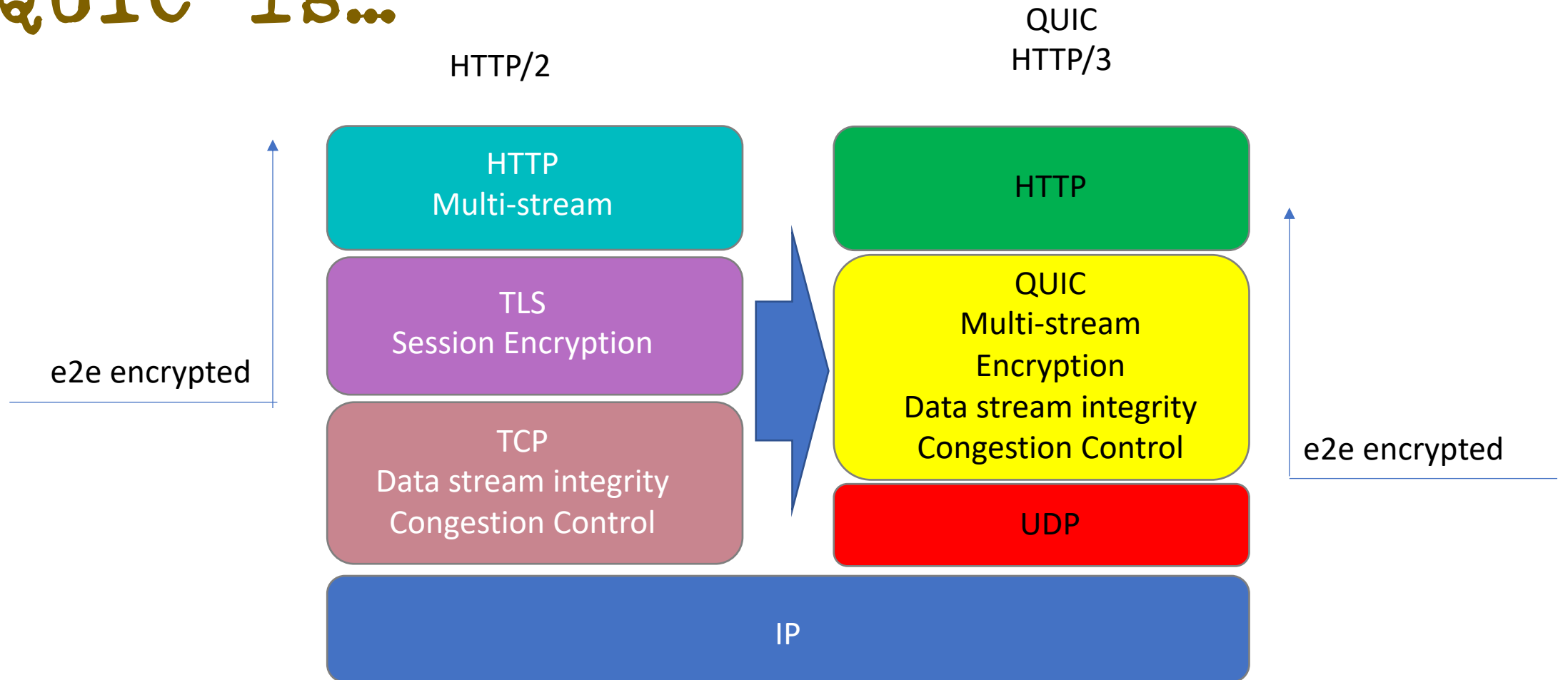


A Quick look at QUIC

Geoff Huston, Joao Damas,

APNIC Labs

QUIC is...



TCP is..

A transport protocol that constructs a reliable full duplex adaptive streaming service constructed on top of an unreliable IP datagram service

- Uses a coordinated state between the two end systems without any network intervention or mediation
- Uses a sliding window to allow lost data to be resent
- Uses ACK-clocking to regulate the sending behaviour to match network path capacity estimate
- Has been tweaked to support (*to some degree) multi-stream and RPC transport models

TCP isn't...

- Fully independent of the underlying platform's transport services
- Fully multi-stream (it has head of line blocking)
- Free from on-the-wire network intervention (TCP control parameters are sent in the clear)
- Has e2e encryption as a second step / afterthought
- Everything for everyone – it relies on the application to perform data framing and in-band control

QUIC is...

Constructed upon a transport level framing protocol that offers applications access to the basic IP datagram services offered by IP through the use of UDP

All other transport services (data integrity, session control, congestion control, encryption) are shifted towards the application. A platform may provide a QUIC API, but the application can also provide its own service

So much more than just “encrypted TCP over UDP”:

Support for multi-stream multiplexing that avoids head-of-line blocking and exploits a shared congestion and encryption state

Faster - Combines transport and encryption setup exchange in a single 3-way exchange

Customisable - QUIC implementations can use individual flow controllers per flow

QUIC places its transport control fields inside the encryption envelope, so QUIC has minimal exposure to the network

Supports record and RPC service models as well as streaming and datagram

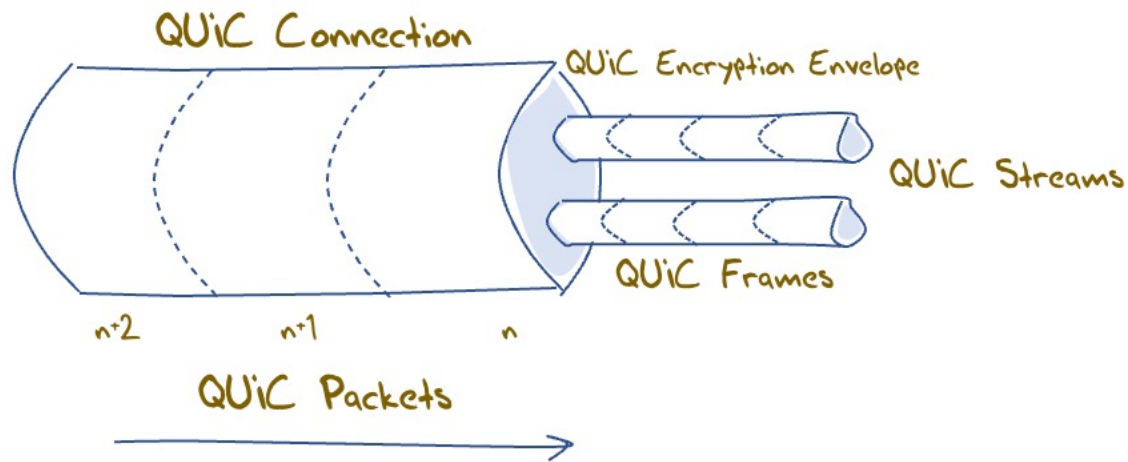
QUIC is address agile

- NATs are potentially hostile to QUIC because of the outer UDP wrapper
 - A NAT may rebind (shift the externally visible address/port of a host during a session), as NATs are not generally aware of UDP streaming states
- QUIC uses a persistent “connection ID”
 - If a host receives a QUIC frame with the same connection ID and a new IP address / port it will send a challenge by way of a random value that should be echoed back. This is all performed within the e2e encryption envelope. That way a QUIC e2e session can map into new address/port associations on the fly

QUIC also...

- Is IP fragmentation intolerant – QUIC uses PMTUD, or defaults to 1,200 octet UDP payloads
- Never retransmits a QUIC packet – retransmitted data is sent in the next QUIC packet number – this avoids ambiguity about packet retransmission
- Extends TCP SACK to 256 packet number ranges (up from 3)
- Separately encrypts each QUIC packet
- May load multiple QUIC packets in a single UDP frame

QUIC flow structuring



A QUIC connection is broken into “streams” which are reliable data flows – each stream performs stream-based loss recovery, congestion control, and relative stream scheduling for bandwidth allocation

QUIC also supports unreliable encrypted datagram delivery

QUIC and RPC

- By associating each RPC request/reply with a new stream, QUIC can support asynchronous RPC transactions using reliable messaging
 - This can handle lost, mis-ordered and duplicated RPC messages without common blocking or throttling

QUIC and Load Balancing

- This assumes that a front-end load balancer is capable of performing load balancing on UDP flows using the UDP connection 5-tuple
- If the remote end performs NAT rebinding the load balancer will be thrown by this shift, and it has no direct visibility into the e2e session to uncover the connection ID
- Using UDP to carry sustained high-volume streams may not match the internal optimisations used in server content delivery networks
- If we really want large scale QUIC with front end load balancing and if we still need to tolerate NATs then we will need to think about how the end point can share the connection ID state with its front end load balancer, or how to terminate the QUIC session in the front end and use a second session to a selected server


QUIC and DOS

- Very little lies outside the encryption envelope in QUIC
- Which means all incoming packets addressed to the QUIC port need to be decrypted
- But the session uses symmetric crypto so the packet decode overhead is far smaller than the asymmetric load
- Its not the best answer, but its not disastrous either!

Looking for QUIC

- At APNIC we use Ads to perform large scale measurements of network service capabilities as seen by users
 - IPv6 deployment
 - DNSSEC validation
 - Fragmentation
- Can we use this measurement platform to see the level of use of QUIC in today's network?

Setting up QUIC

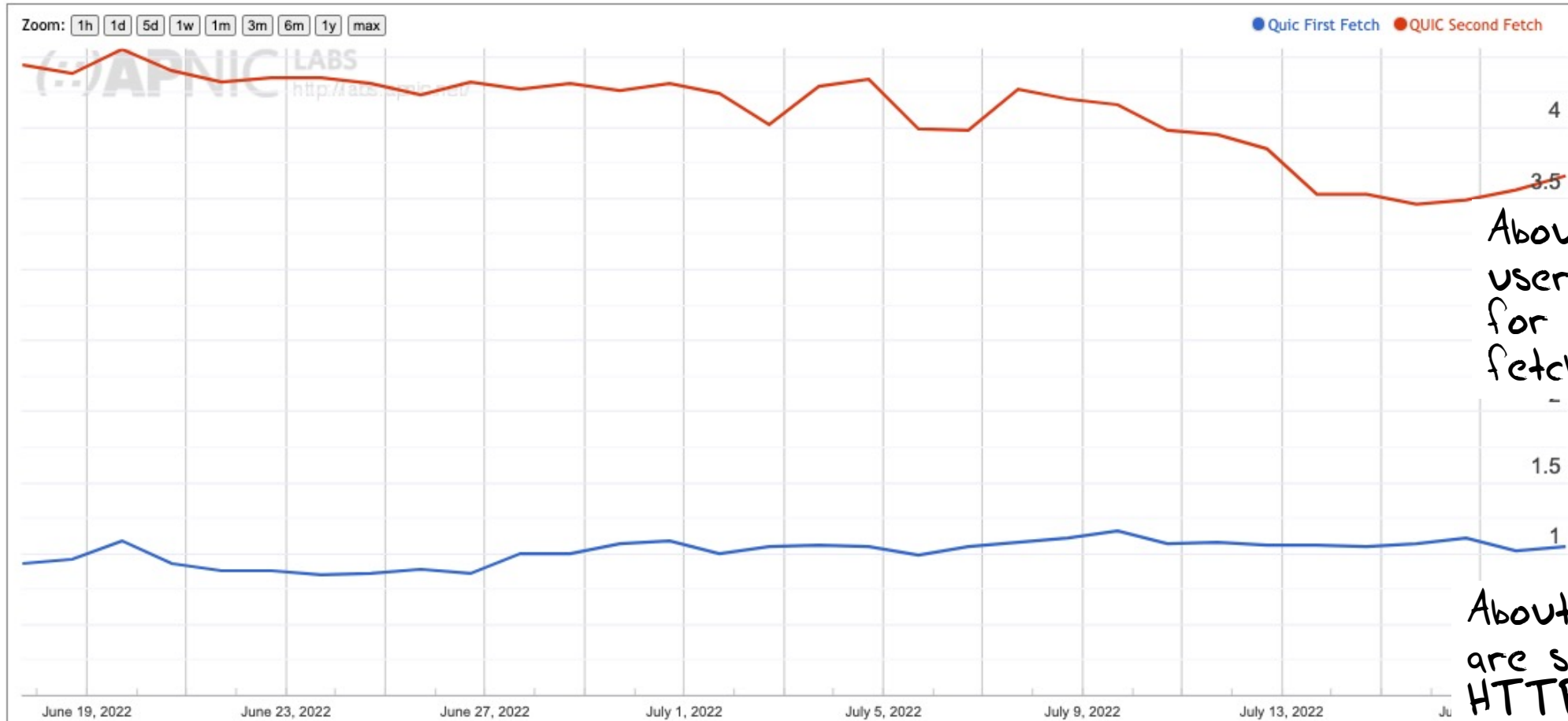
- Server:
 - nginx v1.21.7 with QUIC functions included
- DNS:
 - Set up an HTTPS record for each URL with value: **alpn="h3"**
- Content:
 - **Alt-Svc: h3=":443"** 

 (This second method requires a subsequent query to allow the client to use the Alt-Svc capability. We perform a delayed second query for this URL in the measurement experiment)

Setting up QUIC

- Server:
 - nginx v1.21.7 with QUIC functions included
 - DNS:
 - Set up an HTTPS record for each URL with value: `alpn="h3"`
 - Content:
 - `Alt-Svc: h3=":443"`
- First Fetch
- Second Fetch

QUIC Use - June/July 2022



About 3.5% of users use HTTP/3 for the second fetch

About 1% of users are seen to use HTTP/3 on first fetch

This result looks wrong!

- Some 90% of the browsers we “see” via the ad campaign identify themselves as Chrome
- And Chrome has been supporting a switch to QUIC via the Alt-Svc directive since 2020



Chromium Blog

News and developments from the open source browser project

Chrome is deploying HTTP/3 and IETF QUIC

Wednesday, October 7, 2020

QUIC is a new networking transport protocol that combines the features of TCP, TLS, and more. HTTP/3 is the latest version of HTTP, the protocol that carries the vast majority of Web traffic. HTTP/3 only runs over QUIC.

This result looks wrong!

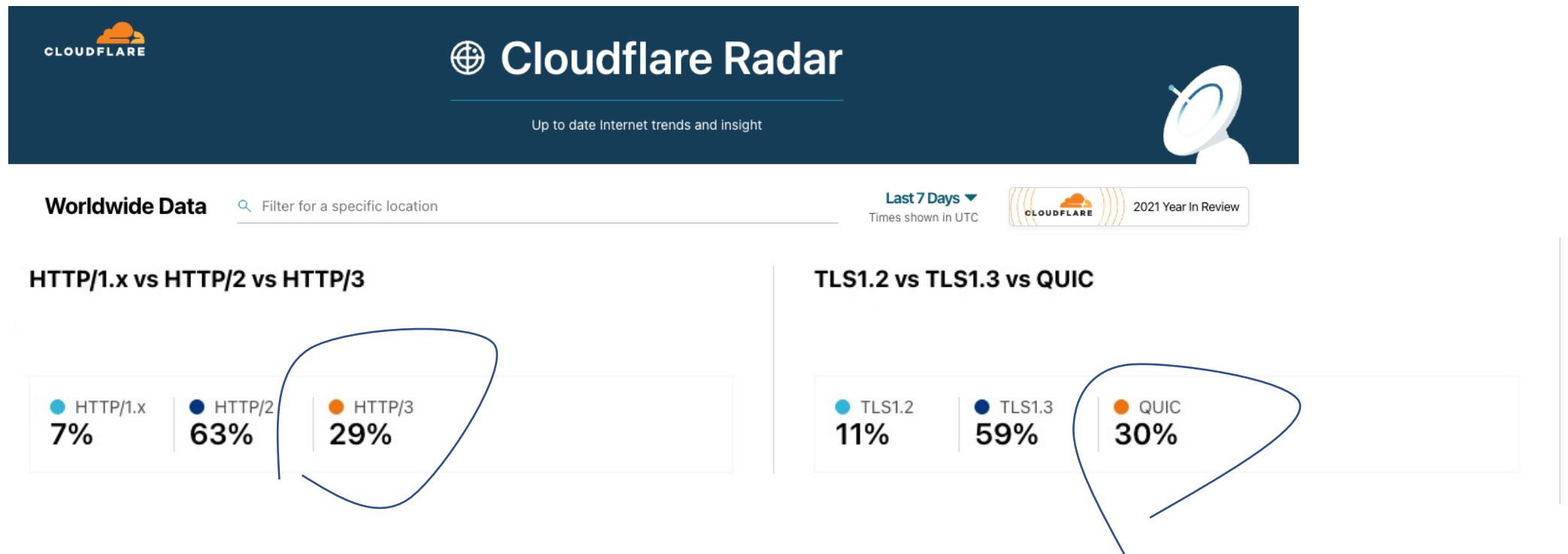
- Some 90% of the browsers we “see” via the ad campaign identify themselves as Chrome
- And Chrome has been supporting a switch to QUIC via the Alt-Svc directive since 2020
- So we should be seeing a far higher level of QUIC use than 3.5%

Hmm

- What do others see?

Cloudflare's Numbers

Our QUIC use numbers are **far lower** than other published measures



Maybe 2 fetches is not enough?

- So we changed the experiment to fetch the same URL 7 times with a 2 second pause between each fetch
- Surely this would flush out QUIC use!

Nope!

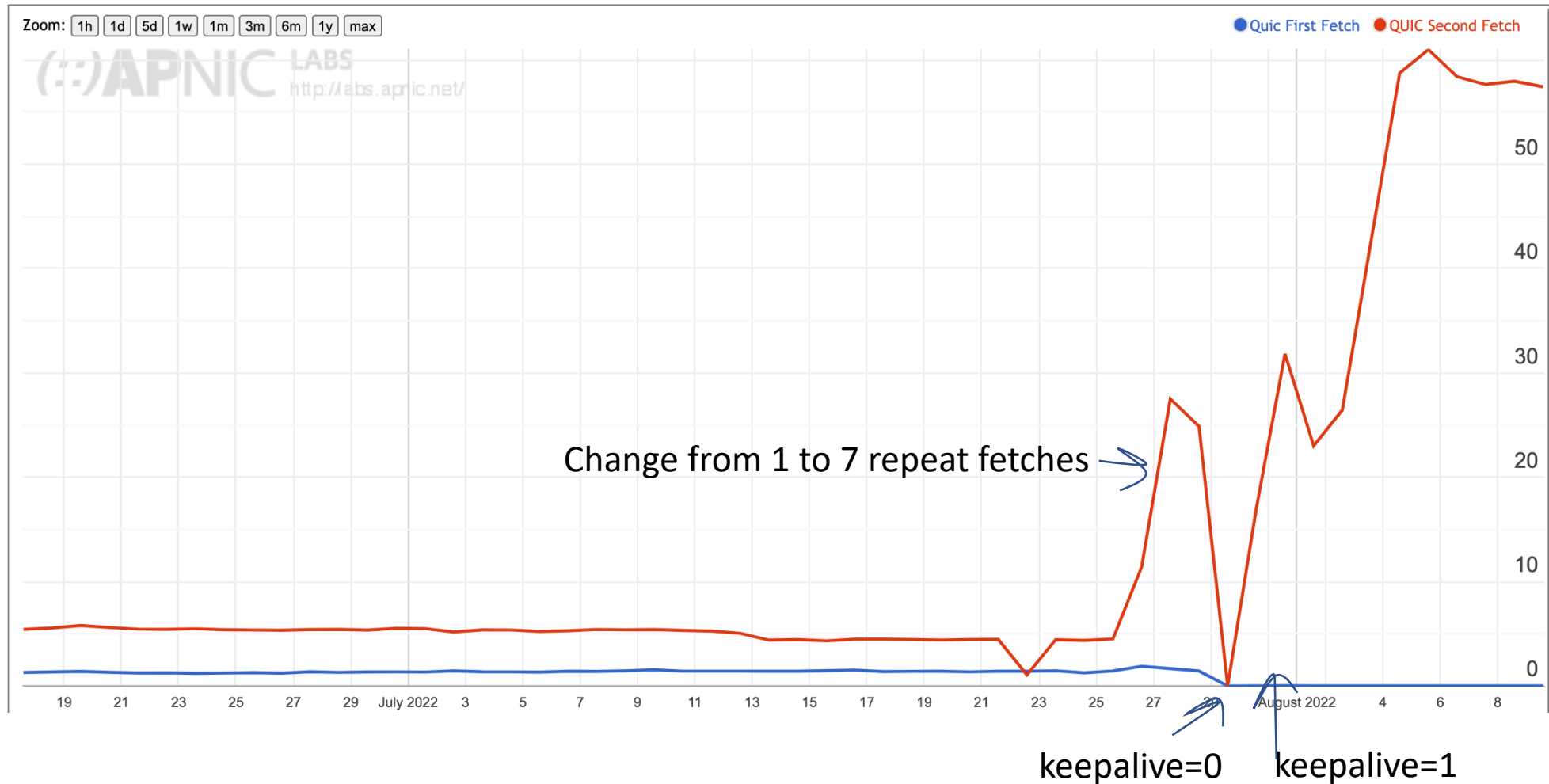
- No change!
- The problem appears to be related to HTTP/2 and persistent connections
- When the browser performs the followup fetch the connection is likely still open and then the browser will prefer to use the open connection over opening up a new QUIC connect
- So on the NGINX server let's set the keepalive session parameter to 0 seconds
- Yes?

Still Nope!

- That was worse!
- We didn't see any use of QUIC at all
- Nothing. Nada. Not even a little bit.

- Seems that if you set keepalive to zero then NGINX disables QUIC completely!
- So we then set the session keepalive parameter to 1 second
- Better?

QUIC Use - June - August 2022



Yes!

- We are seeing a 57% QUIC on the repeat fetches, corresponding to a rate of 63% QUIC use of Chrome clients

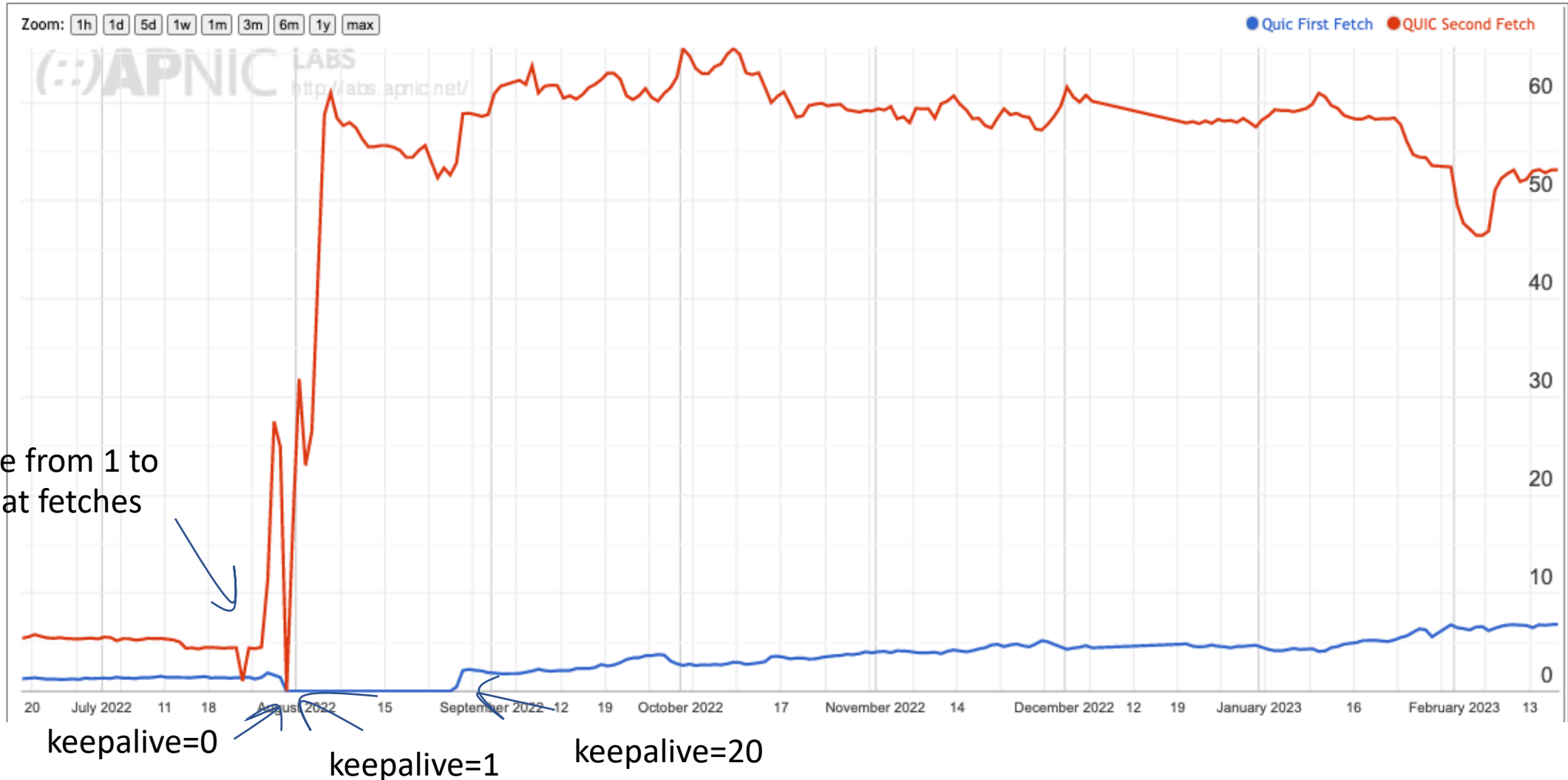
Yes, and No

- We are seeing a 57% QUIC on the repeat fetches, corresponding to a rate of 63% QUIC use of Chrome clients
- But at the same time the first fetch use dropped from 1% to minimal levels
- Which appears to be an issue with Safari, and iOS (and MAC OS)
- And we were seeing Chrome clients revert from QUIC to HTTP/2 across the 7 fetches on a semi-random basis

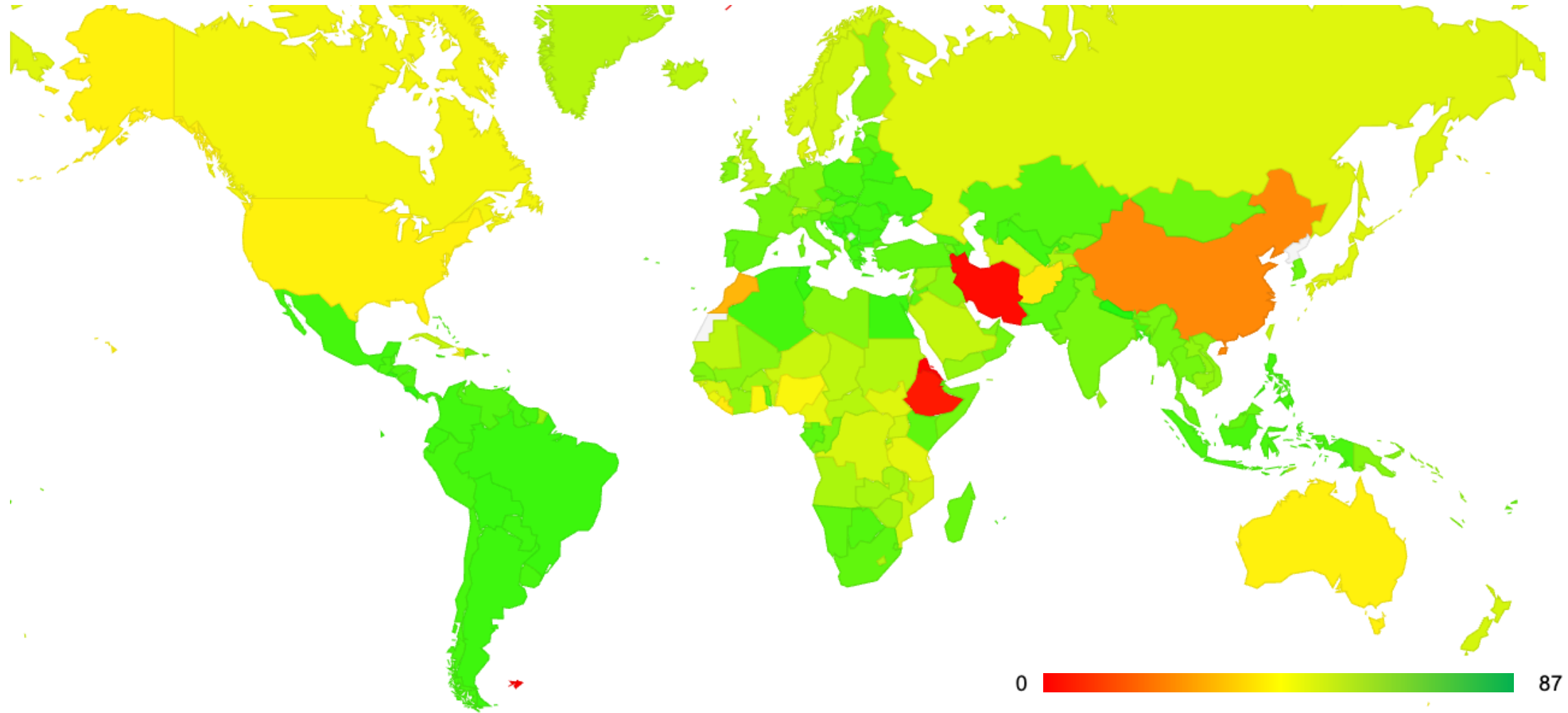
Its all about Keepalive Timers

- After much searching under many rocks we are advised (many thanks to Ryan Hamilton, Martin Thompson and Tommy Pauly for various clues at this stage) that a server keepalive timer value of 1 second is also a Really Bad setting!
- The server is dropping the QUIC connection too aggressively and the browser client drops back to HTTP/2
- The default value of 65 seconds for the server keepalive interval seems to be too long
- And 1 second is too short
- So now let's try a value of 20 seconds...

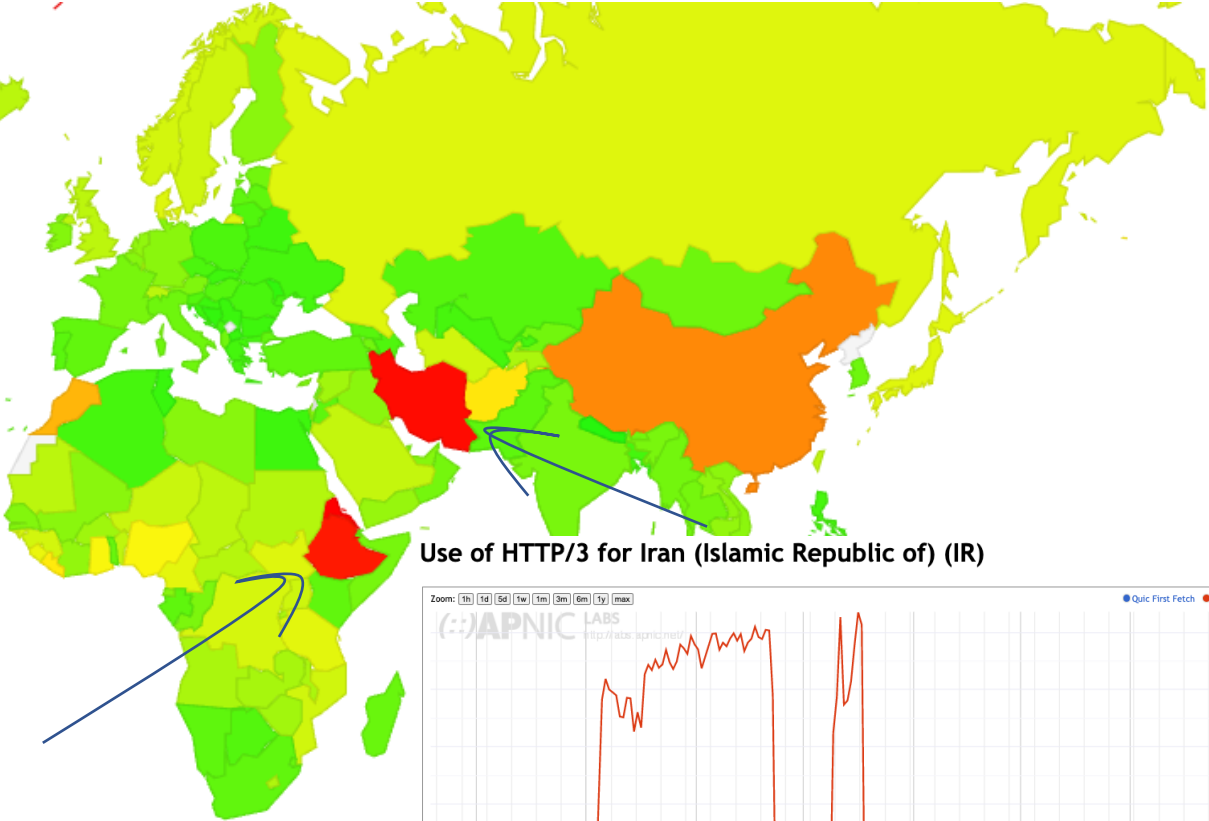
Quic Use



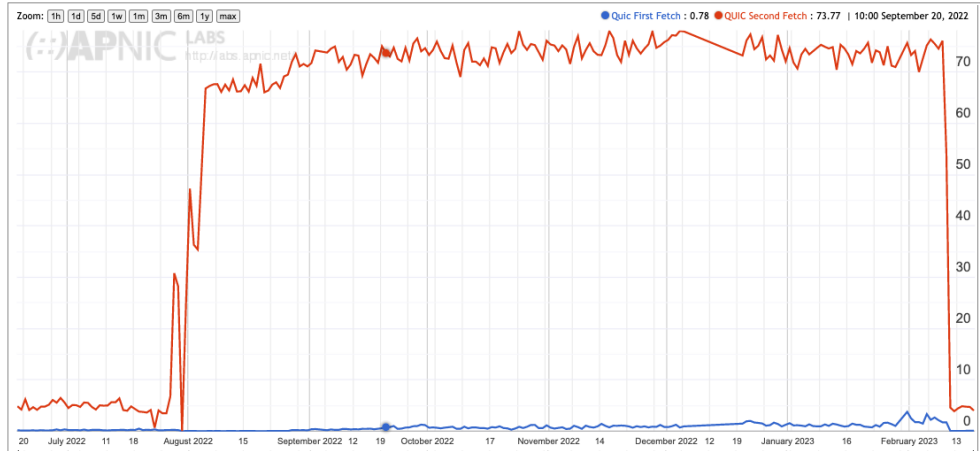
Quic Use - February 2023



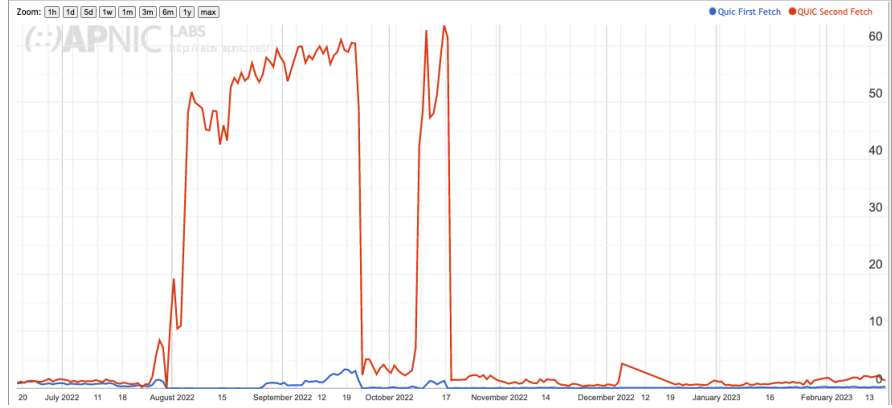
National Filtering of QUIC?



Use of HTTP/3 for Ethiopia (ET)



Use of HTTP/3 for Iran (Islamic Republic of) (IR)



Some Questions:

1. Which clients are performing QUIC and why?
2. What are the QUIC MSS values?
3. What is the QUIC connection failure rate?
4. Is QUIC faster than HTTP/2 + TLS?

1. OS Clients* performing QUIC

	TCP/TLS	QUIC on First Fetch	QUIC on Second Fetch
iOS	5.5%	↑ 93.3%	16.1%
Mac OS	1.0%	2.8%	0.6%
Android	84.5%	1.7%	↑ 77.9%
Win	5.5%	1.4%	4.3%
Linux	0.4%	0.2%	0.2%
Others	3.1%	0.6%	0.9%
	100.0%	100.0%	100.0%

* Based on reported browser string

1. Browser Clients* performing QUIC

	TCP/TLS	QUIC on First Fetch	QUIC on Second Fetch
Chrome	91.8%	4.1%	↑ 81.7%
Safari	4.3%	↑ 93.3%	↑ 16.1%
Firefox	0.8%	2.4%	1.0%
Edge	0.7%	0.0%	0.5%
Opera	0.2%	0.1%	0.6%
Others	2.2%	0.1%	0.1%
	100.0%	100.0%	100.0%

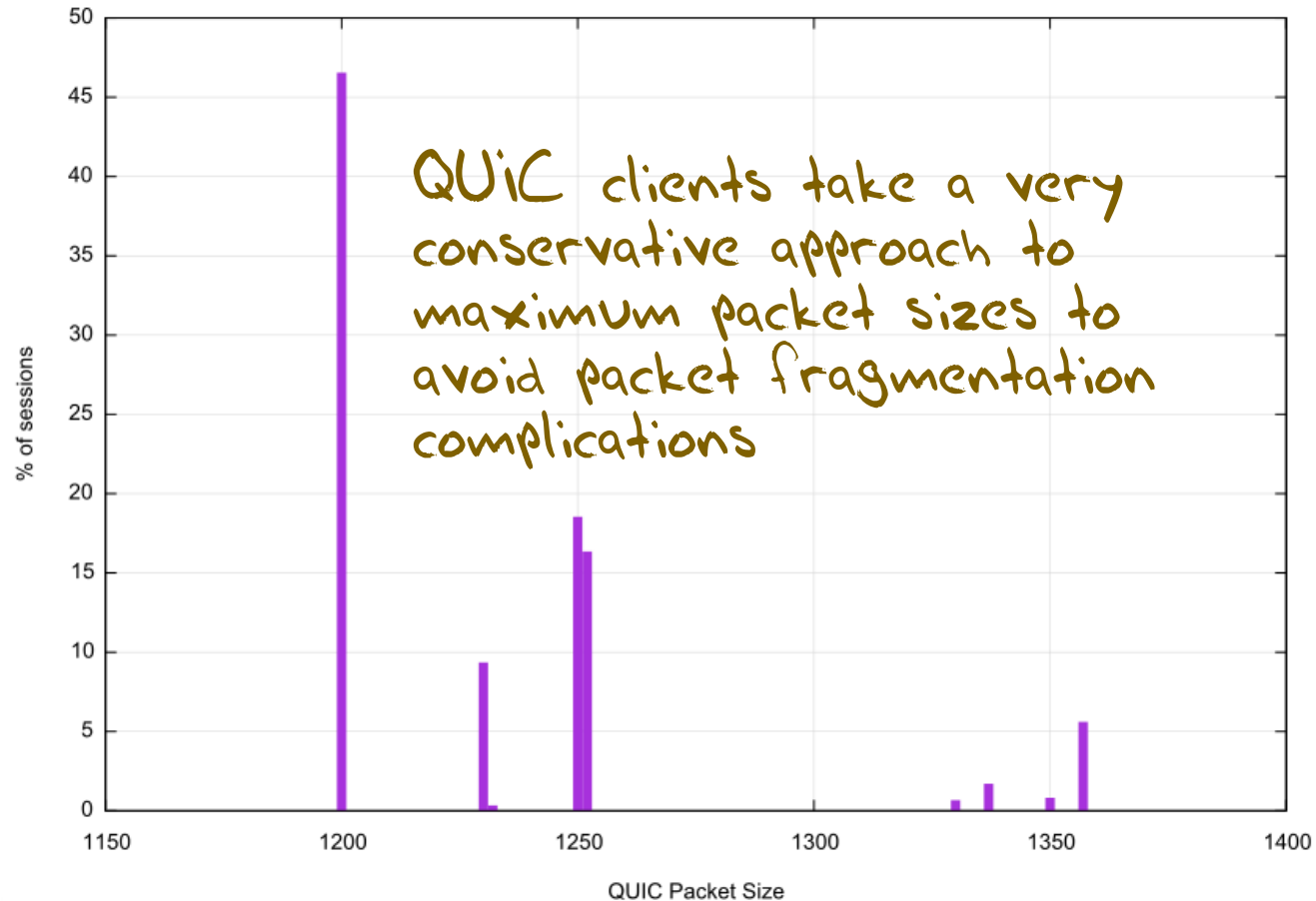
* Based on reported browser string

1. Who does QUIC and why?

Apple Safari clients use a **DNS HTTPS** query and *some* of these clients then follow up with a fetch over QUIC. The observed DNS HTTPS query to QUIC fetch conversion rate was relatively small.

Chrome clients use the **Alt-Svc** field as a QUIC trigger for *most* clients. The observed QUIC conversion rate was high, but not universal.

2. QUIC Packet Size distribution



Maximum Packet Sizes used in QUIC sessions:

1,200 octets – 46.6%

1,250 octets – 18.5%

1,252 octets – 16.4%

3. QUIC Connection Loss

In this measurement framework we cannot measure client -> server loss, but we can measure server-> client loss by looking for incomplete QUIC initial connections that do not complete

(this form of connection loss could be due to the client filtering incoming UDP port 443 packets)

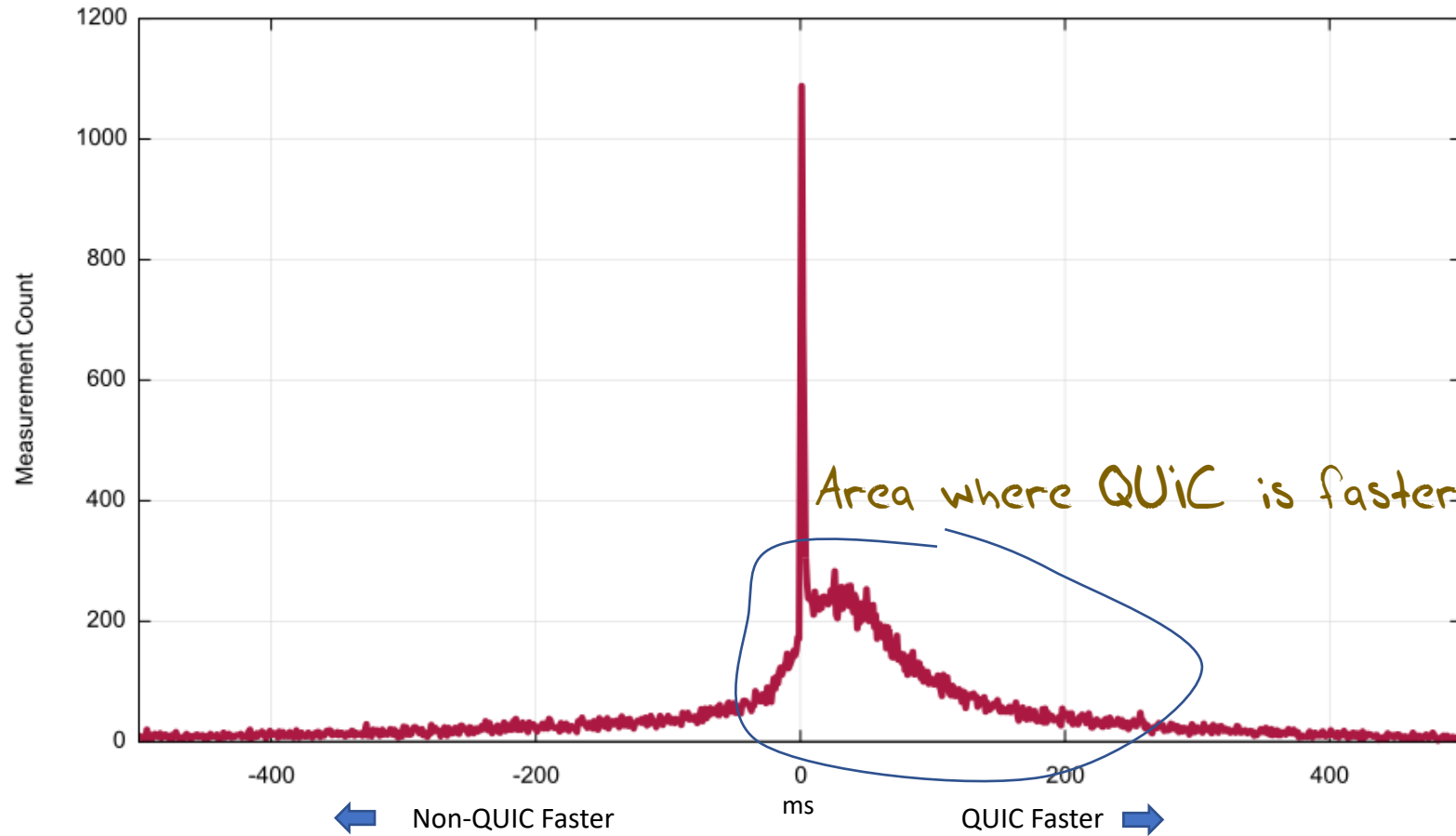
Initial QUIC Connections:	19,211,357
Failed Connections:	46,645
Failure Rate:	0.24%

4. Is QUIC Faster?

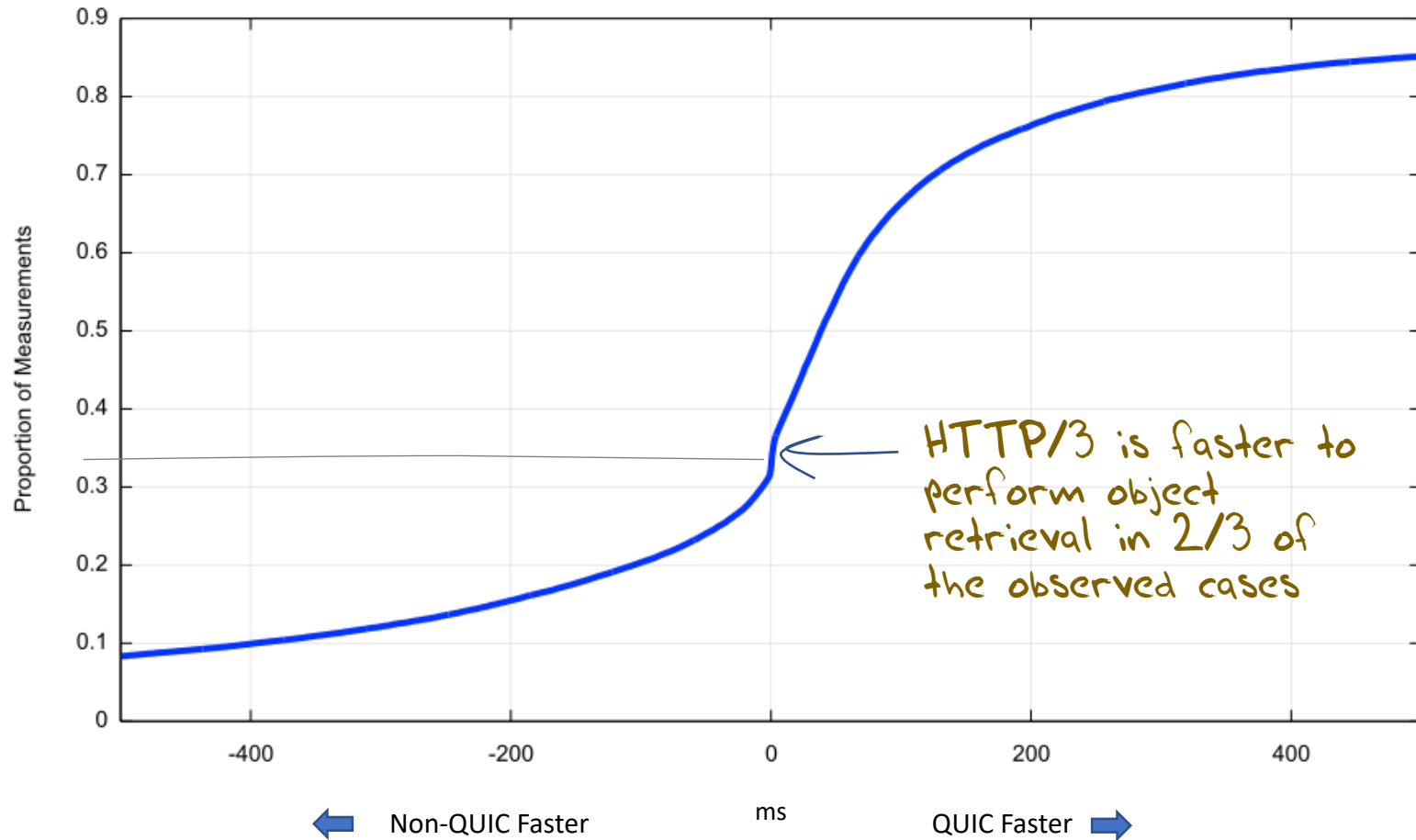
Let's compare the user-measured time to load an object using HTTP/2 and the same user's measured time to load the same object using HTTP/3

- There are a number of variables in the user time measurement, including varying time penalties relating to the internal task scheduling within the browser, but these individual factors should be cancelled out over a large enough sample set

4. TCP/TLS vs QUIC speed difference



4. Cumulative Distribution



Thanks!

**Ongoing HTTP/3 Measurement Report at APNIC Labs:
<https://stats.labs.apnic.net/quic>**