



QUIC Protocol Overview for Enterprises

With Packet Analysis Examples

Bill.Alderson@SecurityInstitute.com

Course PDF <https://CE.SecurityInstitute.com/QUIC>

ISF  **2024**
Information Security Forum
for Texas Government



Packetman007



**QUIC
REPLACES
TCP-SSL-HTTP
READY ?**

INNOTECH

 **ISSA**
Information Systems Security Association
International

 **ISF
2024**

 **LIVE
STREAM**

CAUTION
QUIC
PROTOCOL

SharkFest'23 US
Wireshark Developer and User Conference • San Diego, CA • June 10-15

QUIC Transport Protocol

IETF rfc-9000 & related rfcs





Windows Server 2025

Overview of what's New

Windows Server Summit 2024 | March 26-28 2024



QUIC Protocol Is Rapidly Replacing TCP!

Every Major Client Web Browser
Chrome – Edge – Safari – Mozilla

Most Social Media Apps 90%+
Google – Facebook

Apple – Microsoft Server 2022+
CDN' s Cloudflare - Fastly

SharkFest 2023

How QUIC Works And What Are The Security Concerns?

Why QUIC is Faster? Show You How To Perform “Packet Based Analysis”

Distillation of 1,000+ IETF rfc-9000 and Related Standards Pages

Why Does It Lower Firewall Sessions 20 to 1 vs TCP?

How to Implement QUIC on Your App or Site - Fast!



Cloudflare Radar



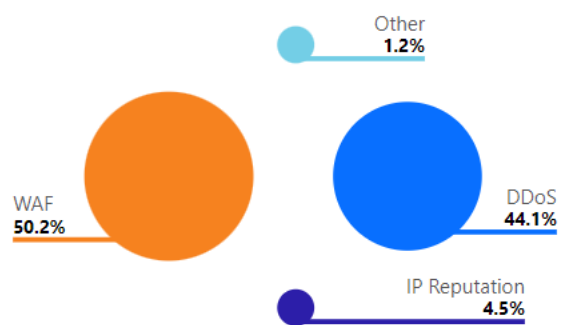
QUIC Protocol Adoption

Security & Attacks

Insight into network and application layer attack traffic

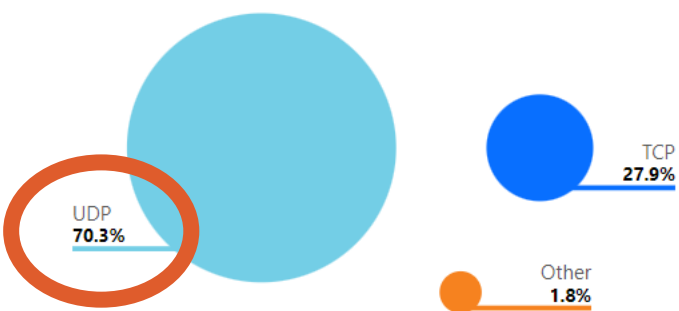
Layer 7 Attacks

Top Mitigation Techniques



Layer 3/4 Attacks

DDoS Attack Type

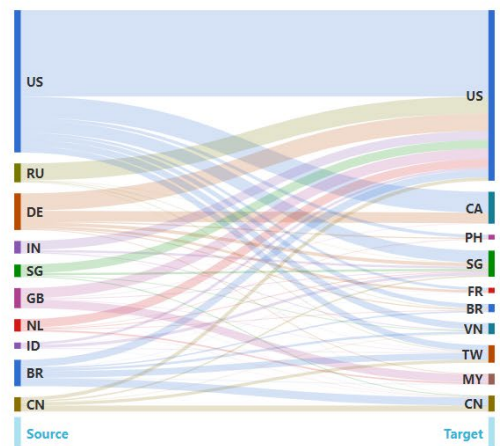


Application layer attack activity

Top 10 attacks by target or source location



Sort order: Source Target

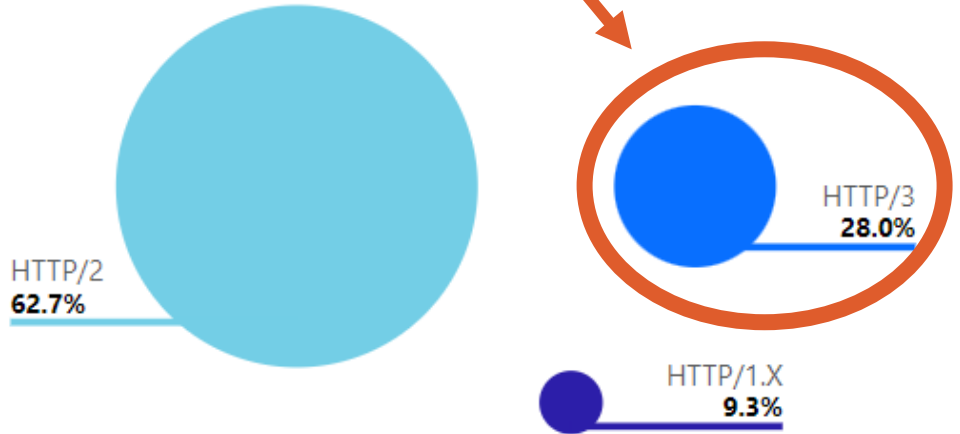
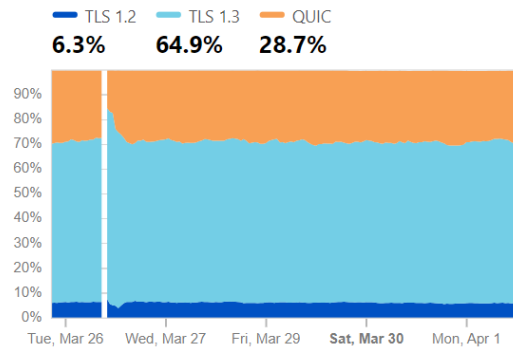


HTTP versions

HTTP/1.x vs. HTTP/2 vs. HTTP/3

TLS 1.2 vs. TLS 1.3 vs. QUIC

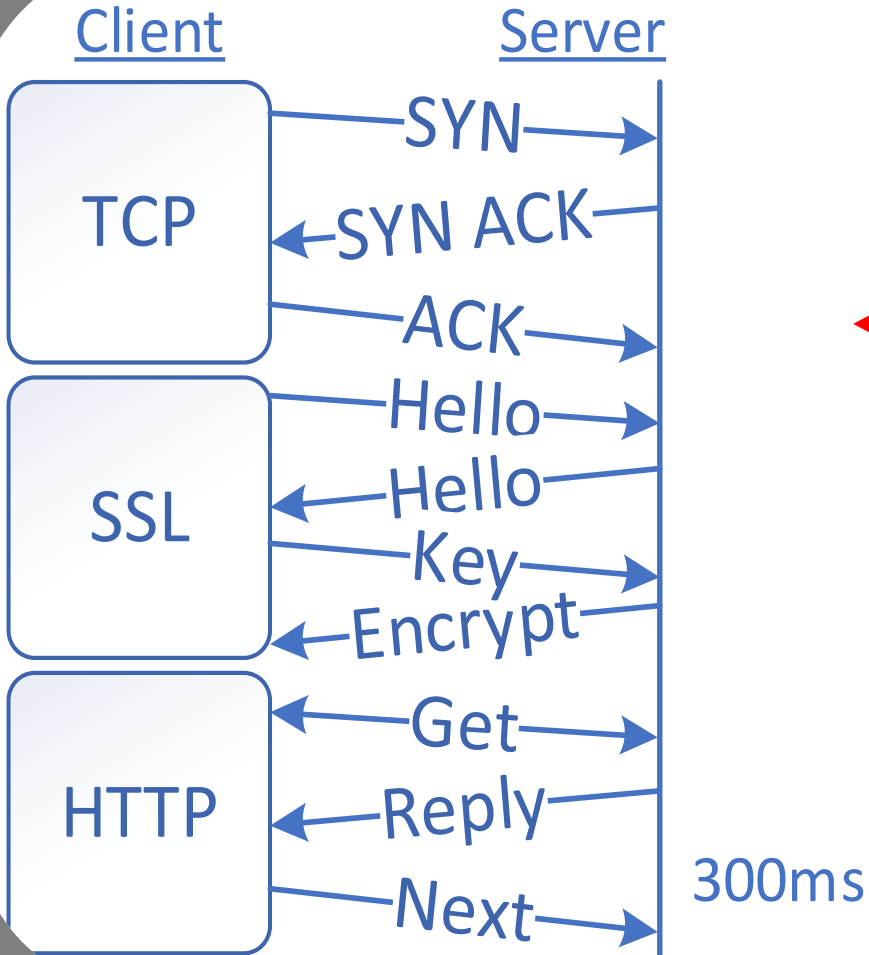
Distribution of secure traffic by protocol



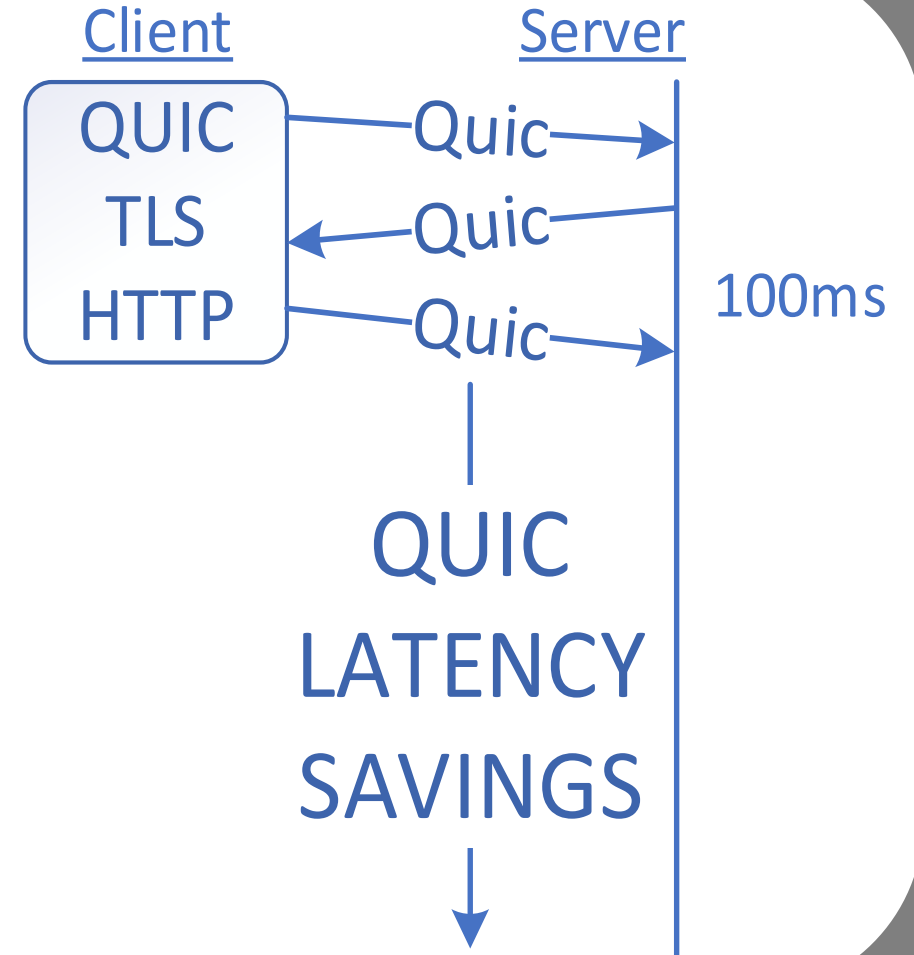
Current Internet 1x

QUIC 3x Faster

Basics of QUIC Protocol



← 300ms 100ms →
TCP Vs. QUIC
3x
Faster





Firewall Vendors Tell Enterprises “DENY” QUIC

Firewall Vendors that recommend Blocking QUIC UDP Port 443 in Enterprise Networks

Cisco
PaloAlto Networks
CheckPoint
Fortinet

Each vendor provides specific instructions to block QUIC

Web Performance
at a Price...

3x
FASTER

SECURITY
THREATS



An “Attractive Nuisance”
Until Enterprise Firewall
Improvements Secure Its Use

GotQuic?

- 2012 accidentally discovered... by Jim Roskind at Google, now AWS
- Google, YouTube, Gmail, Facebook, Microsoft Uber and Cloudflare already use QUIC!
- Distant or rural users receive the biggest performance gain.

QUIC Initial Connect – some headers exposed

gquic.tag.sni == "r1---sn-q4f7sn7r.googlevideo.com"

No.	Time	Source	Destination	Protocol	Length	QLen	Q PNum	CumBytes	Info	Server Name Indication	QVer	Proof demand	Idle connection state	QMax in	QEstInitRTT	Initial session/connr	Initial stream fk	QPadLen	C
1	0.000000	172.20.1.29	173.194.141.247	QUIC	1392	1350	1	1392	Client Hello	r1---sn-q4f7sn7r.googlevideo.com	Q050	X509	30 (0x0000001e)	100	23149	15728640	6291456	800	
5	0.029508	172.20.1.29	173.194.141.247	QUIC	1392	1350	4	2784	Client Hello	r1---sn-q4f7sn7r.googlevideo.com	Q050	X509	30 (0x0000001e)	100	23149	15728640	6291456	731	

Tag/value: 72312d2d2d736e2d71346637736e37722e676f6f676c65766964656f2e636f6d
 Server Name Indication: r1---sn-q4f7sn7r.googlevideo.com
 Tag/value: STK (Source Address Token) (l=54)
 Tag Type: STK (Source Address Token)
 Tag offset end: 86
 [Tag length: 54]
 Tag/value: c9e211c10f97814d29305cbda5453ff1d23cece426489de66b374ba91c024fca14ae8a4e...

00b0 42 01 00 00 53 46 43 57 46 01 00 00 72 31 2d 2d B...SFCW F...r1--
 00c0 2d 73 6e 2d 71 34 66 37 73 6e 37 72 2e 67 6f 6f -sn-q4f7 sn7r.goo
 00d0 67 6c 65 76 69 64 65 6f 2e 63 6f 6d c9 e2 11 c1 glevideo .com...
 00e0 0f 97 81 4d 29 30 5c bd a5 45 3f f1 d2 3c ec e4 ...M)\...E?...<..
 00f0 26 48 9d e6 6b 37 4b a9 1c 02 4f ca 14 ae 8a 4e &H...k7K...D...N
 0100 e6 fe 88 6e 7c af cc 64 1f fc e0 00 ec 57 38 63 ...n]...d...W8c
 0110 c2 4c 51 30 35 30 01 e8 81 60 92 92 1a e8 7e ed [Q050]...
 0120 80 86 a2 15 82 91 5f b9 5a 5d e9 b9 a2 0d a3 78Z]...x
 0130 8d 3d 48 fa 3b 11 66 0d 37 34 e4 85 b9 20 7d 6f --H;...F...74...}o
 0140 a8 43 a3 0d 00 c6 41 45 53 47 43 68 72 6f 6d 65 C...AE SgChrome
 0150 2f 38 36 2e 30 2e 34 32 34 30 2e 31 39 38 20 57 /86.0.42 40.198 W
 0160 69 6e 64 6f 77 73 20 4e 54 20 31 30 2e 30 3b 20 indows N T 10.0;
 0170 57 69 6e 36 34 3b 20 78 36 34 ab c5 6b ce 0f a4 Win64; x 64-k...
 0180 ..

There are a few things a firewall can validate in the initial QUIC connection

There also a few things the firewall can harvest from connection requests for reference in addition to the IP addresses only a vendor can implement.

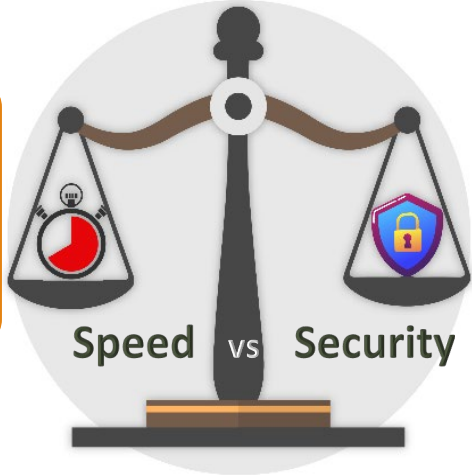
A powerful firewall can build custom filters with the help of Wireshark traces

Enterprise firewall improvements need to identify well formed QUIC packet headers

Improve and upgrade SMB - Home router-firewalls to identify QUIC

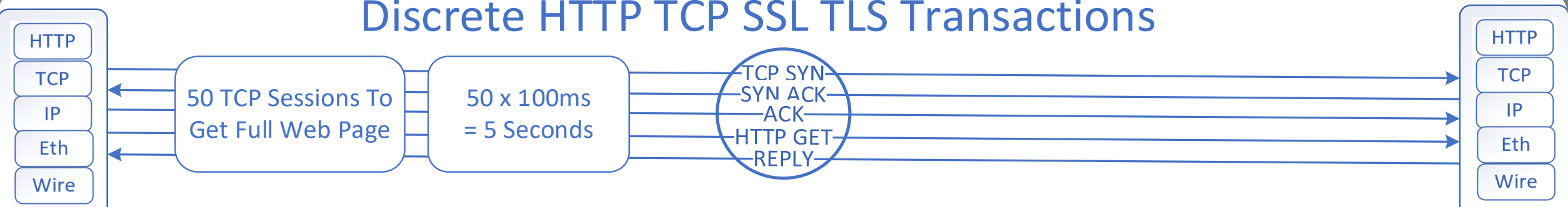
Technically it is possible upgrades can be accomplished at reasonable costs

New firewalls worldwide might cost \$1 Trillion and take 10+ years

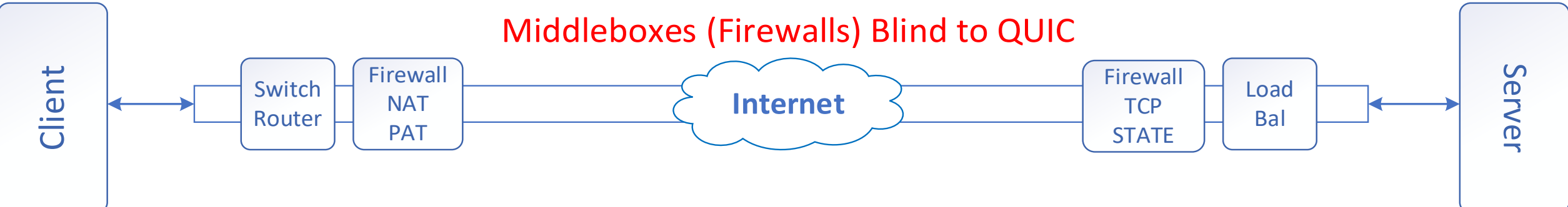


Why Middleboxes (Firewalls) Blind to QUIC?

Discrete HTTP TCP SSL TLS Transactions

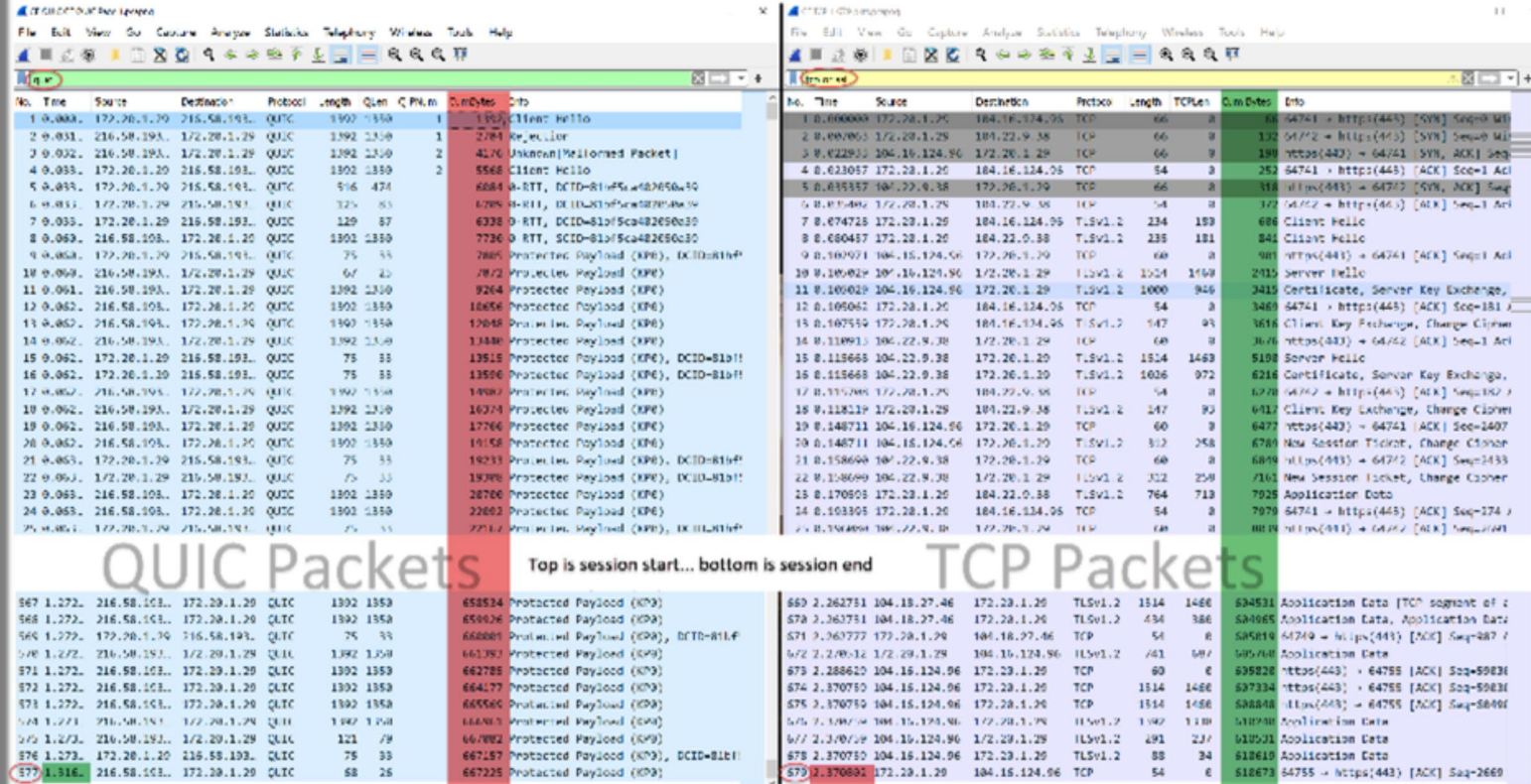


Middleboxes (Firewalls) Blind to QUIC



QUIC Combined 0-RTT HTTP TLS FEC w/Fast Mobile Reconnects

QUIC vs. TCP Full Page Packet by Packet Performance Analysis



**Benchmarks
Prove It
Faster!**

Full Page Load With Multiple Objects

Measurement	TCP	QUIC	QUIC RAW Advantage	Advantage QUIC HTTP/3	Difference
Sessions	13	1	12	<div style="width: 92%;"></div>	92% Less Sessions
Packets	679	577	102	<div style="width: 15%;"></div>	15% QUIC Fewer Packets
Time span, seconds	2.371	1.317	1.054	<div style="width: 44%;"></div>	44% QUIC Significantly Better Time, seconds
Average pps	286	438	152	<div style="width: 53%;"></div>	53% QUIC Better Average pps
Average packet size	899	1156	257	<div style="width: 29%;"></div>	29% QUIC Larger packet size
Bytes	610673	667225	-56552	<div style="width: -9%;"></div>	-9% TCP Fewer Bytes
Average bits/s	2060000	4053000	1993000	<div style="width: 97%;"></div>	97% QUIC Better Throughput bits/s

Benchmark Page <https://cloudflare-quic.com>

Benchmark Results

Single Object One Video Load

Measurement	TCP	QUIC	QUIC RAW Advantage	Mixed Results TCP HTTP/2	Difference
Sessions	1	1	0		0% Same Sessions
Packets	1985	2123	-138		-7% TCP Fewer Packets
Time span, seconds	40.332	40.241	0.091		0% QUIC Marginally Better Time
Average pps	49	53	4		7% QUIC Better Average pps
Average packet size	1089	1192	103		9% QUIC Larger packet size
Bytes	2161738	2530541	-368803		-17% TCP Fewer Bytes
Average bits/s	428000	503000	75000		18% QUIC Better Throughput bits/s

Full Page Load With Multiple Objects

Measurement	TCP	QUIC	QUIC RAW Advantage	Advantage QUIC HTTP/3	Difference
Sessions	13	1	12		92% Less Sessions
Packets	679	577	102		15% QUIC Fewer Packets
Time span, seconds	2.371	1.317	1.054		44% QUIC Significantly Better Time, seconds
Average pps	286	438	152		53% QUIC Better Average pps
Average packet size	899	1156	257		29% QUIC Larger packet size
Bytes	610673	667225	-56552		-9% TCP Fewer Bytes
Average bits/s	2060000	4053000	1993000		97% QUIC Better Throughput bits/s

Triple+ Web Performance

at a Price...

**3x
FASTER**

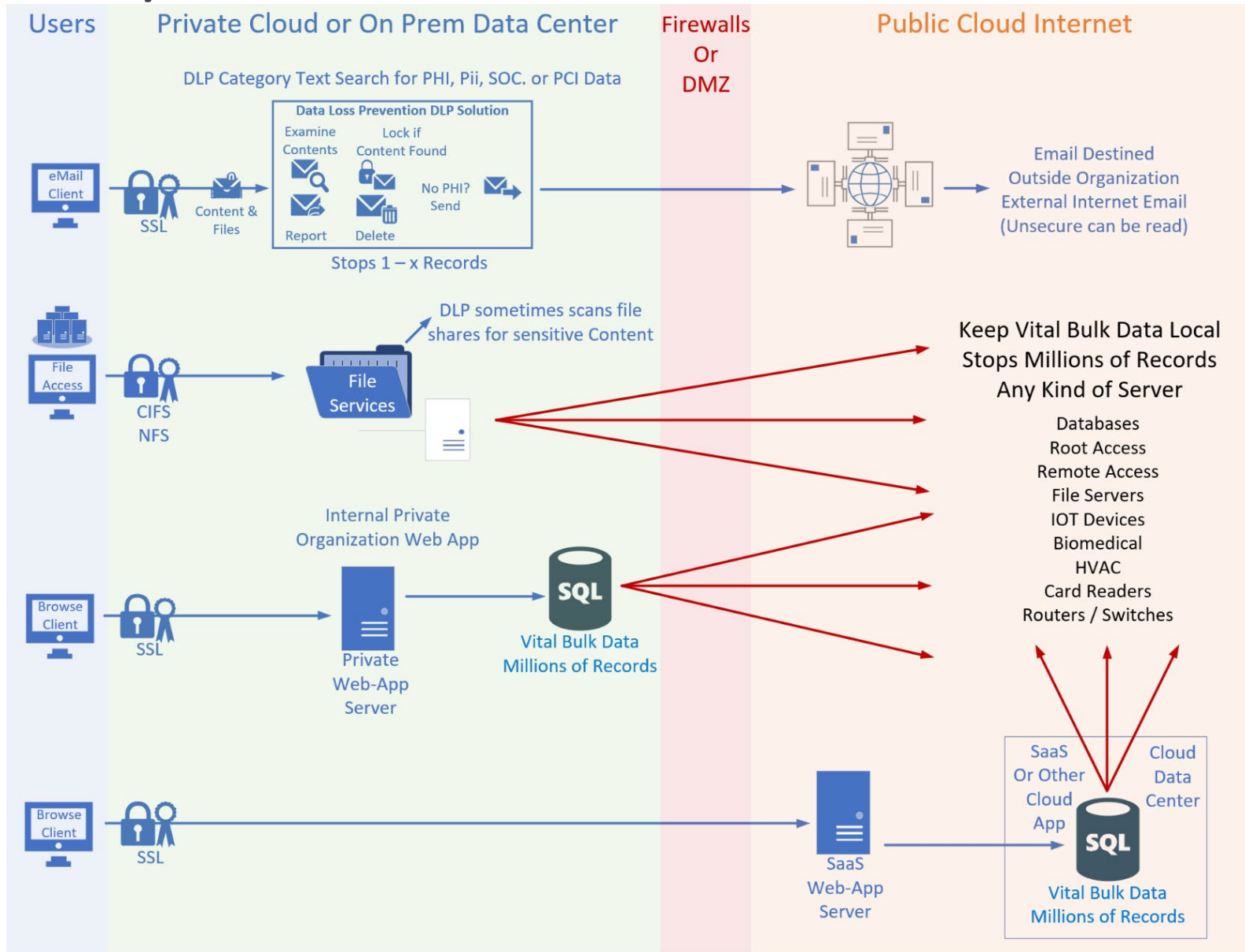

**SECURITY
THREATS**

**What Will
You Do?**



QUIC

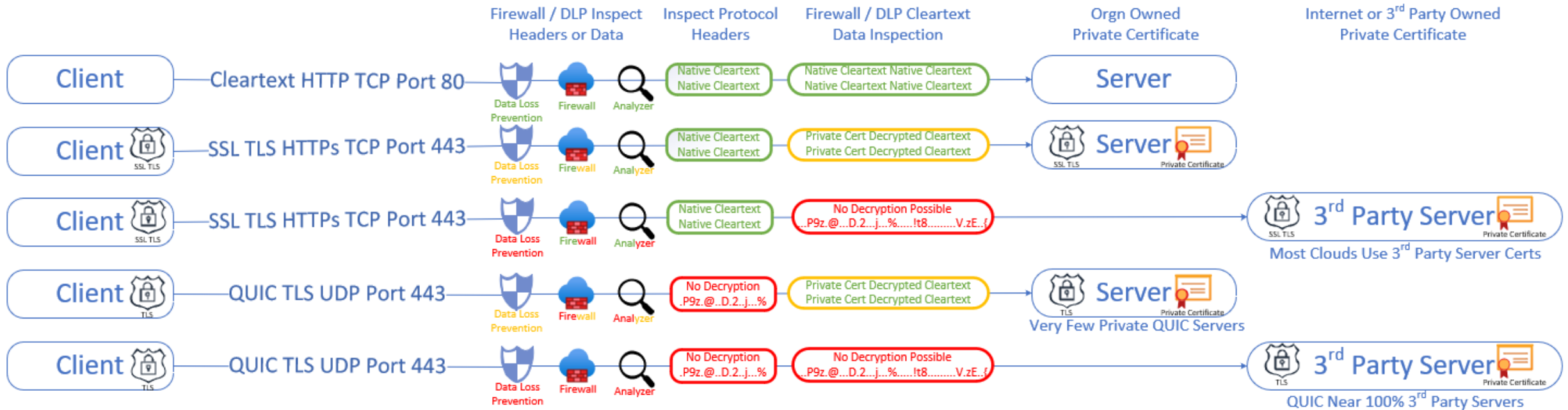
DLP May Not Check QUIC TLS 1.3!!!!!!



When does Encryption Prevent DLP Scanning?

Using 3rd party private cert prevents scanning

90+ Percent of Malware arrived Via Encrypted Traffic: WatchGuard



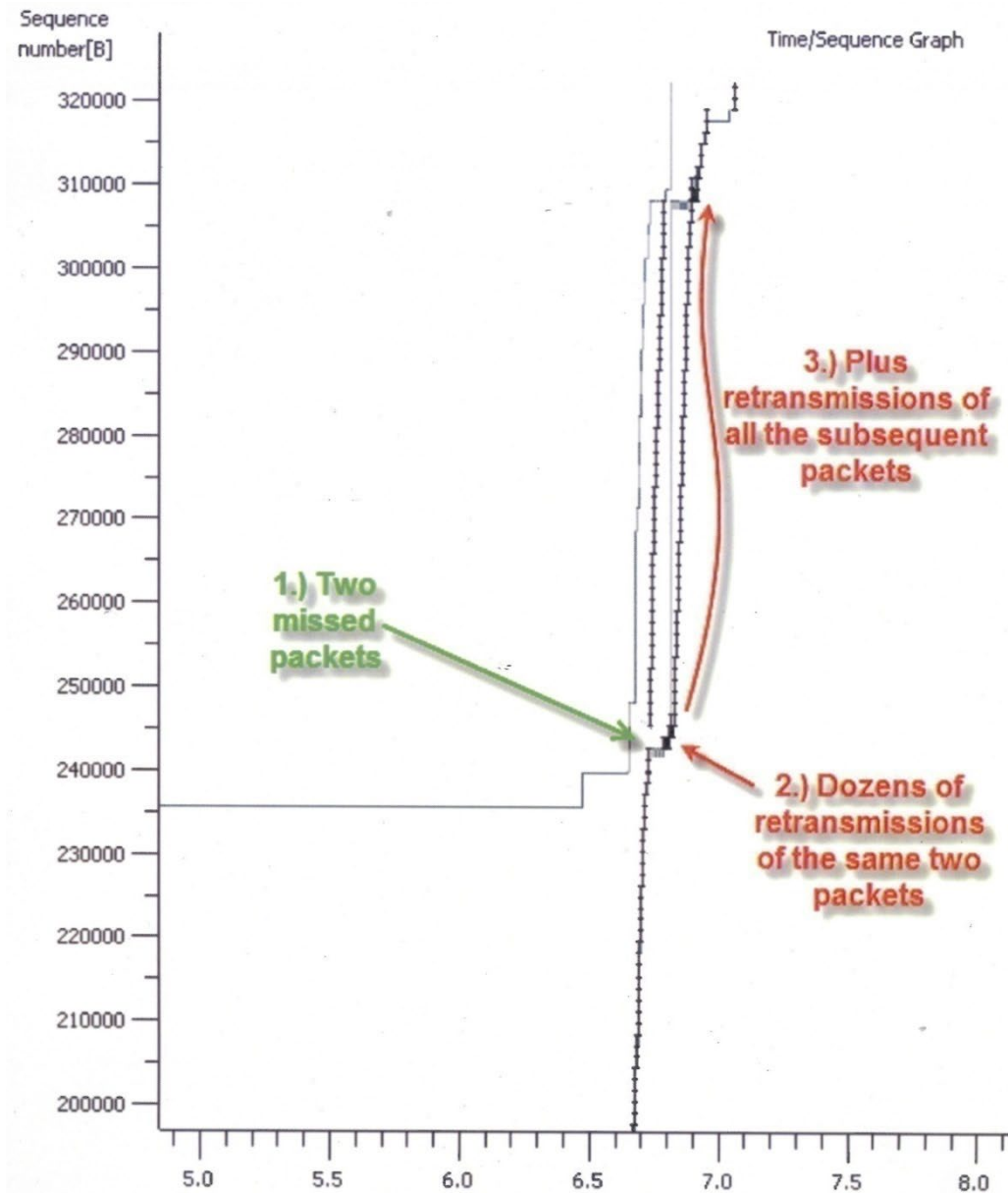
Here is a message from Vinton Cerf, known as the father of the Internet, created to encourage us on Security Zero-Day Prevention.



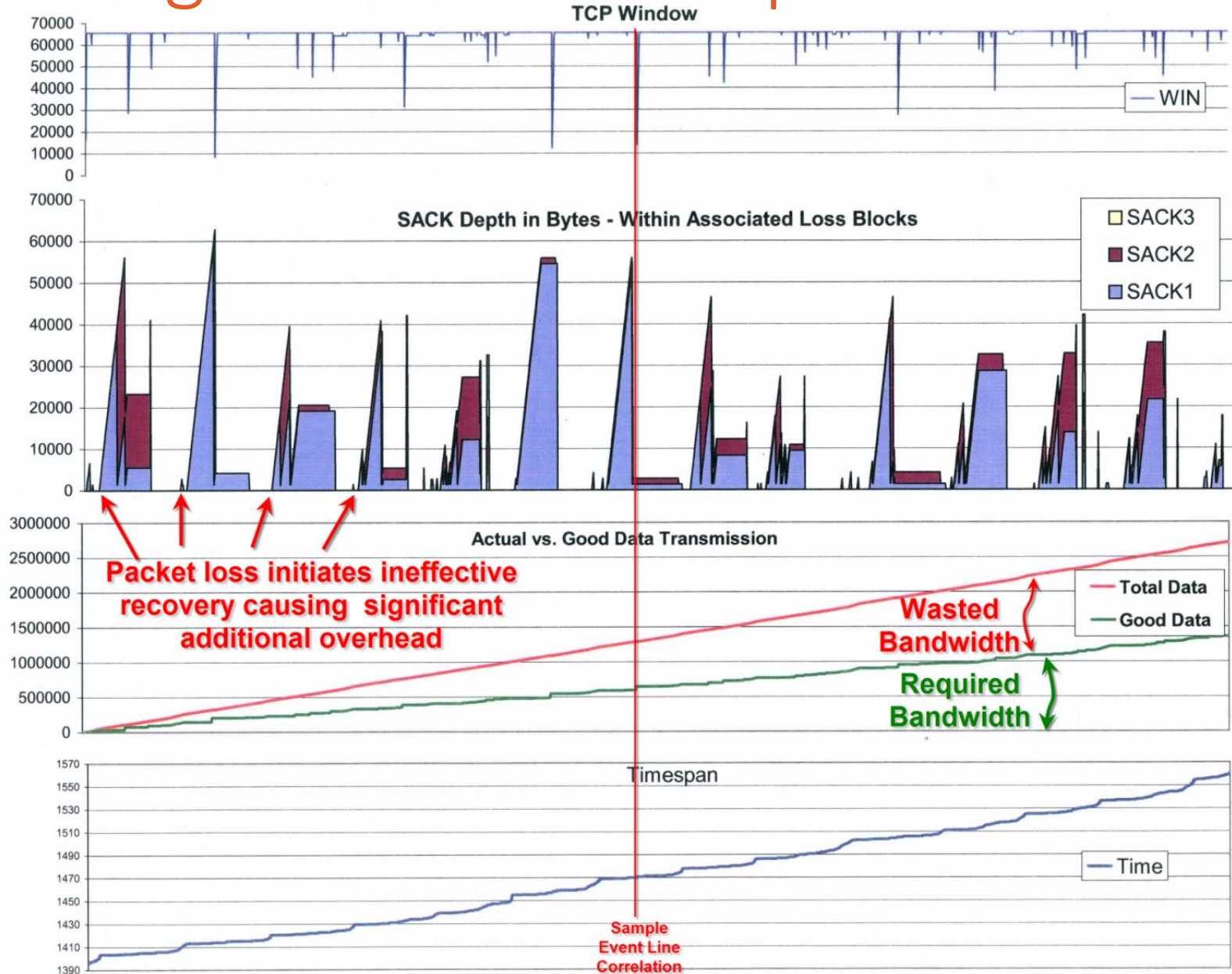
50 year old TCP's Idiosyncrasies

SOME REASONS FOR QUIC

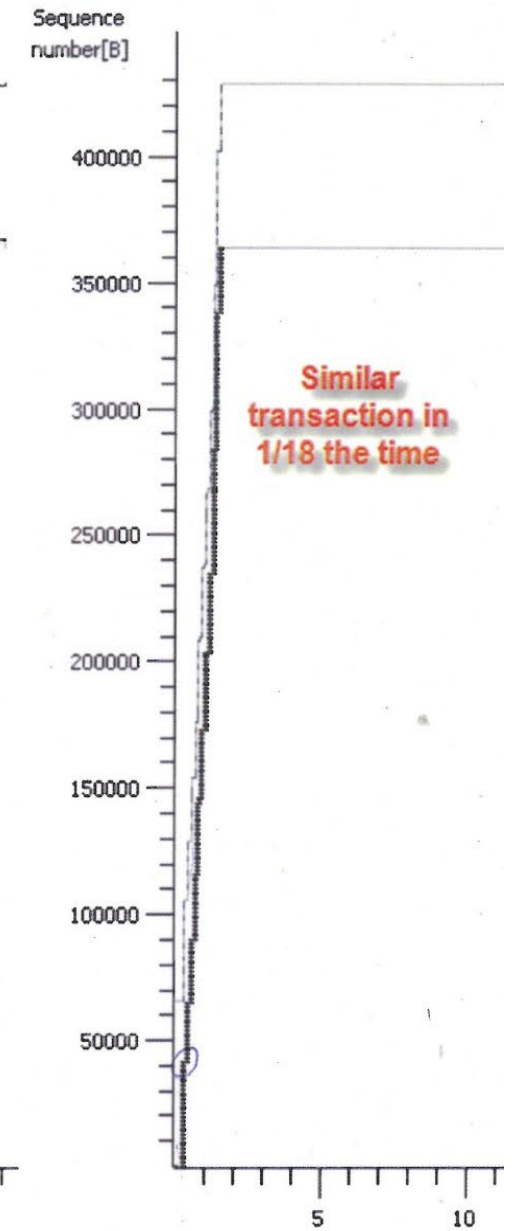
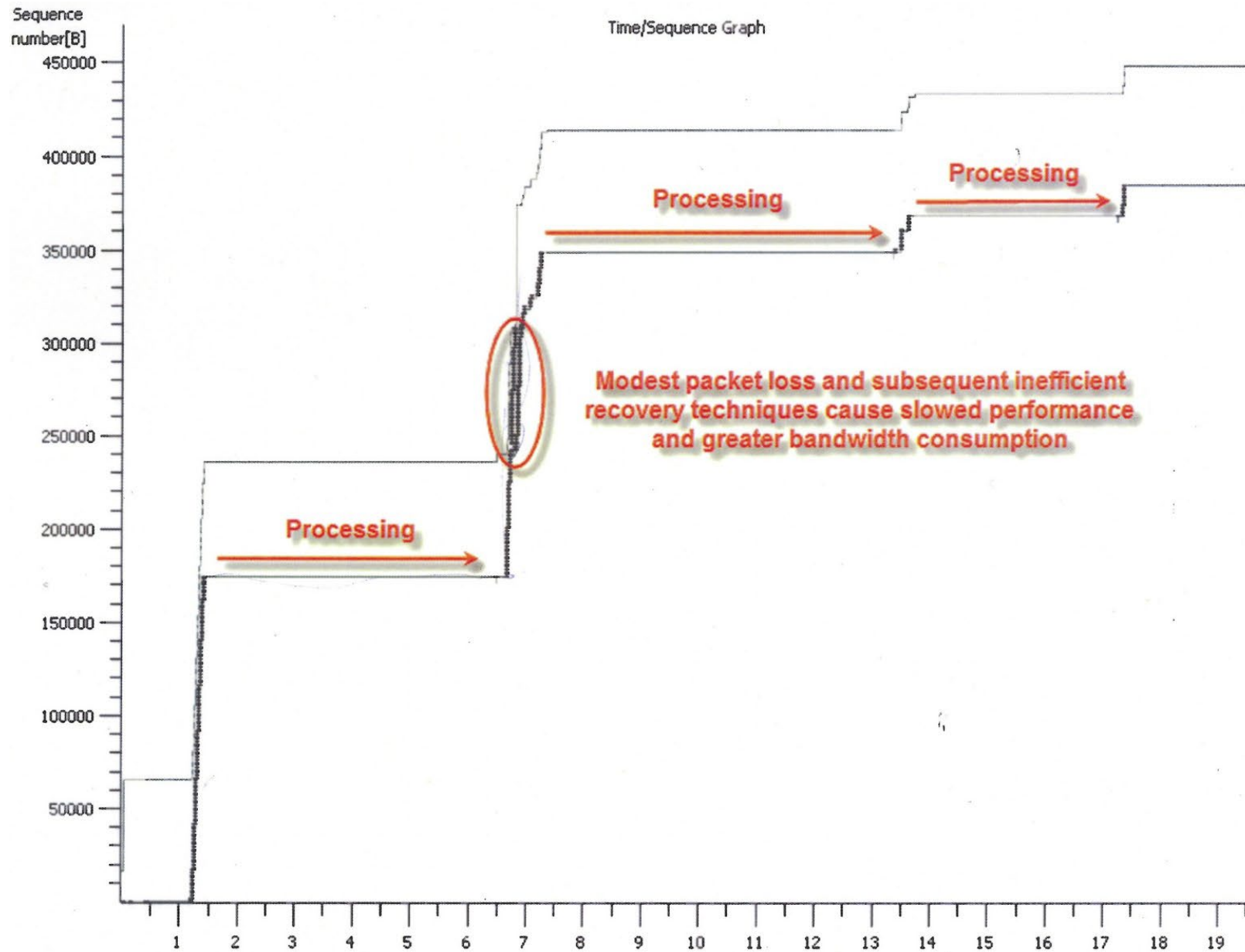
TCP Data Duplication Details



Significant Data Duplication



Data Duplication & App Processing



TCP – Packet Loss – Poor Recovery

```
41991 > https [ACK] Seq=1292614730 Ack=1606373238 Win=65535 Len=0
https > 41991 [PSH, ACK] Seq=1606381036 Ack=1292614730 Win=64316 Len=1380
https > 41991 [PSH, ACK] Seq=1606382416 Ack=1292614730 Win=64316 Len=848
41991 > https [ACK] Seq=1292614730 Ack=1606375466 Win=65535 Len=0
https > 41991 [PSH, ACK] Seq=1606383264 Ack=1292614730 Win=64316 Len=1380
https > 41991 [PSH, ACK] Seq=1606384644 Ack=1292614730 Win=64316 Len=848
41991 > https [ACK] Seq=1292614730 Ack=1606377960 Win=65535 Len=0
https > 41991 [Missed] Seq=1606385492 Ack=1292614730 Win=64316 Len=1380
https > 41991 [PSH, ACK] Seq=1606386872 Ack=1292614730 Win=64316 Len=848
41991 > https [ACK] Seq=1292614730 Ack=1606380188 Win=65535 Len=0
https > 41991 [PSH, ACK] Seq=1606387720 Ack=1292614730 Win=64316 Len=1380
https > 41991 [PSH, ACK] Seq=1606389100 Ack=1292614730 Win=64316 Len=848
41991 > https [ACK] Seq=1292614730 Ack=1606382416 Win=65535 Len=0
https > 41991 [PSH, ACK] Seq=1606389948 Ack=1292614730 Win=64316 Len=1380
https > 41991 [PSH, ACK] Seq=1606391328 Ack=1292614730 Win=64316 Len=848
41991 > https [ACK] Seq=1292614730 Ack=1606384644 Win=65535 Len=0
https > 41991 [PSH, ACK] Seq=1606392176 Ack=1292614730 Win=64316 Len=1380
https > 41991 [PSH, ACK] Seq=1606393556 Ack=1292614730 Win=64316 Len=848
41991 > https [ACK] Seq=1292614730 Ack=1606385492 Win=64687 Len=0 Last good ACK
[TCP Dup ACK 214#1] 41991 > https [ACK] Seq=1292614730 Ack=1606385492 Win=64687 Len=0 SLE=2909630688 SRE=2909631536
https > 41991 [PSH, ACK] Seq=1606394404 Ack=1292614730 Win=64316 Len=1114
[TCP Dup ACK 214#2] 41991 > https [ACK] Seq=1292614730 Ack=1606385492 Win=64687 Len=0 SLE=2909630688 SRE=2909632916
[TCP Fast Retransmission] https > 41991 [PSH, ACK] Seq=1606385492 Ack=1292614730 Win=64316 Len=1114
[TCP Dup ACK 214#3] 41991 > https [ACK] Seq=1292614730 Ack=1606385492 Win=64687 Len=0 SLE=2909630688 SRE=2909633764
[TCP Dup ACK 214#4] 41991 > https [ACK] Seq=1292614730 Ack=1606385492 Win=64687 Len=0 SLE=2909630688 SRE=2909635144
[TCP Dup ACK 214#5] 41991 > https [ACK] Seq=1292614730 Ack=1606385492 Win=64687 Len=0 SLE=2909630688 SRE=2909635992
[TCP Dup ACK 214#6] 41991 > https [ACK] Seq=1292614730 Ack=1606385492 Win=64687 Len=0 SLE=2909630688 SRE=2909637372
[TCP Dup ACK 214#7] 41991 > https [ACK] Seq=1292614730 Ack=1606385492 Win=64687 Len=0 SLE=2909630688 SRE=2909638220
[TCP Dup ACK 214#8] 41991 > https [ACK] Seq=1292614730 Ack=1606385492 Win=64687 Len=0 SLE=2909630688 SRE=2909639334
[TCP Retransmission] https > 41991 [PSH, ACK] Seq=1606385492 Ack=1292614730 Win=64316 Len=1380
41991 > https [ACK] Seq=1292614730 Ack=1606395518 Win=65535 Len=0 Recovery
https > 41991 [PSH, ACK] Seq=1606395518 Ack=1292614730 Win=64316 Len=1114
https > 41991 [PSH, ACK] Seq=1606396632 Ack=1292614730 Win=64316 Len=1380
https > 41991 [PSH, ACK] Seq=1606398012 Ack=1292614730 Win=64316 Len=848
https > 41991 [PSH, ACK] Seq=1606398860 Ack=1292614730 Win=64316 Len=1380
https > 41991 [PSH, ACK] Seq=1606400240 Ack=1292614730 Win=64316 Len=848
https > 41991 [PSH, ACK] Seq=1606401088 Ack=1292614730 Win=64316 Len=1380
https > 41991 [PSH, ACK] Seq=1606402468 Ack=1292614730 Win=64316 Len=848
https > 41991 [PSH, ACK] Seq=1606403316 Ack=1292614730 Win=64316 Len=1380
https > 41991 [PSH, ACK] Seq=1606404696 Ack=1292614730 Win=64316 Len=848
41991 > https [ACK] Seq=1292614730 Ack=1606398012 Win=65535 Len=0
https > 41991 [PSH, ACK] Seq=1606405544 Ack=1292614730 Win=64316 Len=1380
https > 41991 [PSH, ACK] Seq=1606406924 Ack=1292614730 Win=64316 Len=848
41991 > https [ACK] Seq=1292614730 Ack=1606400240 Win=65535 Len=0
https > 41991 [PSH, ACK] Seq=1606407772 Ack=1292614730 Win=64316 Len=1380
https > 41991 [PSH, ACK] Seq=1606409152 Ack=1292614730 Win=64316 Len=848
41991 > https [ACK] Seq=1292614730 Ack=1606402468 Win=65535 Len=0
https > 41991 [PSH, ACK] Seq=1606410000 Ack=1292614730 Win=64316 Len=1380
https > 41991 [PSH, ACK] Seq=1606411380 Ack=1292614730 Win=64316 Len=848
41991 > https [ACK] Seq=1292614730 Ack=1606404696 Win=65535 Len=0
https > 41991 [PSH, ACK] Seq=1606412228 Ack=1292614730 Win=64316 Len=1380
https > 41991 [PSH, ACK] Seq=1606413608 Ack=1292614730 Win=64316 Len=848
41991 > https [ACK] Seq=1292614730 Ack=1606405544 Win=64687 Len=0
```

Ack-SLE Hole Size
1303245196 should be 1380

Selective Ack Numbers are mis-calculated

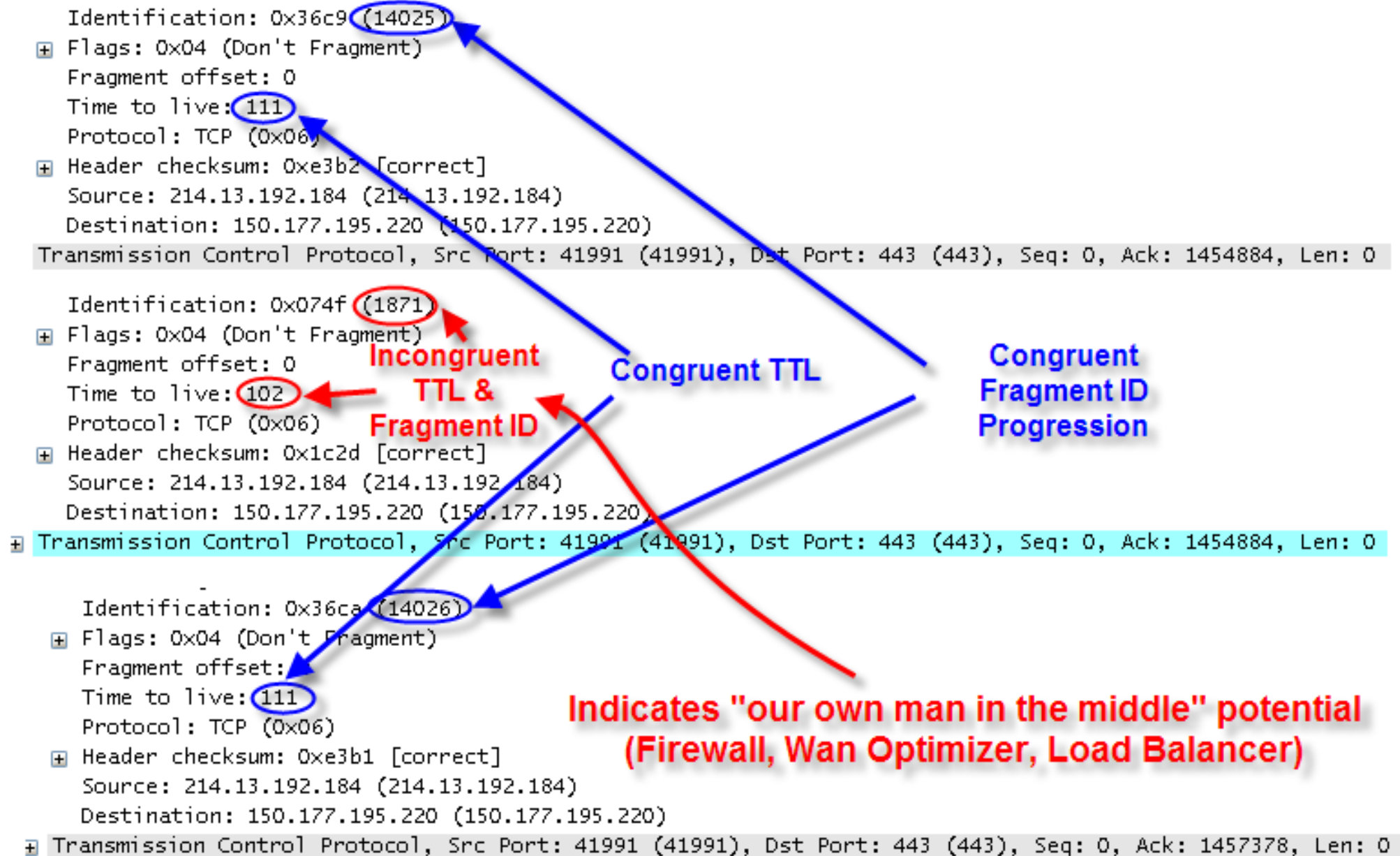
Ack-SLE should have been

SLE-SLR hole correct at 848

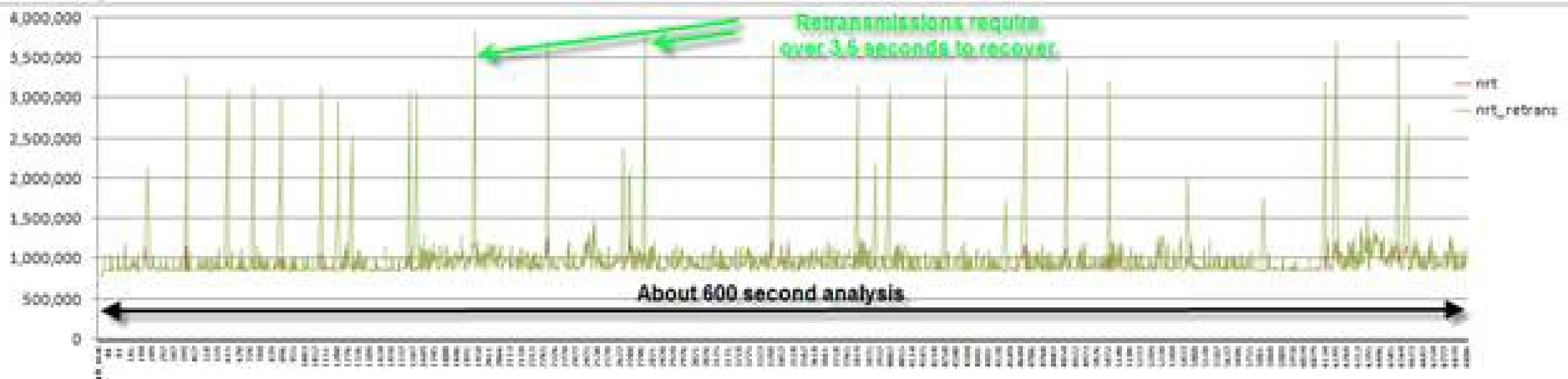
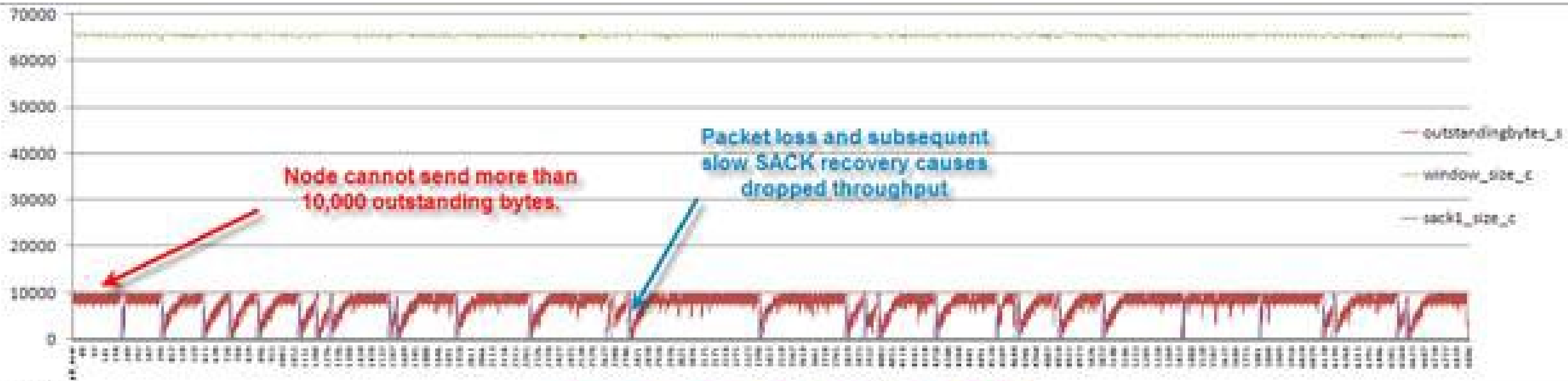
Nevertheless, recovery occurs over three seconds later!

This behavior repeats throughout the session.

HOP/TTL Incongruity “our own man in the middle”

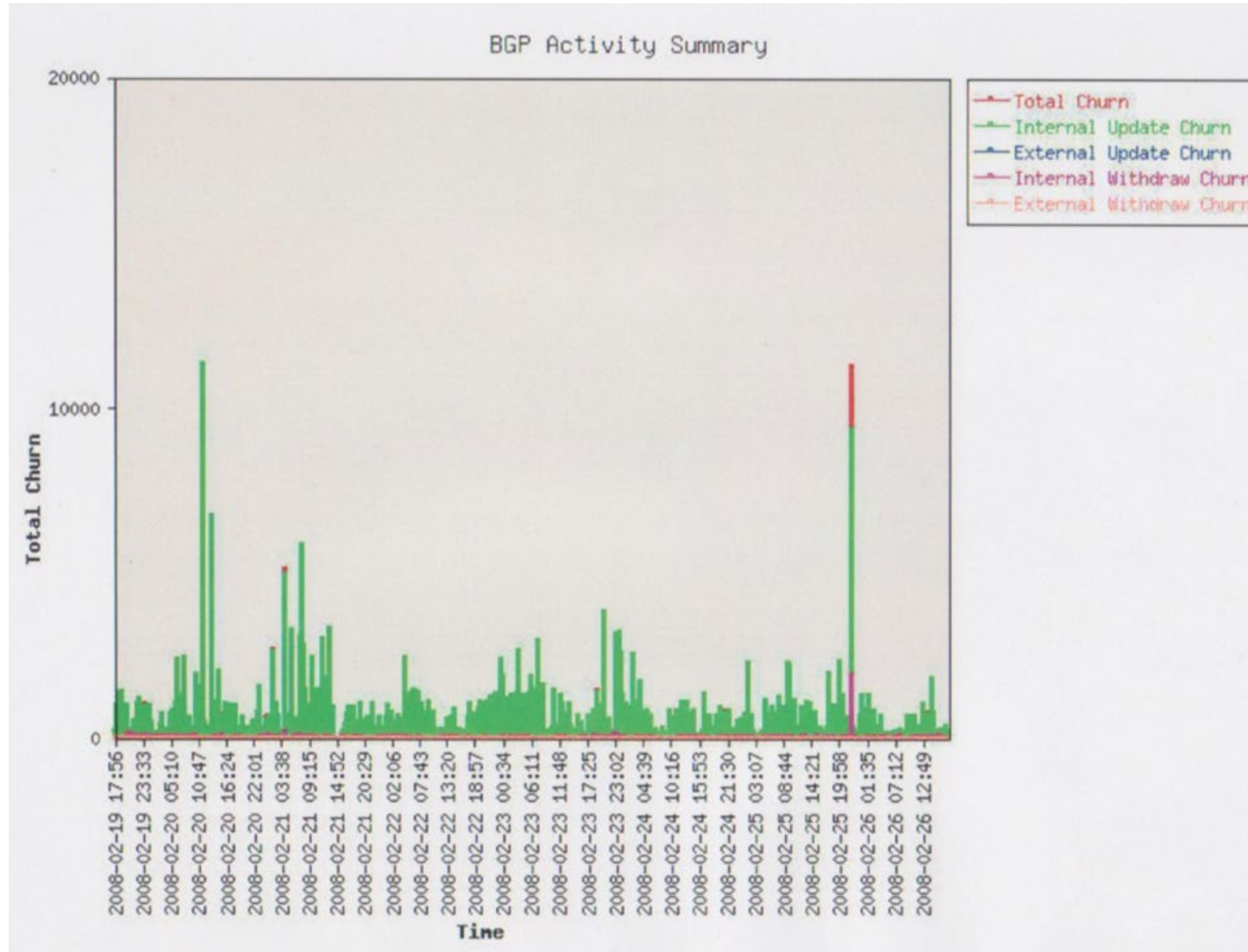


TCP – Session Performance



NAT, PAT or Route Changes Impact on Sessions

Instability of routing metrics



QUIC Decrypted

quic libre tls key test.pcapng

File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help

quic.stream.stream_id == 0

No.	T Since St	Delta	Source	Destination	Protocol	Length	Info
2637	13.099943000	0.043996000	192.168.0.211	quic2.end2endspeed.com	QUIC	1292	Initial, DCID=79a35af38d80a3ff, PKN: 1, PADDING, PING, CRYPTO, CRYPTO, CRYPTO, CRYPTO, PING, CRYPTO, PADDING
2640	13.165631000	0.026792000	quic2.end2endspeed.com	192.168.0.211	QUIC	1294	Handshake, SCID=00000000000010097ec9e4b8c0a22578e354957c, PKN: 0, CRYPTO, CRYPTO
2641	13.166051000	0.000420000	quic2.end2endspeed.com	192.168.0.211	QUIC	1294	Handshake, SCID=00000000000010097ec9e4b8c0a22578e354957c, PKN: 1, CRYPTO
2642	13.166051000	0.000000000	quic2.end2endspeed.com	192.168.0.211	QUIC	1288	Handshake, SCID=00000000000010097ec9e4b8c0a22578e354957c, PKN: 2, CRYPTO
2643	13.166500000	0.000449000	192.168.0.211	quic2.end2endspeed.com	QUIC	93	Handshake, DCID=00000000000010097ec9e4b8c0a22578e354957c, PKN: 2, ACK
2644	13.167734000	0.001234000	192.168.0.211	quic2.end2endspeed.com	QUIC	94	Handshake, DCID=00000000000010097ec9e4b8c0a22578e354957c, PKN: 3, ACK
2648	13.223102000	0.021612000	quic2.end2endspeed.com	192.168.0.211	QUIC	1123	Handshake, SCID=00000000000010097ec9e4b8c0a22578e354957c, PKN: 3, CRYPTO, CRYPTO, CRYPTO
2649	13.225165000	0.002063000	192.168.0.211	quic2.end2endspeed.com	QUIC	94	Handshake, DCID=00000000000010097ec9e4b8c0a22578e354957c, PKN: 4, ACK
2650	13.225969000	0.000804000	192.168.0.211	quic2.end2endspeed.com	QUIC	127	Handshake, DCID=00000000000010097ec9e4b8c0a22578e354957c, PKN: 5, CRYPTO
2651	13.226235000	0.000266000	192.168.0.211	quic2.end2endspeed.com	HTTP3	125	Protected Payload (KP0), DCID=00000000000010097ec9e4b8c0a22578e354957c, PKN: 6, STREAM(2), SETTINGS
2652	13.226678000	0.000443000	192.168.0.211	quic2.end2endspeed.com	HTTP3	494	Protected Payload (KP0), DCID=00000000000010097ec9e4b8c0a22578e354957c, PKN: 7, STREAM(2), PRIORITY_UPDATE, STREAM(0), HE
2658	13.287632000	0.023162000	quic2.end2endspeed.com	192.168.0.211	HTTP3	288	Protected Payload (KP0), PKN: 0, CRYPTO, CRYPTO, DONE, NCI, STREAM(3), STREAM(3), SETTINGS
2659	13.287632000	0.000000000	quic2.end2endspeed.com	192.168.0.211	HTTP3	1292	Protected Payload (KP0), PKN: 1, STREAM(0), HEADERS
2660	13.287632000	0.000000000	quic2.end2endspeed.com	192.168.0.211	QUIC	1292	Protected Payload (KP0), PKN: 2, STREAM(0)
2661	13.287632000	0.000000000	quic2.end2endspeed.com	192.168.0.211	QUIC	1292	Protected Payload (KP0), PKN: 3, STREAM(0)
2662	13.288173000	0.000541000	quic2.end2endspeed.com	192.168.0.211	QUIC	1292	Protected Payload (KP0), PKN: 4, STREAM(0)

Extension: quic_transport_parameters (len=102)
Type: quic_transport_parameters (57)
Length: 102

- Parameter: initial_max_data (len=4) 8585216
- Parameter: initial_max_streams_uni (len=1) 3
- Parameter: initial_max_streams_bidi (len=2) 128
- Parameter: initial_max_stream_data_bidi_local (len=4) 65536
- Parameter: initial_max_stream_data_bidi_remote (len=4) 65536
- Parameter: initial_max_stream_data_uni (len=4) 65536
- Parameter: max_idle_timeout (len=4) 65000 ms
- Parameter: max_udp_payload_size (len=4) 65527
- Parameter: active_connection_id_limit (len=1) 2
- Parameter: GREASE (len=1) 25
- Parameter: ack_delay_exponent (len=1)
- Parameter: original_destination_connection_id (len=8)
- Parameter: initial_source_connection_id (len=20)
- Parameter: stateless_reset_token (len=16)

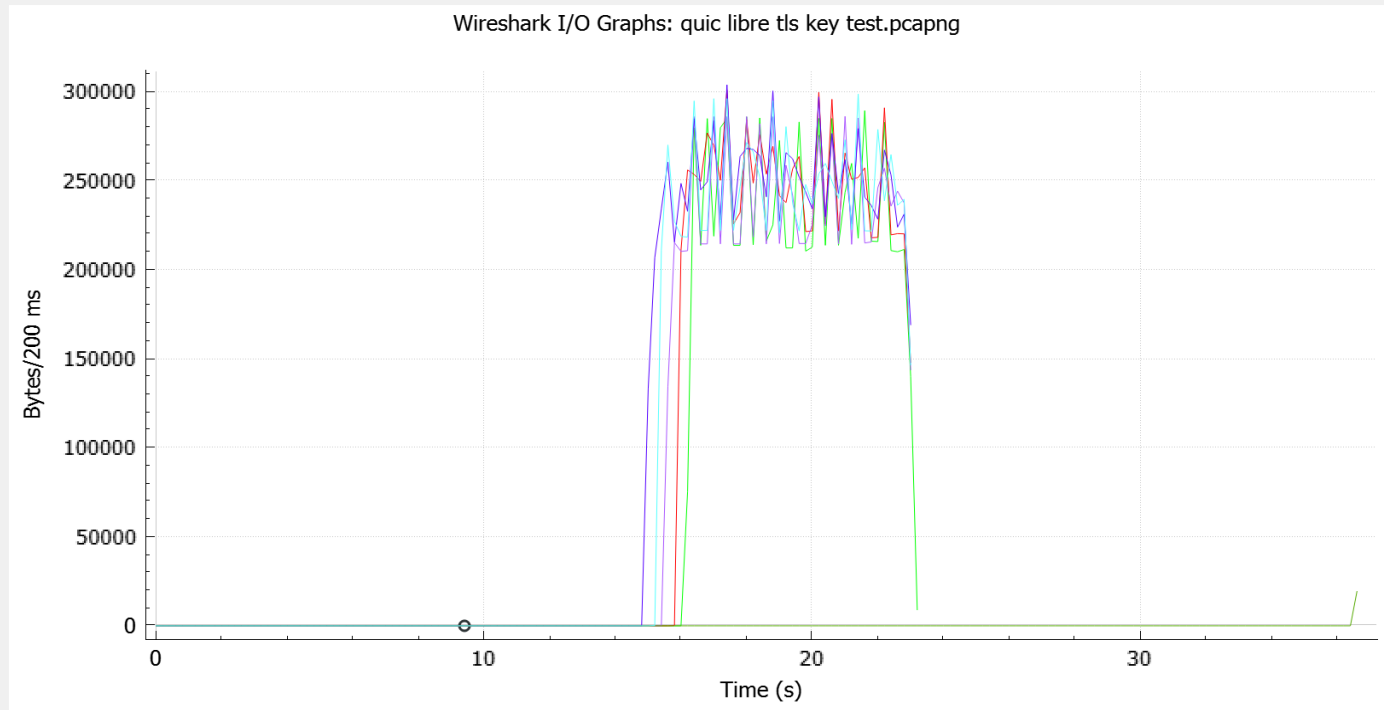
CRYPTO

Frame Type: CRYPTO (0x0000000000000006)
Offset: 125
Length: 926
Crypto Data

TLSv1.3 Record Layer: Handshake Protocol: Certificate (fragment)
Handshake Protocol: Certificate (fragment)
[Reassembled Handshake Message in frame: 2648](#)

```
0000 06 00 40 7d 08 00 00 79 00 77 00 00 00 00 10  @.}...y
0010 00 05 00 03 02 68 33 00 39 00 66 04 04 80 83 00  ....h3.
0020 00 09 01 03 08 02 40 80 05 04 80 01 00 00 06 04  ....@.
0030 80 01 00 00 07 04 80 01 00 00 01 04 80 00 fd e8  ....
0040 03 04 80 00 ff f7 0e 01 02 0b 01 19 0a 01 03 00  ....
0050 08 79 a3 5a f3 8d 80 a3 ff 0f 14 00 00 00 00 00  y.Z....
0060 00 10 09 7e c9 e4 b8 c0 a2 25 78 e3 54 95 7c 02  ....
0070 10 50 08 02 5c 40 f3 2c d2 b0 94 04 80 21 4e 58  .P.\@.,
0080 11 06 40 7d 43 9e 0b 00 0f c7 00 00 0f c3 00 05  @}C...
0090 36 30 82 05 32 30 82 04 1a a0 03 02 01 02 02 12  60..20..
00a0 04 eb 8e ef 4f eb 2b c4 44 29 b3 22 0b c3 ef f9  ...0.+
00b0 ab 33 30 0d 06 09 2a 86 48 86 f7 0d 01 01 0b 05  .30..*.
00c0 00 30 32 31 0b 30 09 06 03 55 04 06 13 02 55 53  .021.0.
00d0 31 16 30 14 06 03 55 04 0a 13 0d 4c 65 74 27 73  1.0..U
00e0 20 45 6e 63 72 79 70 74 31 0b 30 09 06 03 55 04  Encrypt
00f0 03 13 02 52 33 30 1e 17 0d 32 33 30 34 31 36 30  ...R30..
0100 30 33 34 33 34 5a 17 0d 32 33 30 37 31 35 30 30  03434Z..
0110 33 34 33 33 5a 30 21 31 1f 30 1d 06 03 55 04 03  3433Z0!1
0120 13 16 71 75 69 63 32 2e 65 6e 64 32 65 6e 64 73  ..quic2.
0130 70 65 65 64 2e 63 6f 6d 30 82 01 22 30 0d 06 09  ped.com
0140 2a 86 48 86 f7 0d 01 01 01 05 00 03 82 01 0f 00  *.H....
0150 30 82 01 0a 02 82 01 01 00 b0 07 19 4a 30 33 8b  0.....
0160 79 33 5f df 74 ee dd 05 82 d2 38 1c ff b6 35 c6  y3.t...
0170 f0 82 93 b2 2e 0e 63 51 23 b6 2e f6 6a ae e2 ad  ....cQ
0180 3d 4f 48 22 db 5a bb ff 5d 4a 0b 01 bd d3 aa 1b  =OH".Z..
0190 ba 9b 41 a7 5f 13 a7 0c c0 36 98 ae 72 d3 a9 99  ..A....
01a0 ba 42 d8 6e 8c 3c e5 da 0b 43 e8 b1 3b f8 31 97  .B.n<...
01b0 50 e9 8d 75 6a 43 8b ae 32 50 9c 88 95 c4 5c 02  P..ujC..
```

Frame (1294 bytes) Decrypted QUIC (99 bytes) Decrypted QUIC (1060 bytes)

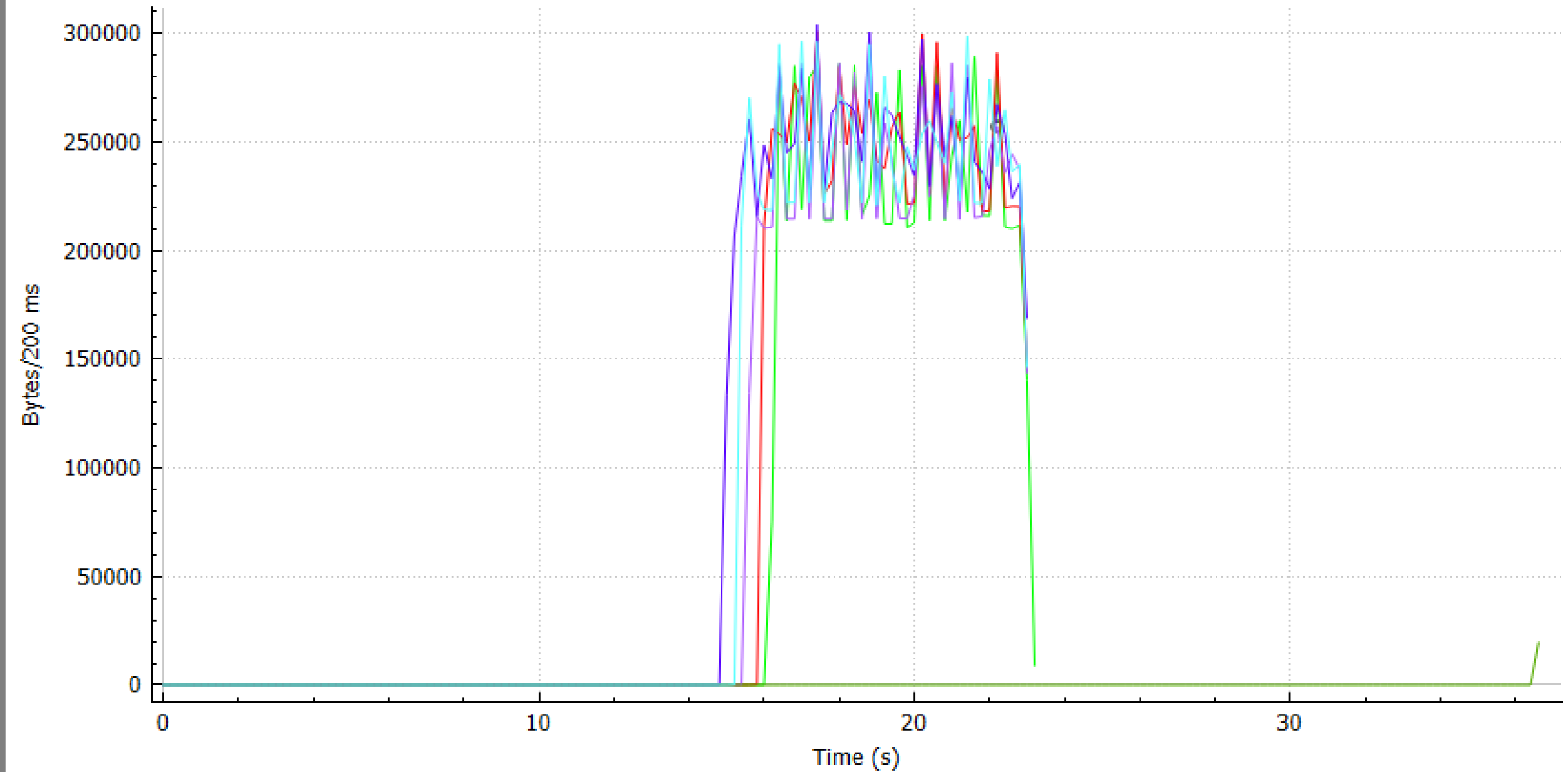


No packets in interval (9.4s).

Enabled	Graph Name	Display Filter	Color	Style	Y Axis
<input type="checkbox"/>	All Packets		Black	Line	Bytes
<input type="checkbox"/>	TCP Errors	tcp.analysis.flags	Dark Red	Bar	Bytes
<input checked="" type="checkbox"/>	Filtered packets	quic.stream.stream_id == 64	Bright Green	Line	Bytes
<input checked="" type="checkbox"/>	Filtered	quic.stream.stream_id == 60	Red	Line	Bytes
<input checked="" type="checkbox"/>	Filtered packets	quic.stream.stream_id == 56	Blue	Line	Bytes
<input checked="" type="checkbox"/>	Filtered packets	quic.stream.stream_id == 48	Purple	Line	Bytes
<input checked="" type="checkbox"/>	Filtered packets	quic.stream.stream_id == 74	Orange	Line	Bytes
<input checked="" type="checkbox"/>	Filtered packets	quic.stream.stream_id == 84	Green	Line	Bytes
<input checked="" type="checkbox"/>	Filtered packets	quic.stream.stream_id == 52	Cyan	Line	Bytes

Mouse drags zooms
 Interval 200 ms Time of day Log scale Automatic Update

Wireshark I/O Graphs: quic libre tls key test.pcapng



How to identify existing QUIC Users

HOPZERO Alarms Captures Investigation Settings bill.alderson@hopzero.com

Find or filter... AppPort = 443 or ... Perspective: Outgoing Protocol: UDP

Not QUIC

QUIC Traffic

C-Type	S-Type	C-bps	S-bps	C-Bytes	S-Bytes	Data	App Name	App Port	Sessions	C-IP	C-DNS	C-Fraud	Low Hops	High Hops	Hop Jitter	S-IP	S-DNS	C-City	C-Country	S-City	S-Country
known	Business	964.12 kbps	83.00 bps	31.41 MB	2.72 kB	✓	https	443	1	192.168.1.1	142.250.190.101	0	24	26	2	142.60	142.250.190.101	Unknown	United States	Unknown	United States
known	Business	1.46 Mbps	267.00 bps	7.44 MB	2.72 kB	✓	https	443	1	192.168.1.1	144.231.142.168	0	26	26	0	144.231	144.231.142.168	Unknown	United States	Unknown	United States
known	Business	1.26 Mbps	1.12 kbps	3.07 MB	1.36 kB	✓	https	443	1	192.168.1.1	142.168.1.1	0	25	26	1	142.168	142.168.1.1	Unknown	United States	Unknown	United States
known	Unknown	4.01 Mbps	2.00 kbps	2.72 MB	1.36 kB	✓	https	443	1	192.168.1.1	142.168.1.1	0	26	26	0	142.168	142.168.1.1	Unknown	United States	Unknown	United States
known	Business	1.71 kbps	0.00 bps	2.67 MB	275 B	✓	https	443	7	192.168.1.1	142.168.1.1	0	24	26	2	142.168	142.168.1.1	Unknown	United States	Unknown	United States
known	Business	33.11 kbps	19.00 bps	2.40 MB	1.36 kB	✓	https	443	6	192.168.1.1	142.168.1.1	0	39	52	13	142.168	142.168.1.1	Unknown	United States	Unknown	United States
known	Business	1.25 kbps	0.00 bps	1.97 MB	216 B	✓	https	443	6	192.168.1.1	142.168.1.1	0	25	41	16	142.168	142.168.1.1	Unknown	United States	Unknown	United States
known	Business	13.39 Mbps	13.55 kbps	1.34 MB	1.36 kB	✓	https	443	1	192.168.1.1	142.168.1.1	0	24	24	0	142.168	142.168.1.1	Unknown	United States	Unknown	United States
known	Business	162.80 kbps	3.34 kbps	995.06 kB	20.40 kB	✓	https	443	16	192.168.1.1	142.168.1.1	0	38	40	2	142.168	142.168.1.1	Unknown	United States	Unknown	United States
known	Business	37.55 kbps	13.20 bps	4.91 MB	1.36 kB	✓	https	443	1	192.168.1.1	142.168.1.1	0	39	52	13	142.168	142.168.1.1	Unknown	United States	Unknown	United States
known	Business	36.11 kbps	493.00 bps	1.13 kB	6.71 kB	✓	https	443	1	192.168.1.1	142.168.1.1	0	25	41	16	142.168	142.168.1.1	Unknown	United States	Unknown	United States
known	Business	1.25 kbps	0.00 bps	1.97 MB	216 B	✓	https	443	6	192.168.1.1	142.168.1.1	0	24	24	0	142.168	142.168.1.1	Unknown	United States	Unknown	United States
known	Business	162.80 kbps	3.34 kbps	995.06 kB	20.40 kB	✓	https	443	16	192.168.1.1	142.168.1.1	0	38	40	2	142.168	142.168.1.1	Unknown	United States	Unknown	United States
known	Business	37.55 kbps	13.20 bps	4.91 MB	1.36 kB	✓	https	443	1	192.168.1.1	142.168.1.1	0	39	52	13	142.168	142.168.1.1	Unknown	United States	Unknown	United States
known	Business	36.11 kbps	493.00 bps	1.13 kB	6.71 kB	✓	https	443	1	192.168.1.1	142.168.1.1	0	25	41	16	142.168	142.168.1.1	Unknown	United States	Unknown	United States
known	Business	1.25 kbps	0.00 bps	1.97 MB	216 B	✓	https	443	6	192.168.1.1	142.168.1.1	0	24	24	0	142.168	142.168.1.1	Unknown	United States	Unknown	United States
known	Business	3.23 Mbps	14.00 kbps	313.19 kB	1.36 kB	✓	https	443	1	192.168.1.1	142.168.1.1	0	26	26	0	142.168	142.168.1.1	Unknown	United States	Unknown	United States
known	Business	125.88 kbps	1.58 kbps	215.70 kB	2.72 kB	✓	https	443	2	192.168.1.1	142.168.1.1	0	54	56	2	142.168	142.168.1.1	Unknown	United States	Unknown	United States
known	Business	51.49 kbps	690.00 bps	206.30 kB	2.76 kB	✓	https	443	3	192.168.1.1	142.168.1.1	0	56	56	0	142.168	142.168.1.1	Unknown	United States	Unknown	United States
known	Business	4.08 kbps	68.00 bps	163.70 kB	2.72 kB	✓	https	443	2	192.168.1.1	142.168.1.1	0	39	41	2	142.168	142.168.1.1	Unknown	United States	Unknown	United States
known	Business	47.43 kbps	846.00 bps	140.96 kB	2.81 kB	✓	https	443	4	192.168.1.1	142.168.1.1	0	25	40	15	142.168	142.168.1.1	Unknown	United States	Unknown	United States

Map metric: S-ASOrg

531 GOOGLE 27 OPENDNS 23 Google LLC 5 Cisco OpenDNS, LLC

QUIC UDP Traffic Visibility

UDP vs TCP vs QUIC Firewall

QUIC uses UDP Port 443 to Servers

UDP firewall state is basic compared to TCP

QUIC encrypts everything above UDP except small part of initial packets containing Link ID

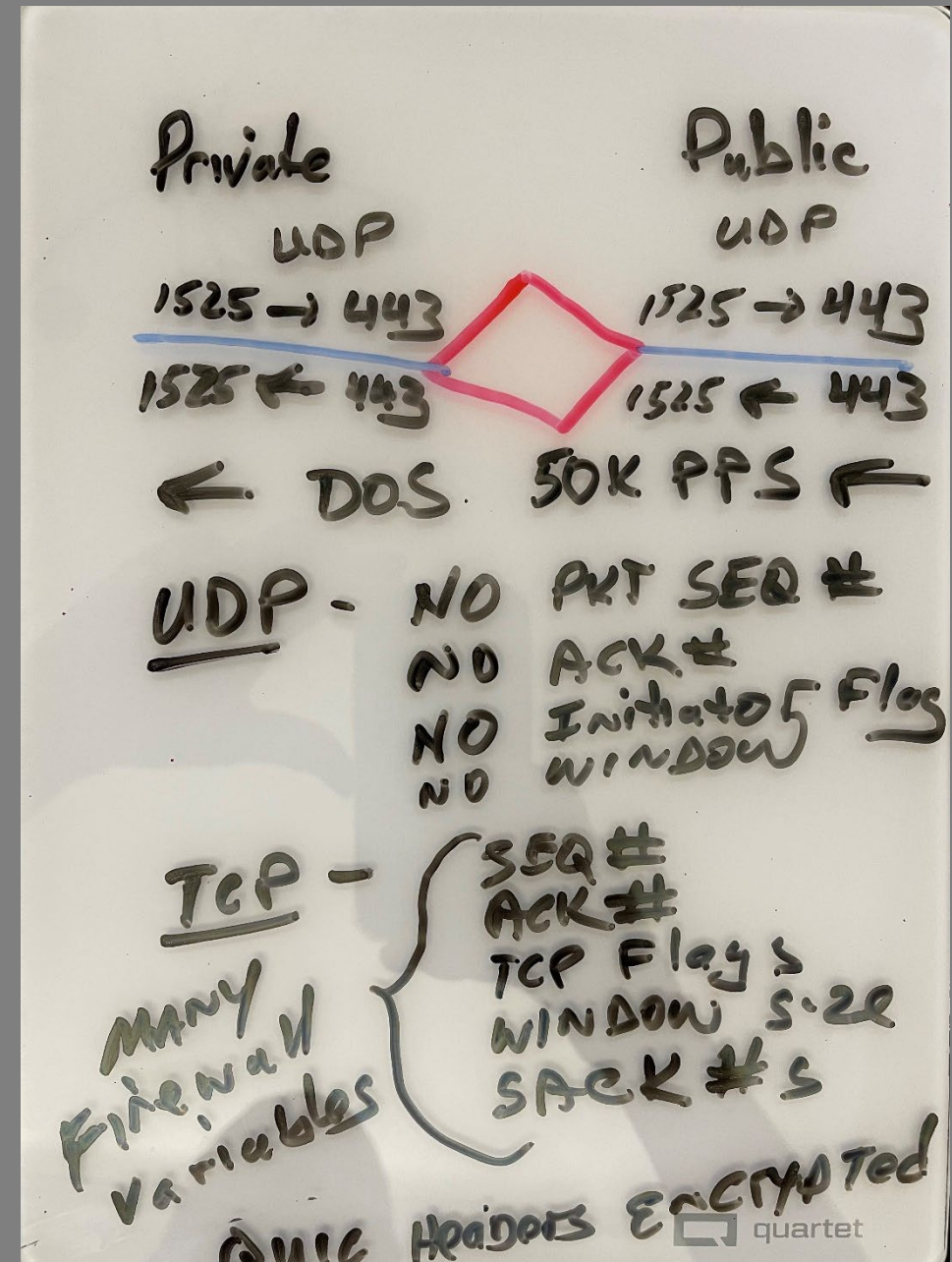
Firewalls blind - cannot inspect protocol or content!

Apps- Browsers now, but many more uses coming

QUIC overcomes NAT issues

It only cares about Link ID's, not Port#'s not IP's

Sends large MTU initial packet sizes



What to do next?

Personal / Home:

- A.) Update Router-Firewall
- B.) Use QUIC carefully
- C.) Join SecurityInstitute.com QUIC Protocol Space

Work / School:

- A.) Inform IT Security about QUIC working, or not.
- B.) Careful they don't "shoot the messenger"

Security Pro's:

- A.) Learn tools to identify QUIC
- B.) Become Certified in QUIC <https://SecurityInstitute.com>

Vendors:

- A.) Let us know about QUIC supported products
- B.) Tell us about new QUIC products coming
- C.) Sponsor and participate in SecurityInstitute.com QUIC

Triple+ Web Performance

at a Price...

**3x
FASTER**


**SECURITY
THREATS**

**What Will
You Do?**



QUIC

Decrypting TLS & QUIC Headers

To record QUIC session information including encryption keys, you can use the `SSLKEYLOGFILE` environment variable. This method is supported by many TLS libraries, such as OpenSSL and BoringSSL, which are often used in QUIC implementations. The `SSLKEYLOGFILE` environment variable specifies a file path where the TLS session secrets will be written, enabling decryption of QUIC traffic for analysis and diagnostic purposes.

Here's how to use the `SSLKEYLOGFILE` method:

1. Set the `SSLKEYLOGFILE` environment variable to the desired file path before starting the client or server application that uses QUIC:

For Linux and macOS:

javascript

```
export SSLKEYLOGFILE=/path/to/your/sslkeylogfile.txt
```

For Windows:

vbnet

```
set SSLKEYLOGFILE=C:\path\to\your\sslkeylogfile.txt
```

2. Run the client or server application that uses QUIC. The application will write the TLS session keys to the specified file as they are established.

3. Use the collected keys to decrypt and analyze the QUIC traffic. Tools like Wireshark can use these keys to decode encrypted QUIC streams in captured packet data.

Remember that handling encryption keys securely is critical, as exposing these keys can compromise the security and privacy of the QUIC sessions. Only use this method for diagnostic purposes and with the appropriate permissions, and always follow best practices for handling sensitive data.

*Some Chat GPT helped find and organize some of this text.

QUIC Improvements over TCP

Head-of-line (HOL) blocking is a problem that occurs in TCP when a lost or delayed packet prevents the processing of subsequent packets in the same data stream, causing increased latency and reduced performance. QUIC solves the head-of-line blocking problem more effectively than TCP by employing the following techniques:

1. **Independent streams:** QUIC uses multiplexed streams over a single connection, where each stream is independent of the others. This means that if a packet is lost or delayed in one stream, it does not affect the other streams. In contrast, TCP treats all data within a connection as a single, ordered byte stream, which means that any packet loss or delay can block the entire connection.
2. **Stream-level error correction:** QUIC handles error correction and retransmissions at the stream level, rather than at the connection level, as in TCP. This enables QUIC to recover from packet losses in one stream without affecting other streams, further reducing the impact of head-of-line blocking.
3. **Faster packet retransmissions:** QUIC can retransmit lost packets more quickly than TCP because it does not rely on a single, global retransmission timer. Instead, QUIC uses per-packet timers and can quickly detect and retransmit lost packets without waiting for a full round-trip time, as is typically the case with TCP.
4. **Selective acknowledgments:** QUIC uses selective acknowledgments (ACKs) to inform the sender about received packets, as well as any gaps in the sequence of received packets. This allows the sender to quickly identify lost packets and retransmit them, reducing the impact of head-of-line blocking.

By using these techniques, QUIC effectively mitigates the head-of-line blocking problem, leading to better performance, reduced latency, and improved user experience, especially in environments with high packet loss or network congestion.

0-RTT Data Request Response Size

In a 0-RTT (Zero Round-Trip Time) session, the amount of data that can be sent in the initial request depends on the server's maximum allowed 0-RTT data size, which can vary depending on the server's configuration and preferences. There isn't a fixed theoretical capacity for all cases, as it depends on the server's specific settings.

However, it's important to note that 0-RTT data should generally be limited to a small amount, as sending large amounts of data in the initial request could increase the risk of replay attacks. The server must enforce proper anti-replay measures and limit the use of 0-RTT data to mitigate this risk.

In practice, 0-RTT data is typically used for non-sensitive, idempotent requests like HTTP GET requests or other actions that can be safely retried without causing unintended side effects. This ensures that even if a replay attack occurs, the consequences are minimal.

In TLS 1.3, the "max_early_data_size" parameter within the "NewSessionTicket" message specifies the maximum amount of 0-RTT data a client can send during a 0-RTT session. The "max_early_data_size" is a 32-bit unsigned integer, so the maximum value that can be represented is $2^{32} - 1$ bytes, which is equal to 4,294,967,295 bytes or approximately 4 GiB.

However, it's important to remember that setting such a high limit for 0-RTT data is not recommended, as it could increase the risk of replay attacks. In practice, servers are likely to set much smaller limits to ensure security and protect against potential abuse.

QUIC Frame Concept

The QUIC protocol uses a modular and extensible framing mechanism, which allows for the efficient encoding of different types of data while also providing flexibility for future enhancements. Some common types of frames in QUIC include:

STREAM frames: These frames carry application data between endpoints and are used for reliable, in-order transmission of data within a specific QUIC stream.

ACK frames: These frames are sent by the receiving endpoint to acknowledge the receipt of one or more packets, indicating the packets' sequence numbers and any gaps (i.e., lost or delayed packets).

MAX_DATA and MAX_STREAM_DATA frames: These frames are used for flow control, with MAX_DATA controlling the overall amount of data that can be sent across all streams and MAX_STREAM_DATA controlling the amount of data that can be sent within a specific stream.

RESET_STREAM frames: These frames are sent by an endpoint to indicate that it wants to abruptly terminate a stream without completing the transmission of all data.

CONNECTION_CLOSE and APPLICATION_CLOSE frames: These frames are used to signal the termination of a QUIC connection, either due to an error or a graceful shutdown initiated by the application.

PING frames: These frames are used to test the connection's liveness and to keep the connection alive in the presence of idle timeouts.

By using frames to carry various types of information, QUIC enables efficient, flexible, and extensible communication between endpoints while maintaining performance and security.

Introduction to QUIC for Network and Security Technologists

QUIC (Quick UDP Internet Connections) is a transport layer protocol started by Jim Roskind at Google (Now AWS) to improve the security, performance, and reliability of web connections. QUIC uses UDP as its transport protocol, providing faster connection establishment, reduced latency, and built-in encryption.

Internet Engineering Task Force IETF changed its name to QUIC – no acronym to lose its Google roots. Greatly enhancing and integrating with TCP features.

Encryption and security: QUIC incorporates Transport Layer Security (TLS) 1.3, ensuring all transmitted data is encrypted by default. This enhances security compared to older HTTP/2 connections, which do not always require encryption.

Faster connection establishment: QUIC reduces the number of round trips required to establish a secure connection, resulting in a faster and more efficient process compared to traditional TCP/TLS connections.

0-RTT connection resumption: QUIC supports 0-RTT (Zero Round-Trip Time) connection resumption, allowing for faster reconnections between clients and servers that have previously communicated. This feature should be implemented with caution, as it can pose a risk of replay attacks.

Connection migration: QUIC allows for connection migration, enabling a client to change its IP address without losing the connection. This feature improves the stability of secure connections in mobile or unstable network environments.

Multiplexed streams and head-of-line blocking: QUIC's support for multiplexed streams can help mitigate head-of-line blocking, enhancing the performance and security of connections by reducing latency.

Forward error correction: QUIC uses forward error correction (FEC) to reduce the impact of packet loss, enhancing the reliability and security of connections.

Potential vulnerabilities: While QUIC is designed with security in mind, potential vulnerabilities exist, such as 0-RTT vulnerabilities, key update attacks, DoS attacks, Connection ID privacy concerns, and implementation flaws. Awareness and mitigation strategies are essential for ensuring optimal security.

Limited adoption and compatibility: QUIC is becoming widely adopted, with more implementations monthly. Network and security technologists should be prepared to work with both QUIC-enabled and non-QUIC environments.

QUIC's Top 5 Security Vulnerabilities

0-RTT vulnerabilities: The 0-RTT (Zero Round-Trip Time) feature can make QUIC connections more susceptible to replay attacks. An attacker may intercept and replay a 0-RTT connection attempt to gain unauthorized access. To mitigate this risk, servers should enforce proper anti-replay measures and limit the use of 0-RTT data.

Key update attacks: QUIC's key update mechanism, which periodically updates encryption keys, could be exploited by attackers to force clients or servers to use weak or compromised keys. This issue can be addressed by implementing proper key management practices and ensuring that keys are securely generated and stored.

Denial of Service (DoS) attacks: QUIC's reliance on the User Datagram Protocol (UDP) could make it more susceptible to DoS attacks. Attackers might flood a server with malformed or large packets to exhaust its resources. Server operators should employ rate limiting, filtering, and other techniques to prevent such attacks.

Connection ID privacy concerns: QUIC's use of Connection IDs to maintain sessions can improve privacy but may also be exploited by attackers to track users across different connections. Ensuring that Connection IDs are generated and managed securely can help minimize this risk.

Implementation flaws: As with any protocol, security issues may arise due to flaws in the implementation of QUIC by software developers. To address this, it is essential to use well-tested and regularly updated libraries, adhere to best practices, and perform thorough security audits and testing of QUIC-enabled applications.

QUIC: Top 10 Things to Know

Encryption by default: QUIC incorporates built-in encryption using Transport Layer Security (TLS) 1.3, ensuring that all data transmitted is secure by default. This is an improvement over HTTP/2, which does not require encryption.

Connection establishment: QUIC reduces the number of round trips required to establish a secure connection, speeding up the process and making it more efficient.

0-RTT connection resumption: QUIC allows for 0-RTT (Zero Round-Trip Time) connection resumption, enabling faster reconnections between clients and servers that have previously communicated. This can, however, pose a risk of replay attacks if not properly implemented.

Improved privacy: QUIC's connection identifiers do not reveal user IP addresses, making it harder for eavesdroppers to track users across different connections and improving privacy.

Resistance to replay attacks: QUIC has built-in mechanisms to counter replay attacks, but proper implementation is essential to ensure the security of the protocol.

Connection migration: QUIC supports connection migration, allowing a client to change its IP address without losing the connection. This can help maintain secure connections, even in mobile or unstable network environments.

Forward error correction: QUIC uses forward error correction (FEC) to reduce the impact of packet loss, enhancing reliability and security.

Reduced impact of head-of-line blocking: QUIC's multiplexed streams can help mitigate head-of-line blocking, improving the performance and security of connections.

Key QUIC vs. TCP Improvements

Head-of-line (HOL) blocking is a problem that occurs in TCP when a lost or delayed packet prevents the processing of subsequent packets in the same data stream, causing increased latency and reduced performance. QUIC solves the head-of-line blocking problem more effectively than TCP by employing the following techniques:

- 1. Independent streams:** QUIC uses multiplexed streams over a single connection, where each stream is independent of the others. This means that if a packet is lost or delayed in one stream, it does not affect the other streams. In contrast, TCP treats all data within a connection as a single, ordered byte stream, which means that any packet loss or delay can block the entire connection.
- 2. Stream-level error correction:** QUIC handles error correction and retransmissions at the stream level, rather than at the connection level, as in TCP. This enables QUIC to recover from packet losses in one stream without affecting other streams, further reducing the impact of head-of-line blocking.
- 3. Faster packet retransmissions:** QUIC can retransmit lost packets more quickly than TCP because it does not rely on a single, global retransmission timer. Instead, QUIC uses per-packet timers and can quickly detect and retransmit lost packets without waiting for a full round-trip time, as is typically the case with TCP.
- 4. Selective acknowledgments:** QUIC uses selective acknowledgments (ACKs) to inform the sender about received packets, as well as any gaps in the sequence of received packets. This allows the sender to quickly identify lost packets and retransmit them, reducing the impact of head-of-line blocking.

QUIC Encryption Explained vs TCP

QUIC packet header encryption is a mechanism that protects certain parts of the QUIC packet header from being observed or modified by third parties, such as middleboxes or eavesdroppers. This enhances privacy and security compared to traditional transport protocols like TCP, where some header information remains exposed.

In QUIC, the packet payload and certain parts of the header are encrypted together using the same encryption keys. The payload is encrypted using modern cryptographic algorithms like AES-GCM or ChaCha20-Poly1305, which also provide authentication.

Not all parts of the QUIC header are encrypted. The packet number, for example, remains in the clear. The reason is to allow for better handling of packet loss and reordering, as the packet number helps identify which packets have been received and which ones are still missing. QUIC:

- 1. Encrypts the payload:** The payload data (e.g., application data) is encrypted using a symmetric key negotiated during the QUIC handshake.
- 2. Protect specific header fields:** QUIC protects certain header fields, such as the Key Phase, Spin Bit, and some reserved bits, using a technique called "header protection." This is done by generating a header protection mask based on the packet encryption key and the unprotected header.
- 3. Apply the header protection mask:** The header protection mask is XORed with the specific header fields that need to be protected. This process encrypts these fields and prevents them from being observed or modified by third parties.
- 4. QUIC uses TLS for header encryption:** The header protection mechanism is built into the QUIC protocol itself. As a result, there's no "second encryption" layer for the header compared to the payload. The encryption keys for both payload and header protection are derived from the same initial secret negotiated during the QUIC handshake.
- 5. QUIC uses per packet encryption vs. TCP Stream-based encryption:** When TCP is combined with TLS, it provides stream-based encryption, which means that the entire data stream is encrypted as a whole rather than on a per-packet basis. This can make it more challenging to handle packet loss or reordering, as lost or out-of-order packets can cause the entire stream to stall until the missing packet is received.
- 6. TCP has Exposed headers:** In, some header information remains exposed, which can potentially be exploited by attackers or used for network analysis by third parties. This can be a privacy and security concern compared to QUIC's header protection.
- 7. TCP does not natively support connection migration:** If a user changes their network connection (e.g., switching from Wi-Fi to cellular data), the existing TCP connection/s must be terminated, and a new connection needs to be established, causing additional latency and potential disruptions.

QUIC vs. TCP Encryption

Per-packet encryption: QUIC encrypts every packet individually with packet numbers in the clear. This allows for better handling of packet loss and reordering compared to TCP. QUIC uses modern cryptographic algorithms such as AES-GCM or ChaCha20-Poly1305 for encryption and authentication.

Packet header protection: QUIC also protects packet headers from being observed or modified by third parties. This enhances privacy and security while preventing potential attacks that could exploit exposed header information.

Connection migration: QUIC supports connection migration, which means that a connection can be transferred between IP addresses without breaking the connection. This can be useful in cases of network changes or mobility (e.g., when a user switches from Wi-Fi to cellular data). Per-packet encryption enables this feature, as packets can be independently decrypted and processed.

TCP encryption (with TLS):

Protocol: TCP is built on top of IP and provides a reliable, ordered, and error-checked delivery of data between applications. TCP is the foundation for many application-level protocols, including HTTP, HTTPS, and FTP.

Stream-based encryption: When TCP is combined with TLS, it provides stream-based encryption, which means that the entire data stream is encrypted as a whole rather than on a per-packet basis. This can make it more challenging to handle packet loss or reordering, as lost or out-of-order packets can cause the entire stream to stall until the missing packet is received.

Exposed headers: In TCP, some header information remains exposed, which can potentially be exploited by attackers or used for network analysis by third parties. This can be a privacy and security concern compared to QUIC's header protection.

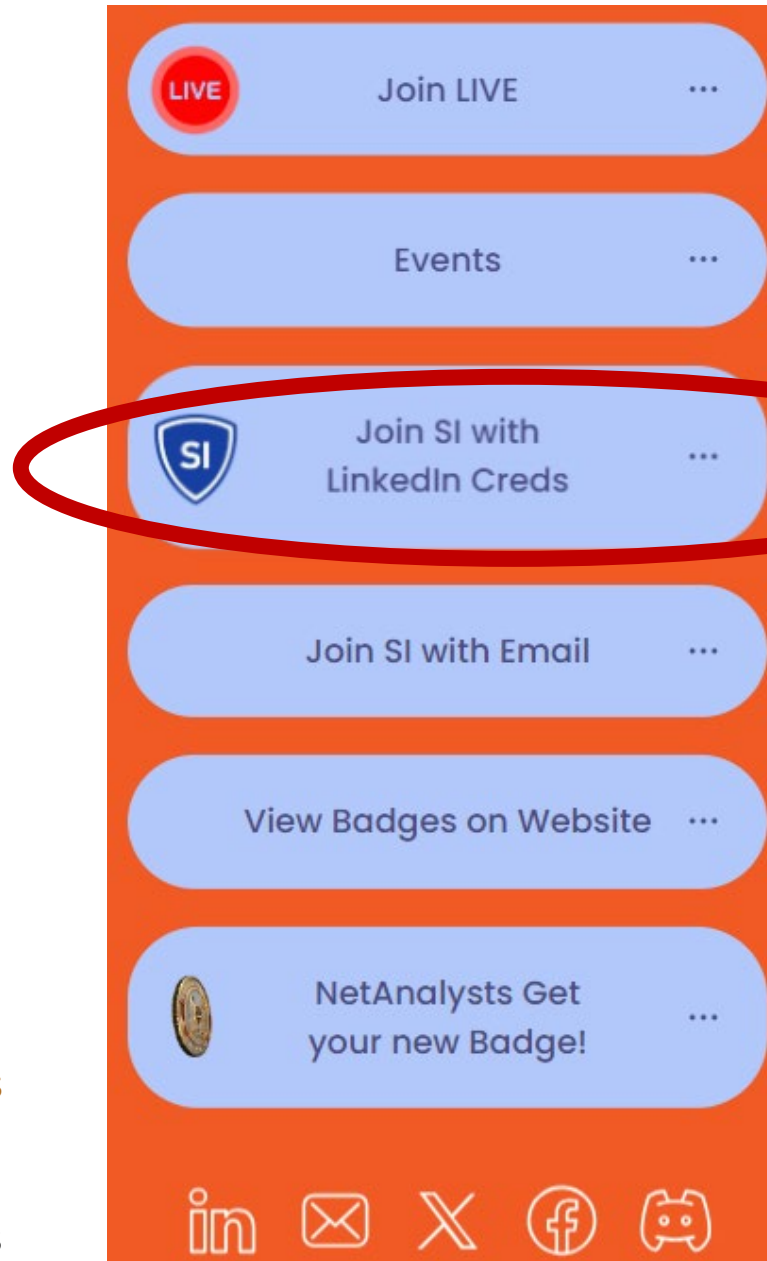
Connection migration limitations: TCP does not natively support connection migration. If a user changes their network connection (e.g., switching from Wi-Fi to cellular data), the existing TCP connection must be terminated, and a new connection needs to be established, causing additional latency and potential disruptions.



SECURITY
INSTITUTE



Topics Prof Assn's Conferences SME's Vendors
Content Videos LiveStream Collaboration
Root Cause Analysis Chat GPT Cybersecurity
QUIC Protocol SharkFest - WireShark Betty Dubois
ISSA / ISC2 Leadership Podcasts



Packetman007