The QUIC transport protocol: design and Internet-Scale deployment

David Calavera

CTO, Netlify

- Motivation
- Design and implementation
- Internet-Scale deployment
- Performance
- Lessons learned
- Conclusion

- Motivation
- Design and implementation
- Testing in Production FTW
- Performance
- Lessons learned
- Conclusion

Motivation

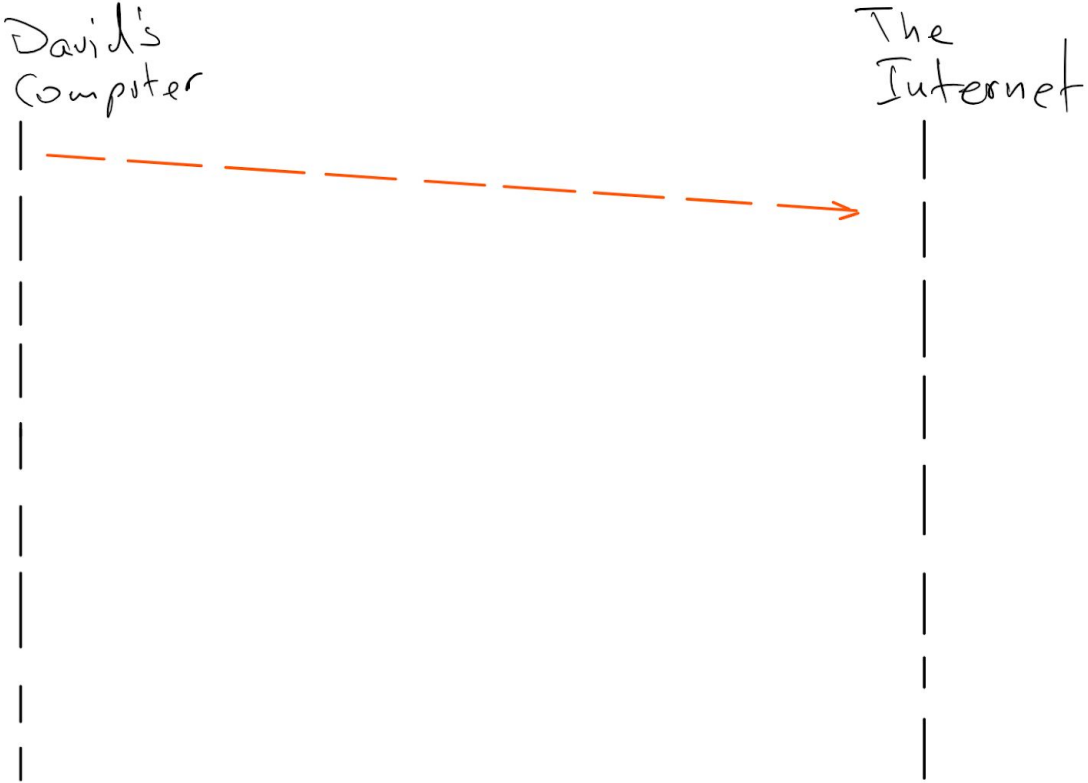
How does the internet work?

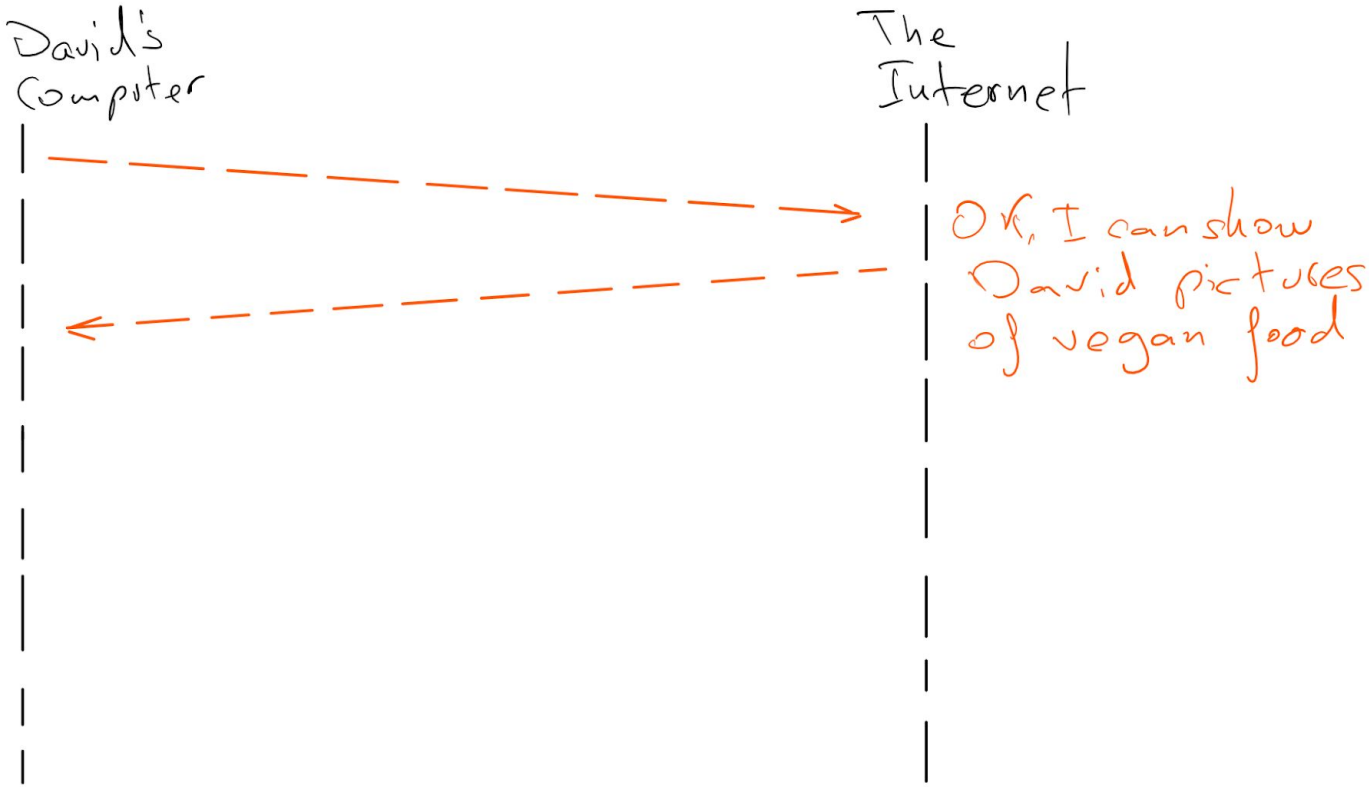
Welcome to my favourite job interview question!

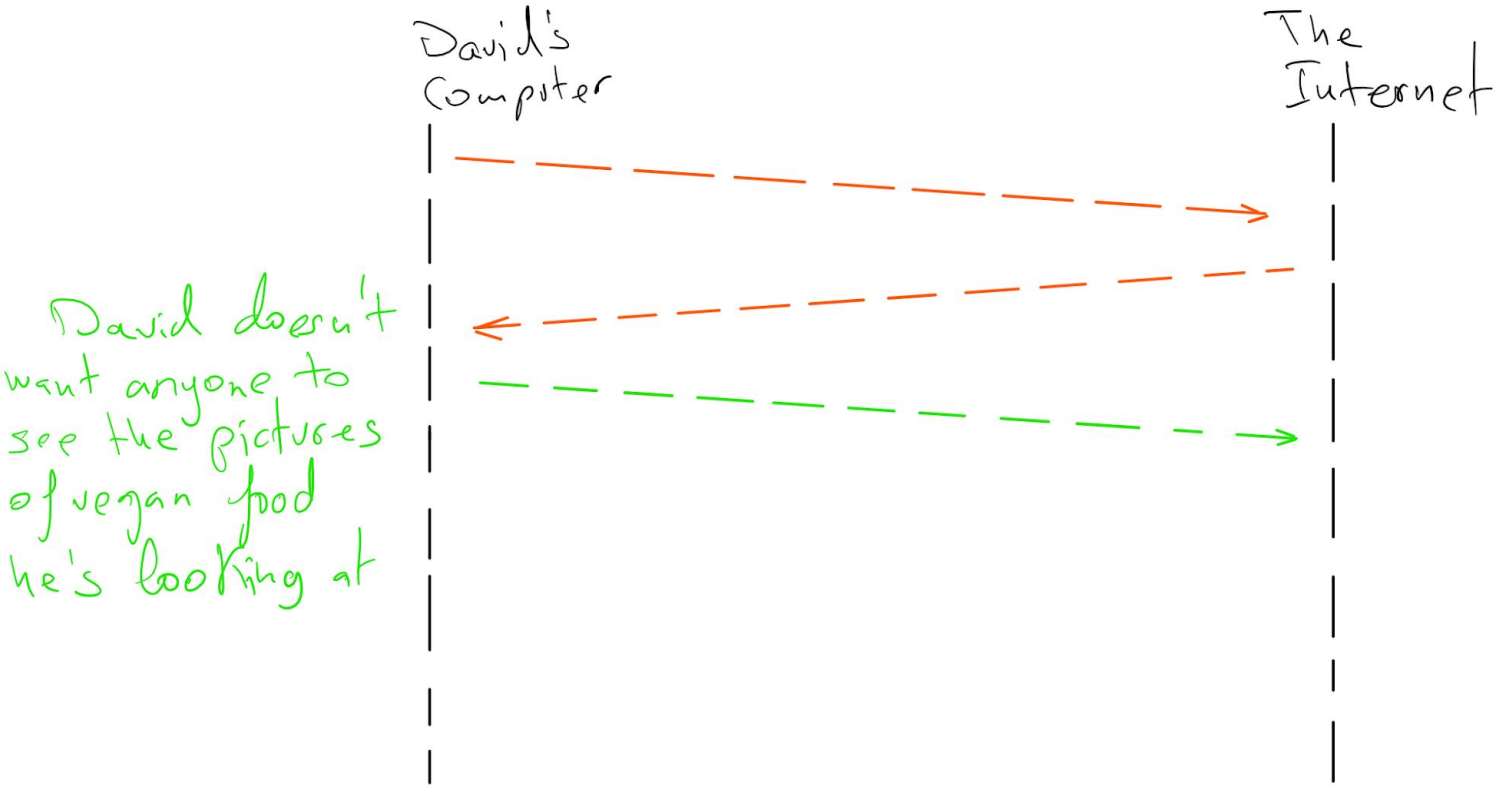
TCP+TLS Handshake Delay

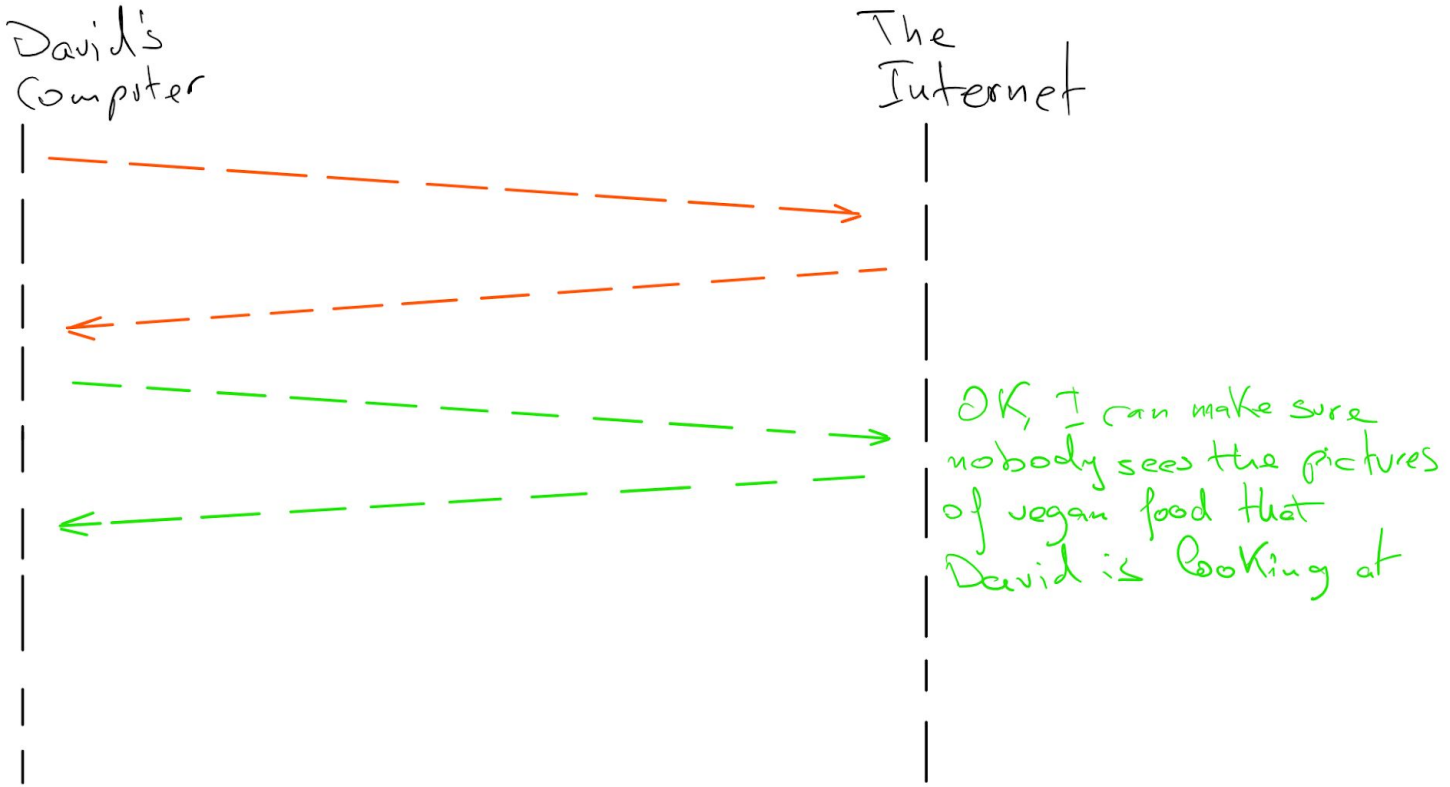
Sooooo chatty

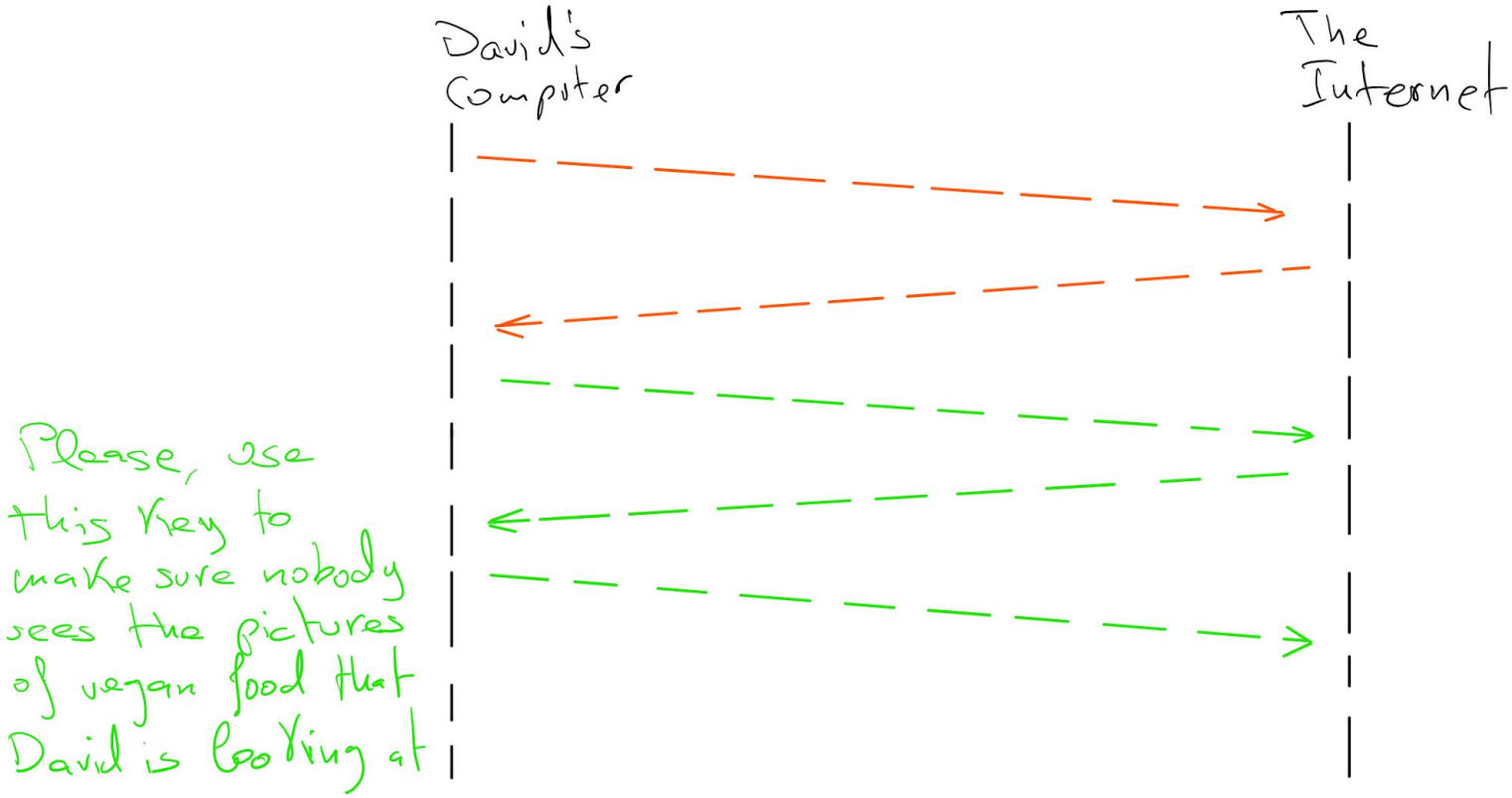
David wants to see pictures of vegan food

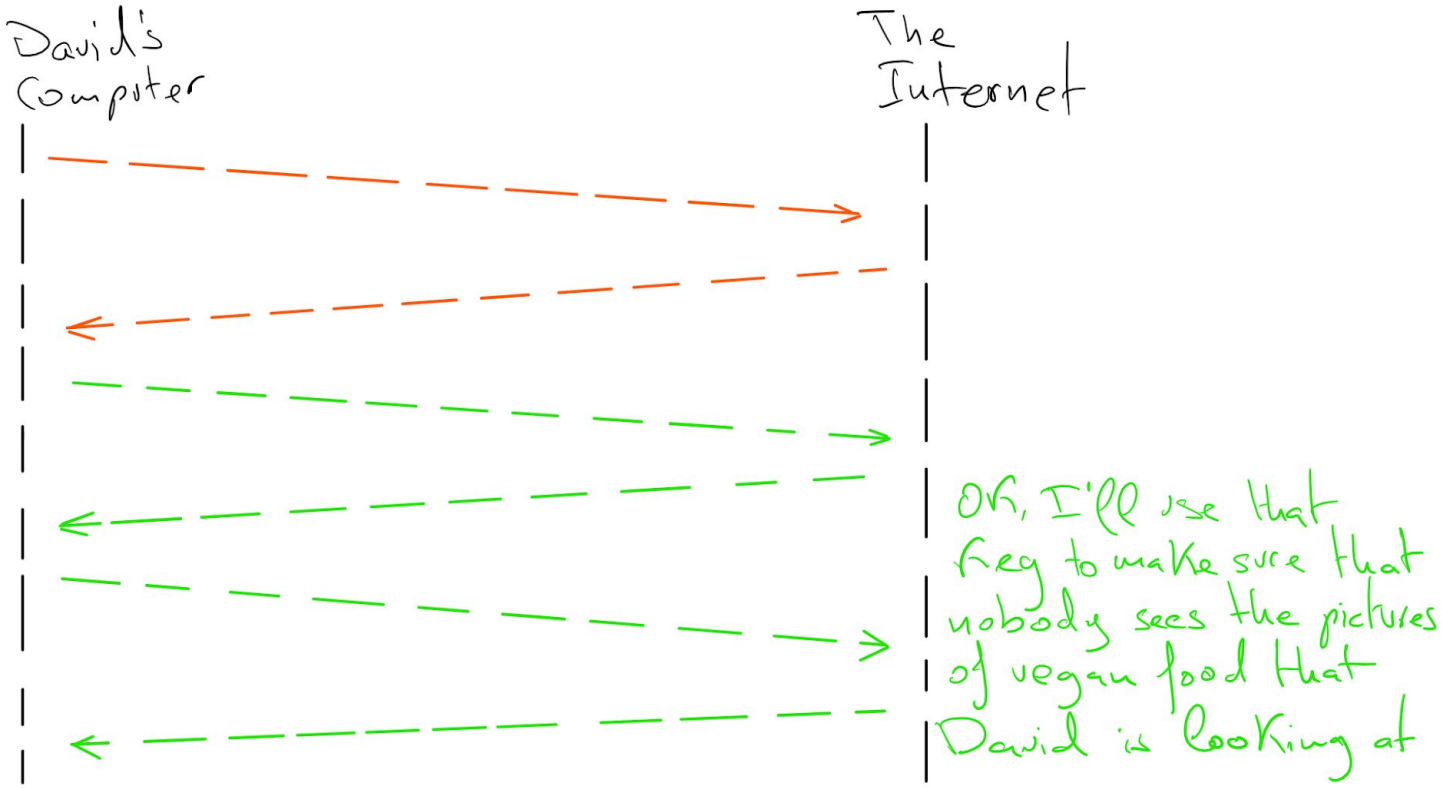



David's Computer
The Internet
OK, I can show David pictures of vegan food


David's Computer
The Internet
David doesn't want anyone to see the pictures of vegan food he's looking at


David's Computer
The Internet
OK, I can make sure nobody sees the pictures of vegan food that David is looking at


David's Computer
The Internet
Please, use this key to make sure nobody sees the pictures of vegan food that David is looking at


David's Computer
The Internet
OK, I'll use that key to make sure that nobody sees the pictures of vegan food that David is looking at

All Internet traffic is going to be encrypted sooner rather than later

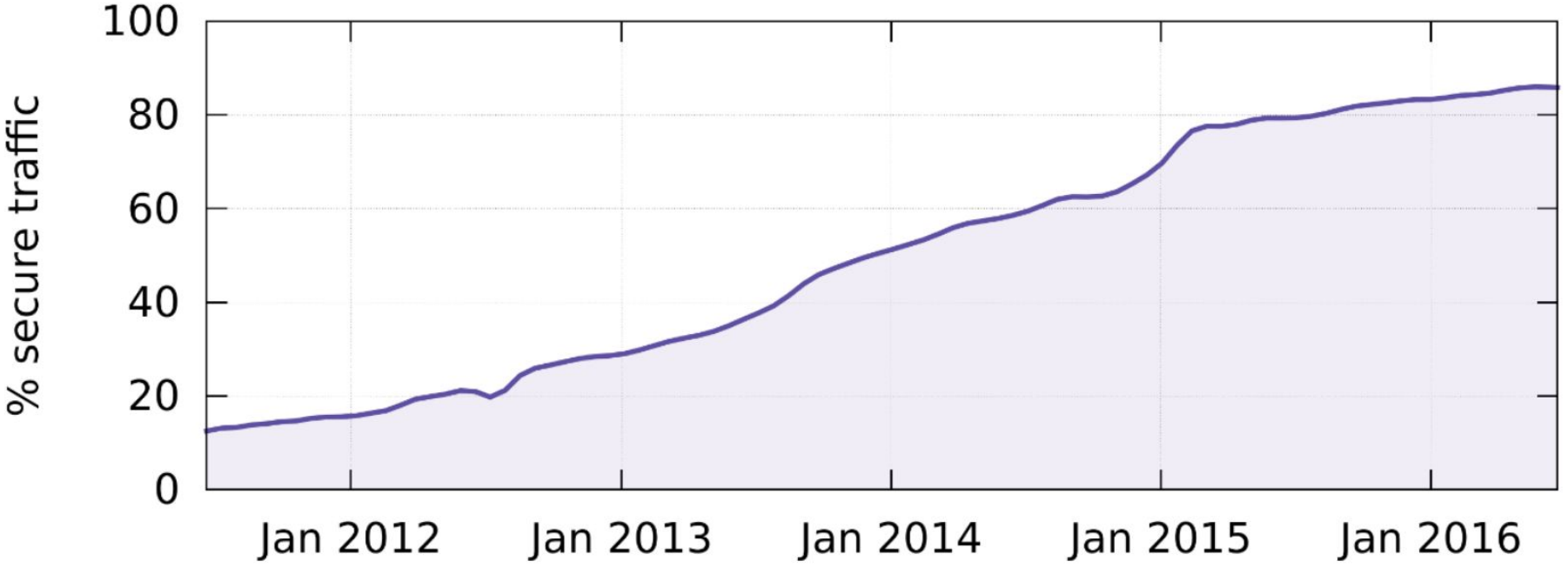
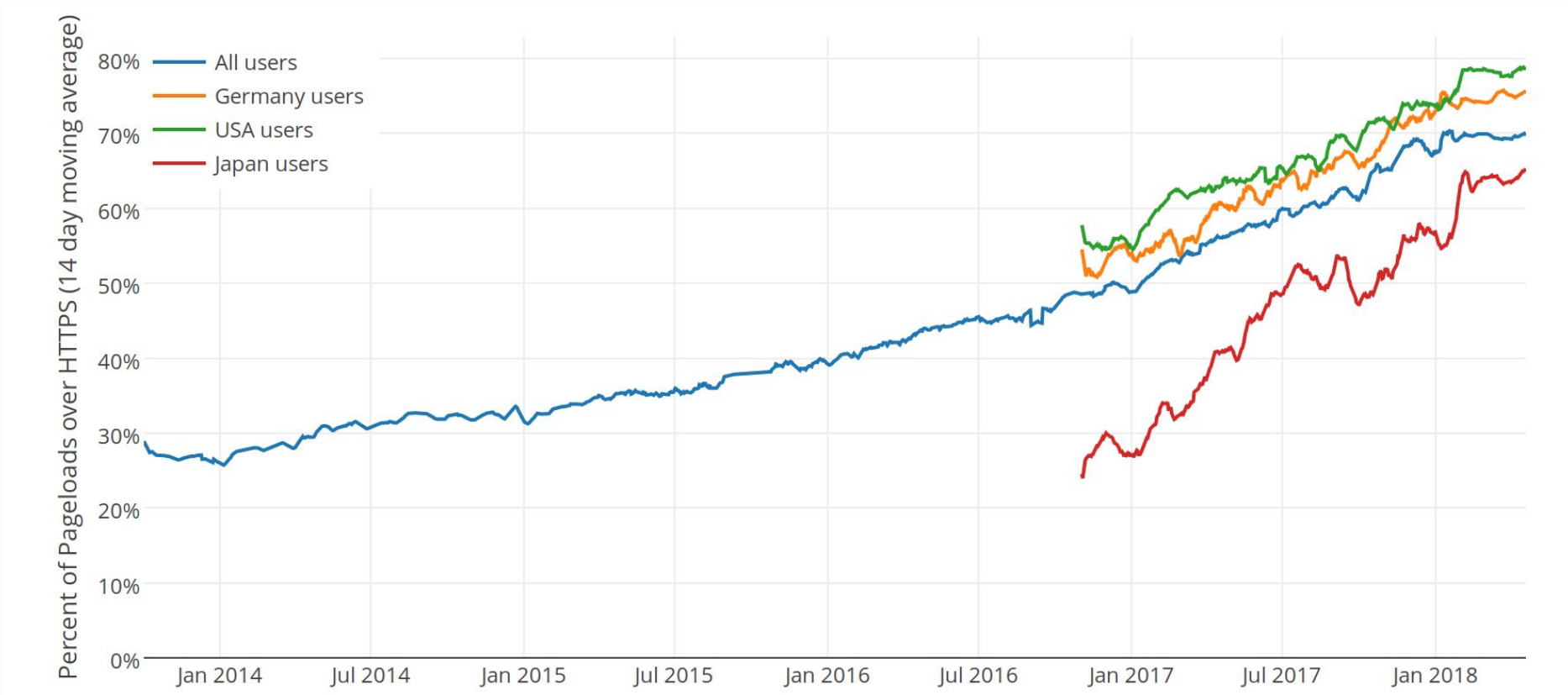


Figure 3: Increase in secure web traffic to Google's front-end servers.

Percentage of Web Pages Loaded by Firefox Using HTTPS

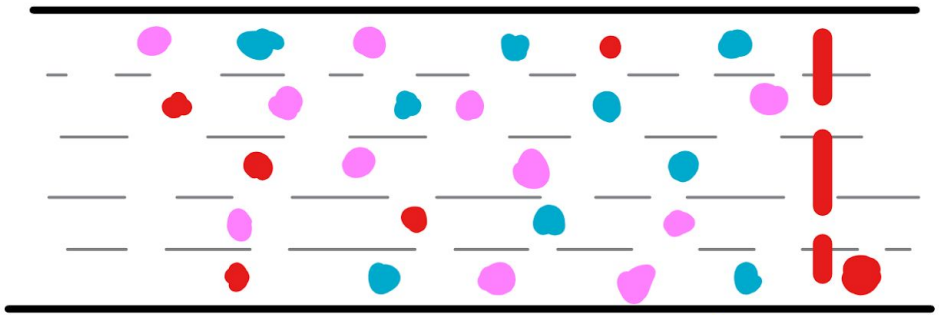
(14-day moving average, source: Firefox Telemetry)



Head-of-line blocking

Sooooo bossy

David's computer



The Internet

TCP Connection

TCP is virtually impossible to change

Design and implementation

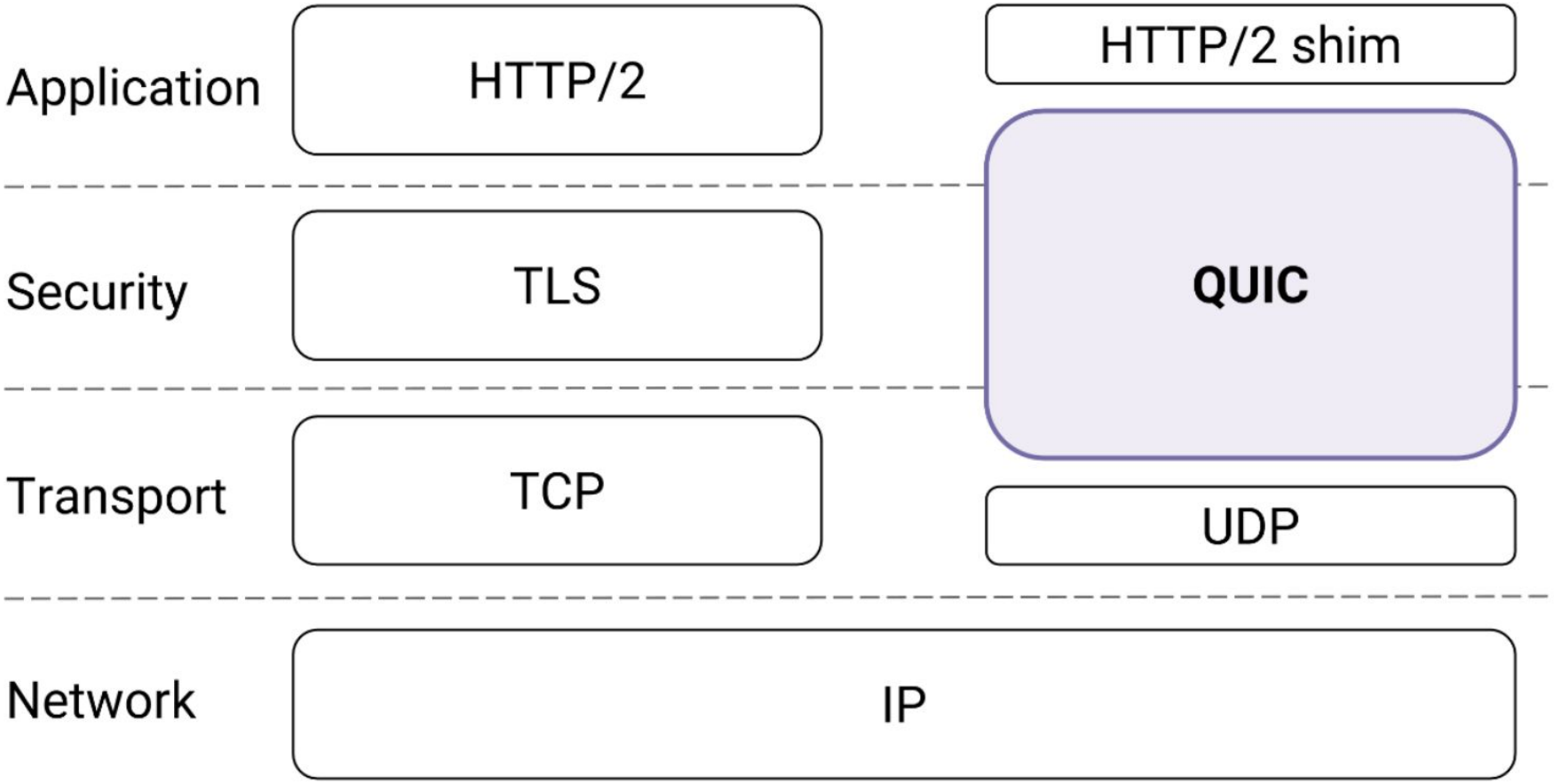


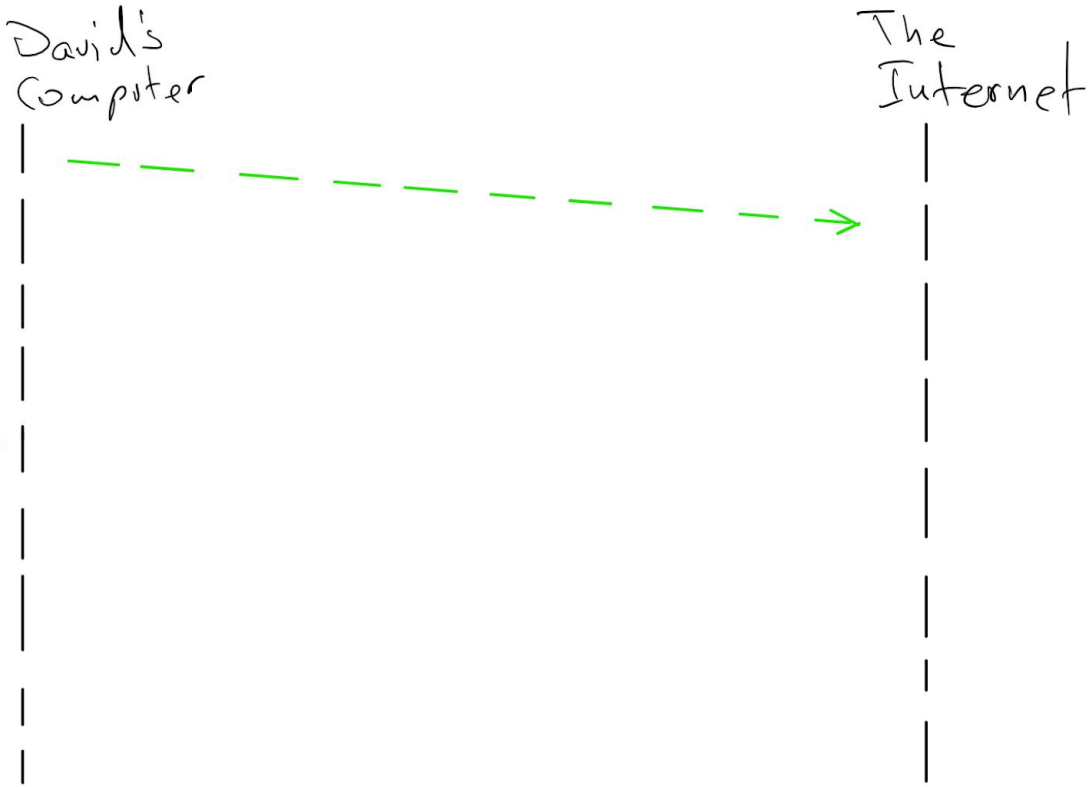
Figure 1: QUIC in the traditional HTTPS stack.

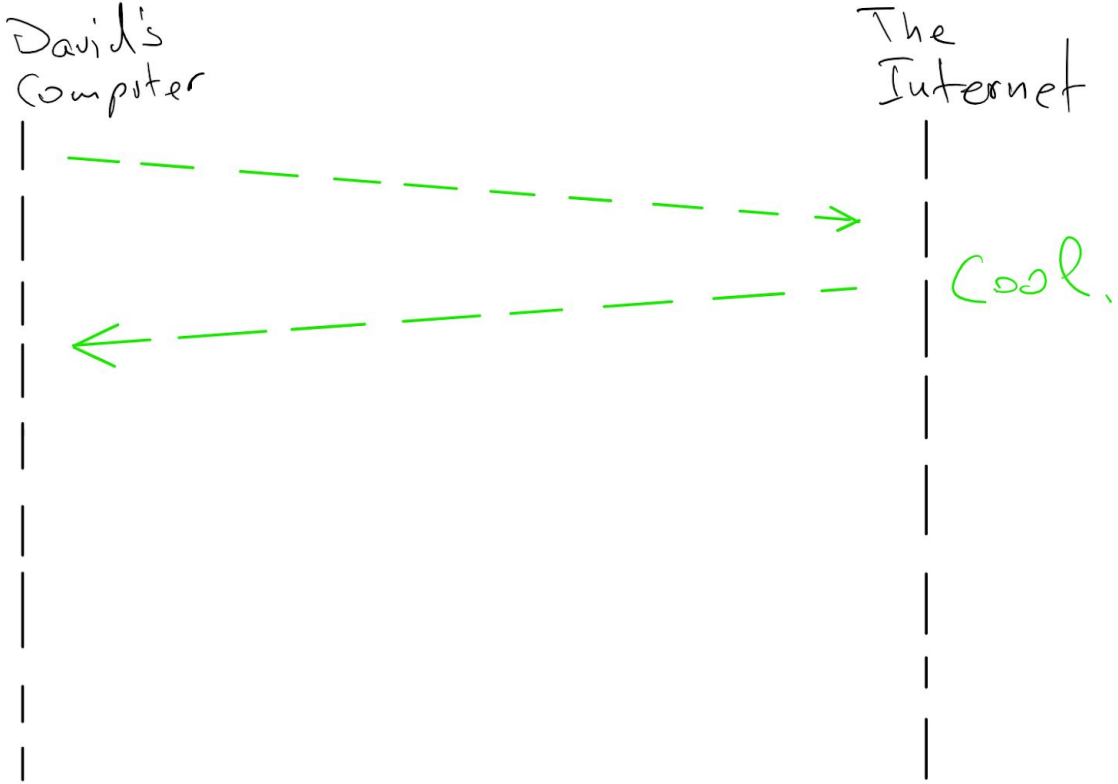
Connection establishment

0 Round-Trip Time

* At best

David wants to see pictures of vegan food. He doesn't want anyone to see the pictures of vegan food he's looking at. He's been here before, use this key to ensure that nobody sees the pictures of vegan food that he's looking at.




David's Computer
The Internet
Cool.

3.1 Connection Establishment

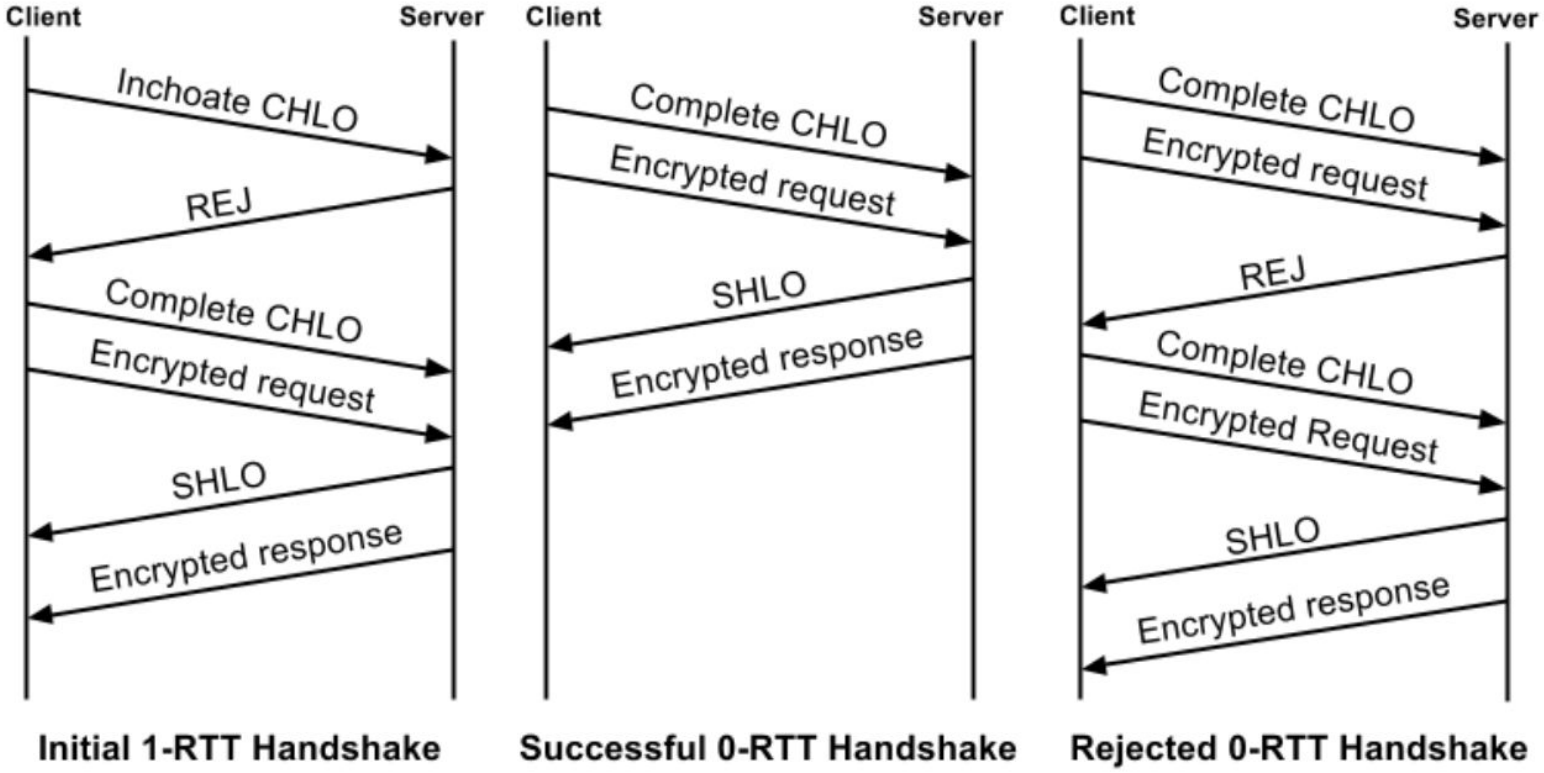


Figure 4: Timeline of QUIC's initial 1-RTT handshake, a subsequent successful 0-RTT handshake, and a failed 0-RTT handshake.

Stream multiplexing

"A lost UDP packet only impacts streams whose data it carries"

Authentication & Encryption

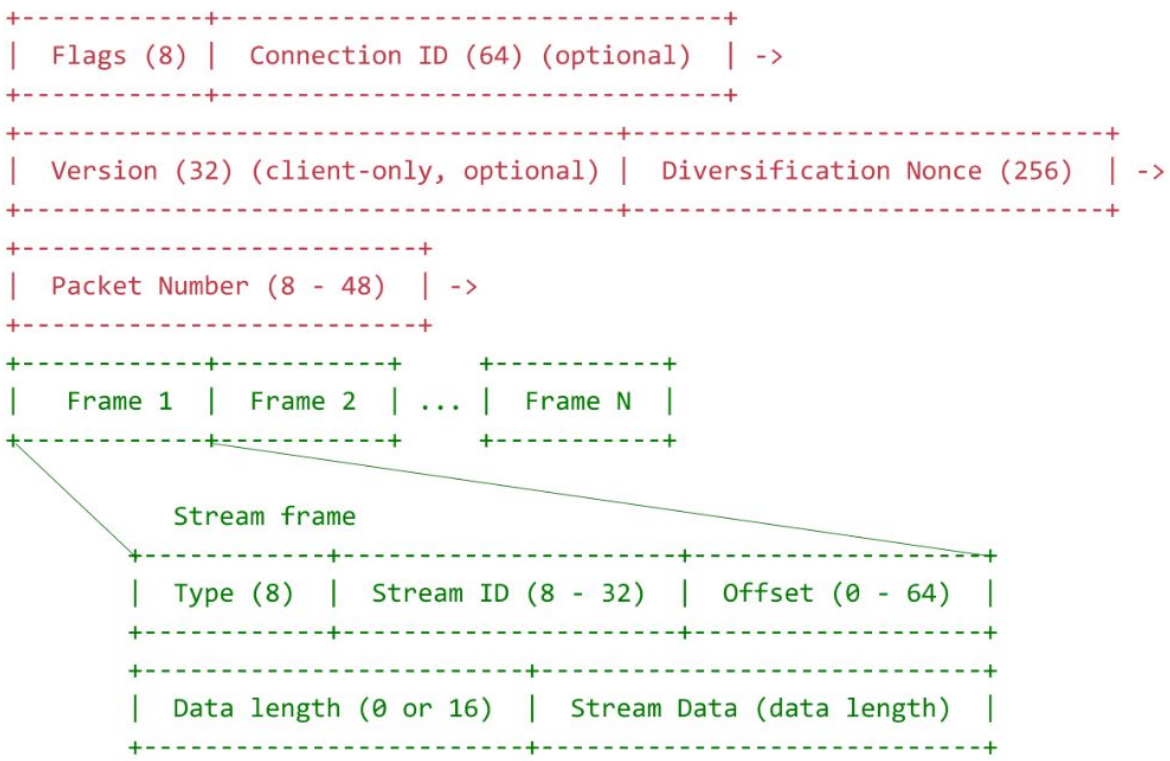


Figure 5: Structure of a QUIC packet, as of version 35 of Google's QUIC implementation. Red is the authenticated but unencrypted public header, green indicates the encrypted body. This packet structure is evolving as QUIC gets standardized at the IETF [2].

Loss recovery

"Each packet carries a new monotonic number"

"Acknowledgments encode the delay since receiving the packet"

Congestion control

Roll your own

NAT rebinding & Connection migration

"QUIC connections are identified by a 64-bit connection ID"

Internet-Scale deployment

"QUIC support was added to Chrome in June 2013"

"Allowed to new features to be A/B tested"

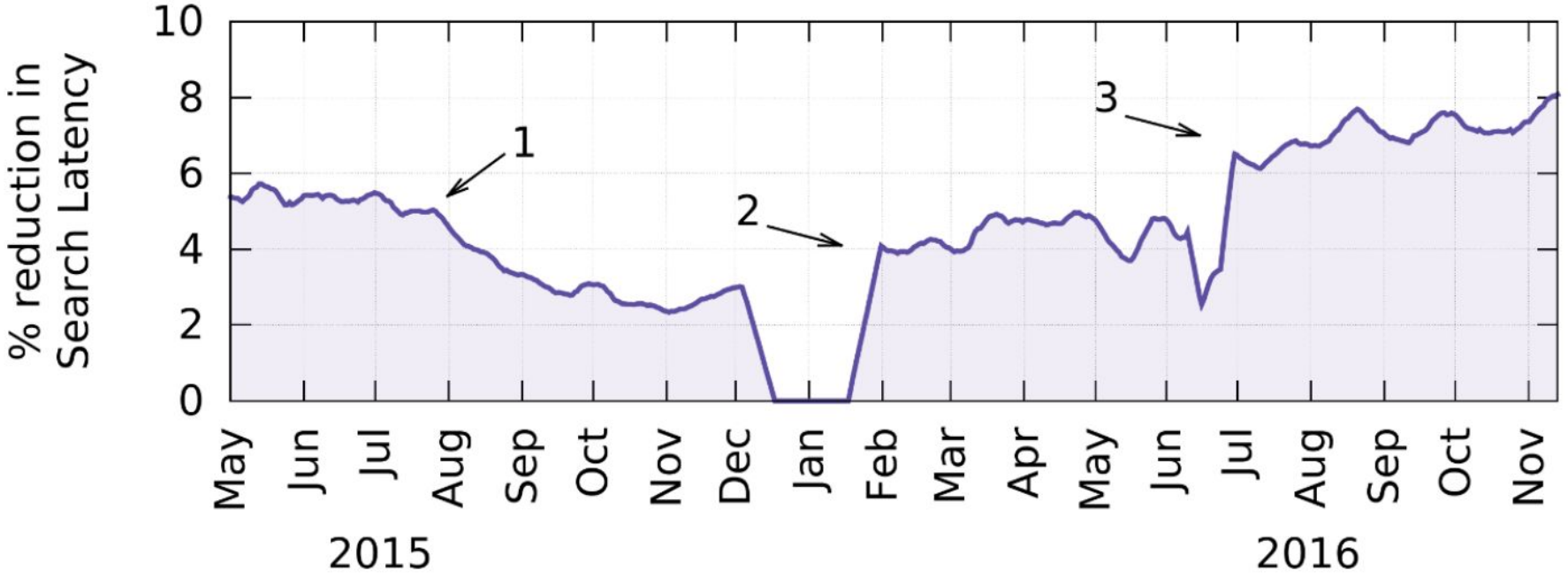


Figure 6: Search Latency reduction for users in the QUIC experiment over an 18-month period. Numbered events are described in Section 5.2.

Performance

Transport and application metrics

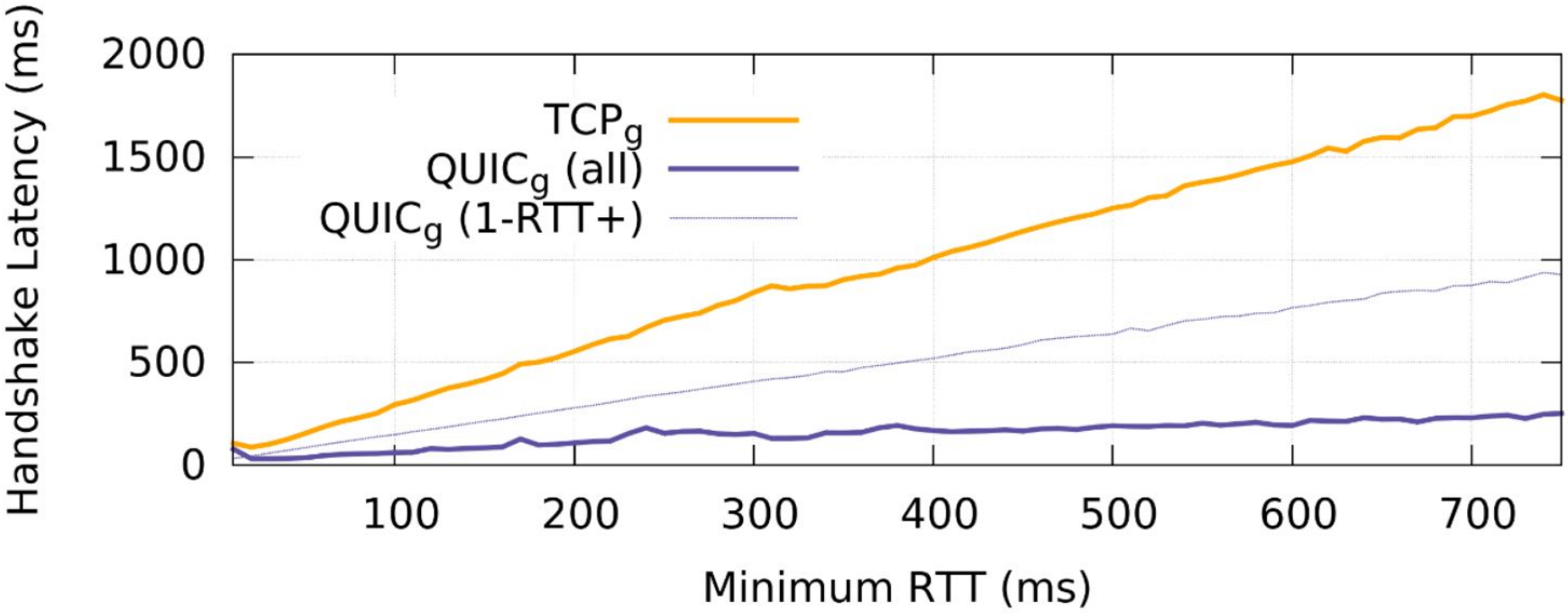


Figure 7: Comparison of handshake latency for $QUIC_g$ and TCP_g versus the minimum RTT of the connection. Solid lines indicate the mean handshake latency for all connections, including 0-RTT connections.

	% latency reduction by percentile						
		Lower latency				Higher latency	
	Mean	1%	5%	10%	50%	90%	95%	99%
Search								
Desktop	8.0	0.4	1.3	1.4	1.5	5.8	10.3	16.7
Mobile	3.6	-0.6	-0.3	0.3	0.5	4.5	8.8	14.3
Video								
Desktop	8.0	1.2	3.1	3.3	4.6	8.4	9.0	10.6
Mobile	5.3	0.0	0.6	0.5	1.2	4.4	5.8	7.5

Table 1: Percent reduction in global Search and Video Latency for users in $QUIC_g$, at the mean and at specific percentiles. A 16.7% reduction at the 99th percentile indicates that the 99th percentile latency for $QUIC_g$ is 16.7% lower than the 99th percentile latency for TCP_g.

		% rebuffer rate reduction by percentile				
		Fewer rebuffers			More rebuffers	
	Mean	< 93%	93%	94 %	95%	99%
Desktop	18.0	*	100.0	70.4	60.0	18.5
Mobile	15.3	*	*	100.0	52.7	8.7

Table 2: Percent reduction in global Video Rebuffer Rate for users in $QUIC_g$ at the mean and at specific percentiles. An 18.5% reduction at the 99th percentile indicates that the 99th percentile rebuffer rate for $QUIC_g$ is 18.5% lower than the 99th percentile rate for TCP_g. An * indicates that neither $QUIC_g$ nor TCP_g have rebuffers at that percentile.

Server CPU utilization

"QUIC's server CPU utilization was about 3.5 times higher"

Lessons learned

Packet size considerations

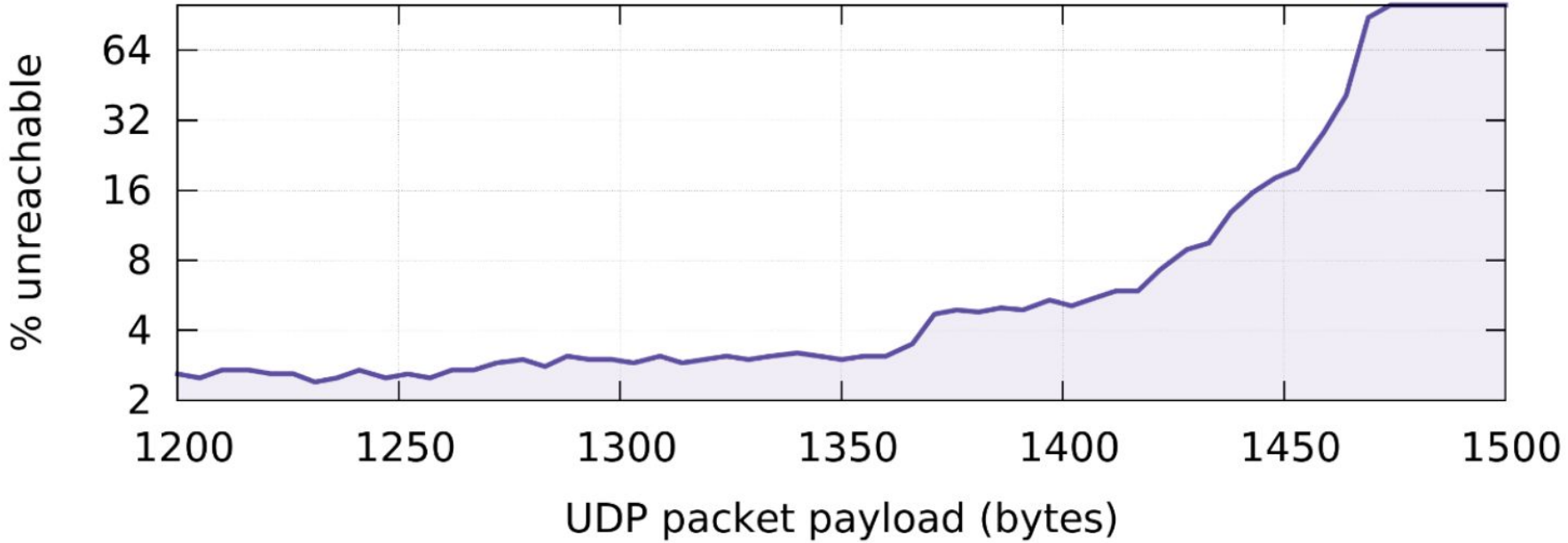
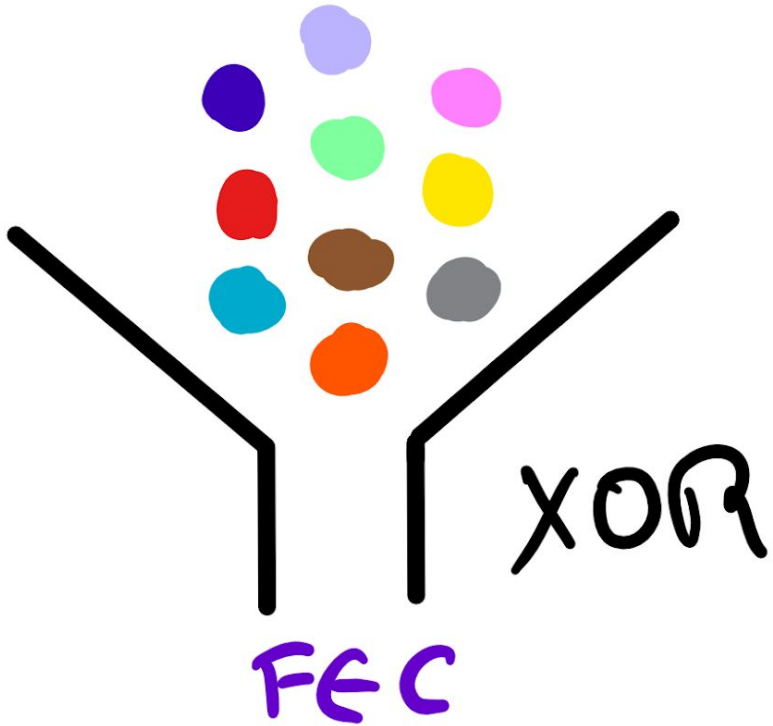


Figure 12: Unreachability with various UDP payload sizes. Data collected over 28 days in January 2014.

UDP Blockage and Throttling

"QUIC is successfully for 95.3% of client connections"

Forward Error Correction


XOR
FEC

"The benefits are limited to under 30% of loss episodes"

User-Space Development

"Led us to uncover a decade old Cubic bug"

Experiences with Middleboxes

RTFS*

* Read The F******** Specification

Conclusion

How Secure and Quick is QUIC

R. Lychev, S. Jero, A. Boldyreva, and C. Nita-Rotaru. 2015.

"Does NOT satisfy the traditional notion of forward secrecy"

Standard WIP

https://quicwg.github.io

Not widely implemented outside Google yet

Chromium, Apache Traffic Server, Caddy

Thank you for listening!

David Calavera

@calavera