

# QUIC Tutorial

A New Internet Transport

## What to expect in the next hour

- Brief history
- Motivations
- High-level overview of work
- Where the working group is today
  
- You may find this tutorial useful if:
  - HTTP/2 and QUIC are buzzwords to you
  - You can break BGP but think of TCP as too high-level
  - You can write a mobile app in 15 mins but have never seen a tcpdump trace

# Caveat Emptor

- This is not a QUIC working group meeting
- If you are already participating in QUIC work
  - Feel free to offer clarifications at any time
  - No questions for you!  
(Wouldn't you much rather be staring at your laptop?)

## A QUIC history

- Experimental protocol, deployed at Google starting in 2014
  - Between Google services and Chrome
  - Improved page load latency, video rebuffer rate
  - Successful experiment today
  - ~35% of Google's egress traffic (~7% of Internet traffic)
  - Akamai deployment in 2016
- QUIC wg formed in Oct 2016
  - Modularize and standardize QUIC in parts
  - HTTP as initial application

# What's HTTP/2?

- Q: What does a webpage look like?
- A: Containers, scripts, many objects

The screenshot shows the IETF website homepage. The IETF logo is circled in red. A news item titled "IETF 104 in Prague" is also circled in red. A city skyline image is circled in red. The page content includes:

## The Internet Engineering Task Force (IETF®)

The goal of the IETF is to make the Internet work better.

The mission of the IETF is to make the Internet work better by producing high quality, relevant technical documents that influence the way people design, use, and manage the Internet. Newcomers to the IETF should [start here](#).

**News**

- IETF 104 in Prague
- Transition to Host IETF 98!
- Chair's Blog
- IETF Daily Dose

**Next Meeting: IETF 98, Chicago, IL, USA**

**IETF 98, Chicago, IL, USA (UTC -5)**  
March 26-31, 2017

- Register
- Important Dates
- Wiki
- IETF 98 Agenda
- Meeting Materials
- Remote Participation
- Hackathon (open to public)

**Recent Meeting: IETF 97, Seoul, South Korea**

**Email Archives**

A new mail archive tool realizing the requirements developed in RFC 6778 is now in use:

- [Search all IETF email archives](#)

If you choose to log in, use your datatracker credentials.  
([Read full announcement in the archives here.](#))

**Internet-Drafts and RFCs Quick Search**

**Related Web Pages**

[IASA and IAOC](#) | [IAB](#) | [RFC Editor](#) | [IANA](#) | [IRTF](#) | [IETF Trust](#) | [ISOC](#)  
[ISOC Fellowship to the IETF Program](#)

**Force (IETF) is an organized activity of the Internet Society (ISOC).**

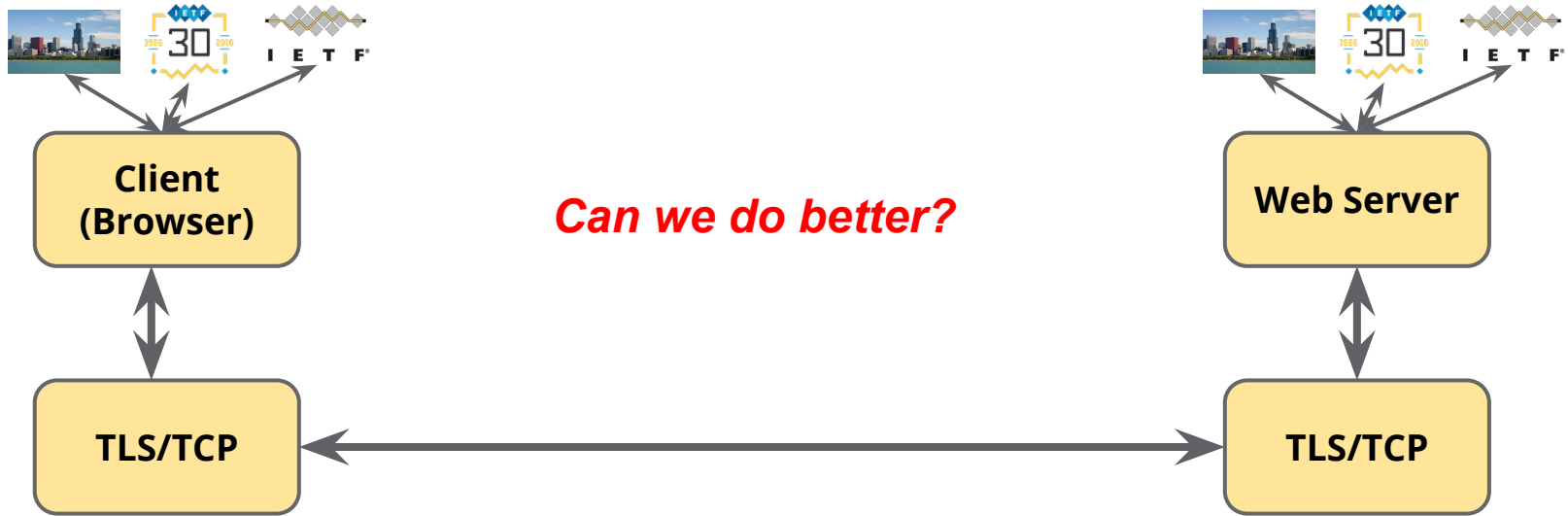
The screenshot shows a file explorer view of the directory structure for www.ietf.org:

- top
- www.ietf.org
  - CSS
    - ietf.js
    - ietf.css
  - images
    - IETF-30-transparent-100px.png
    - ams\_logo.png
    - chat-trans.png
    - ietflogotrans.gif
    - isoc\_logo.gif
    - (index)
  - www6.ietf.org
    - images
      - isoc-2016-logo.png
    - meeting/98/images
      - chicagoskyline.jpg

# First, how does HTTP/1 work?

- Connection setup... the long way
  - 1 round-trip to set up a TCP connection
  - 2 round-trips to set up a TLS 1.2 connection
  - (before you rush to the mic, TFO and TLS 1.3 shortly)
- After setup, HTTP requests/responses flow over connection

# First, how does HTTP/1 work?

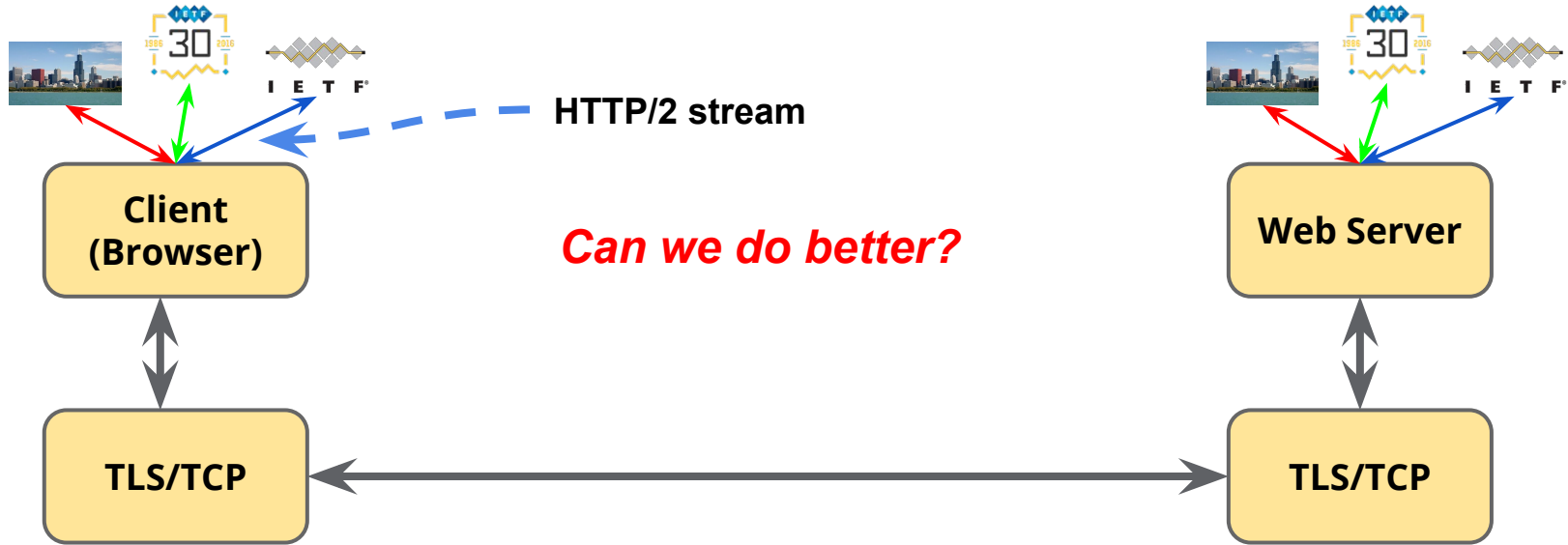


# Dealing with head-of-line (HoL) blocking





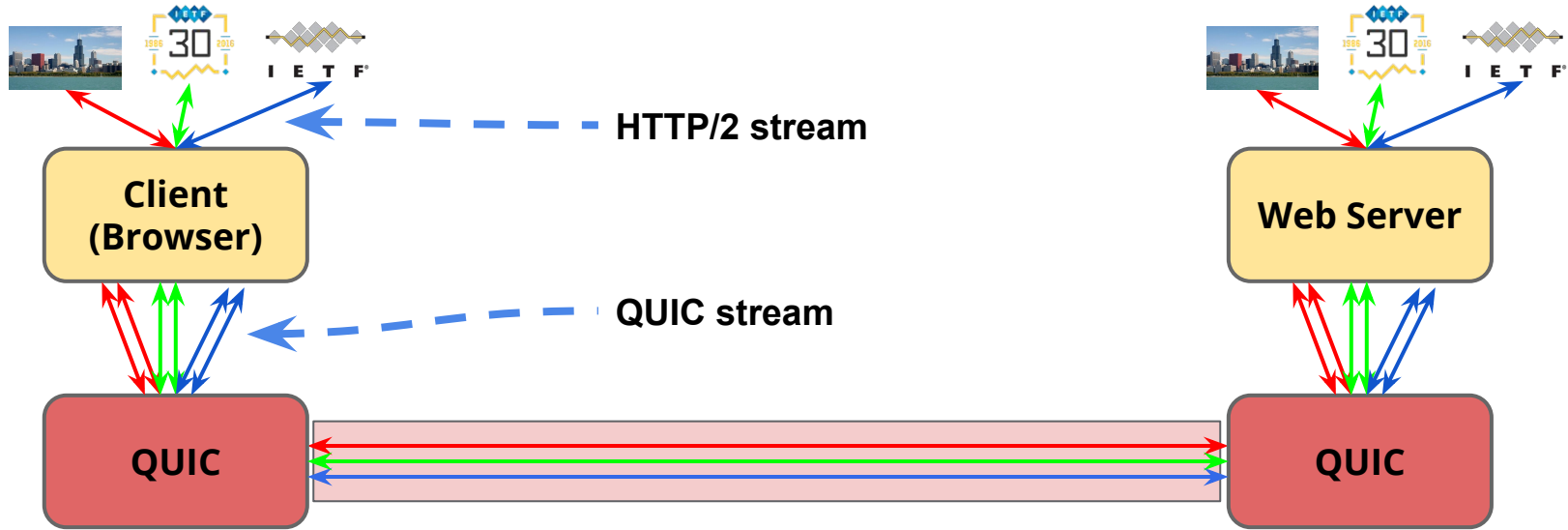
# Better handling of HoL blocking: HTTP/2



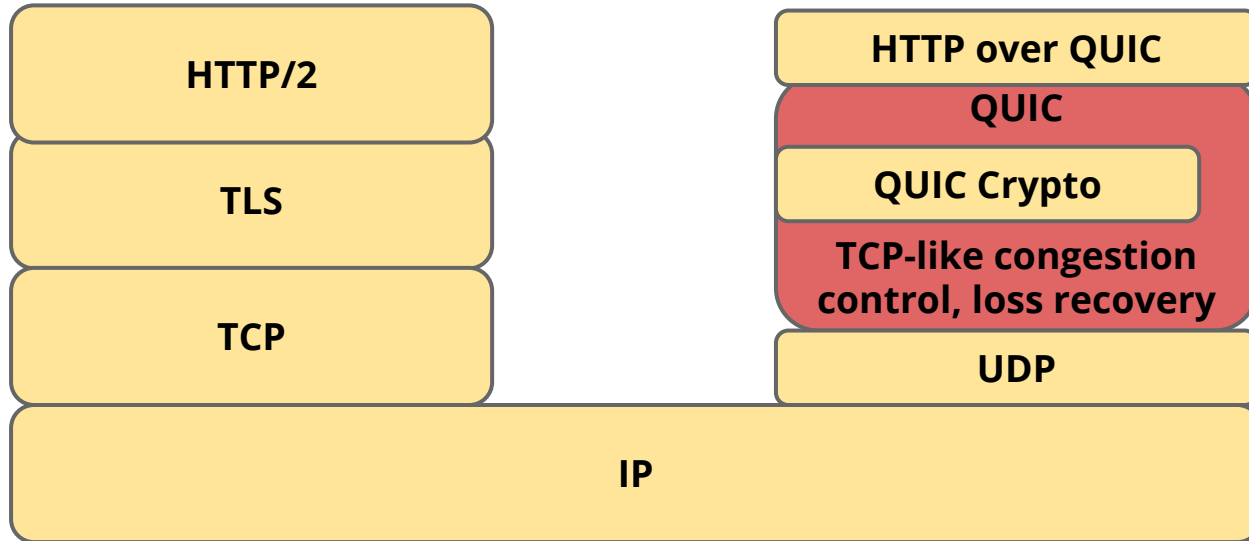
# How does HTTP over QUIC work?

- Connection setup... the QUIC way
  - 0 round-trips to a known server (common)
  - 1 round-trip if crypto keys are not new
  - 2 round-trips if QUIC version negotiation needed
  - (I haven't forgotten about TFO and TLS 1.3)
- After setup, HTTP requests/responses flow over connection

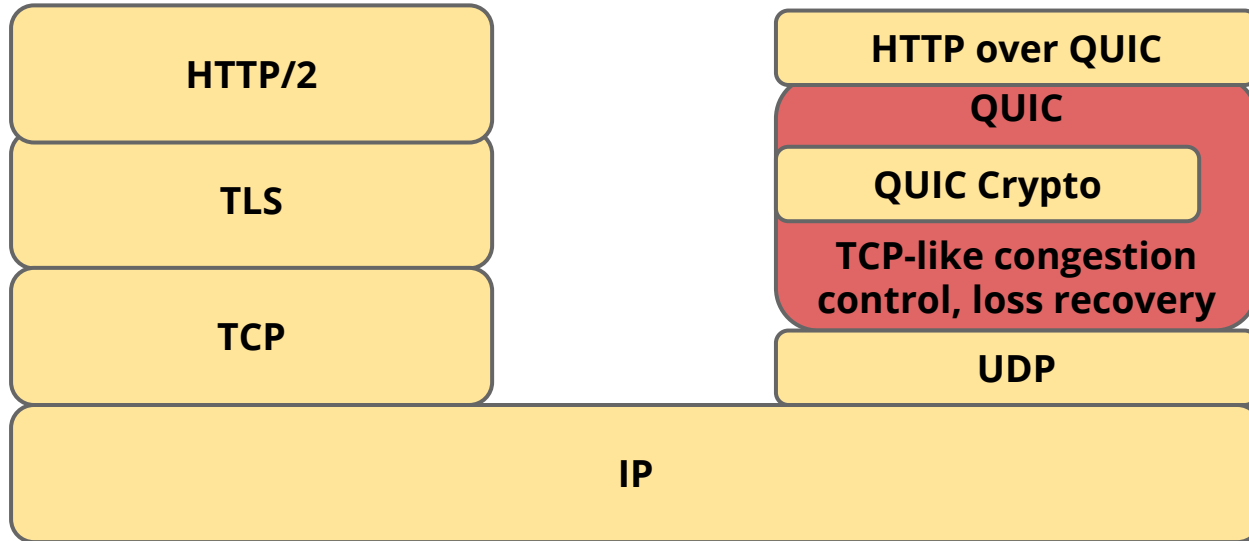
# What's HTTP over QUIC?



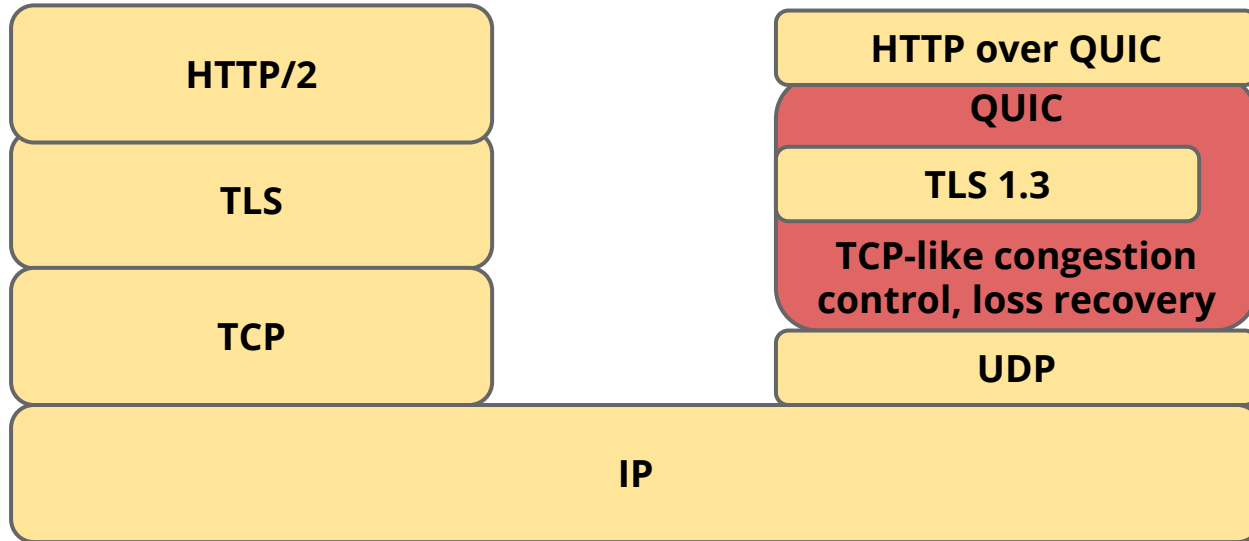
# Old Google QUIC



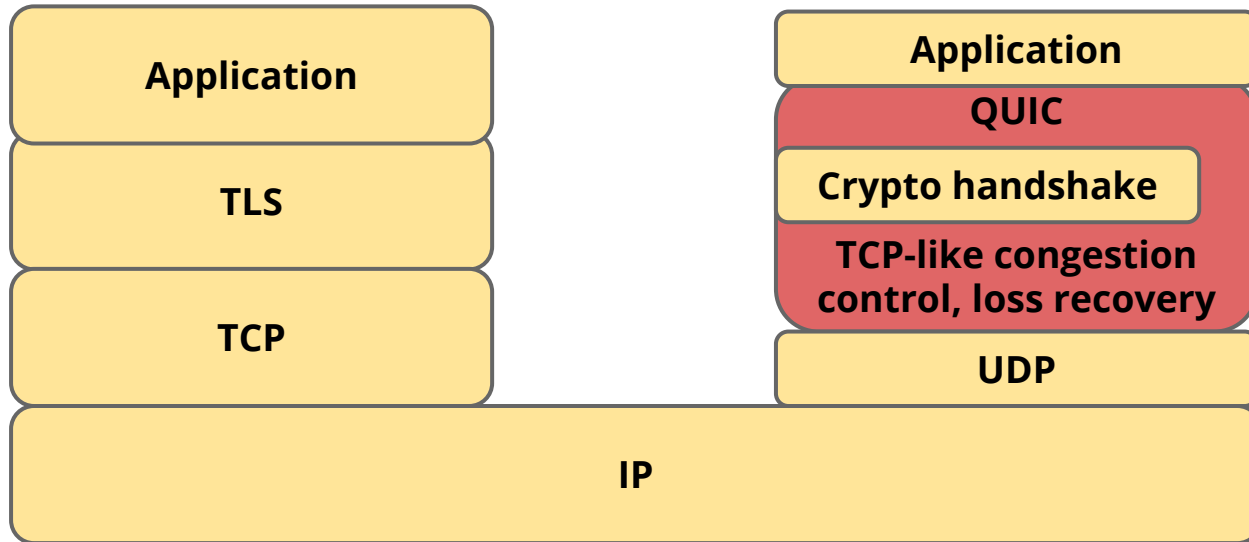
# QUIC working group



# QUIC working group



# An integrated, modularized protocol



**Hang on ... some of this sounds familiar**

**Yes!**

**We're replaying hits from the 1990s and 2000s  
(and adding some new things)**



# Hang on ... some of this sounds familiar

## **TLS 1.3**

Ongoing QUIC work uses TLS 1.3

## **TCP Fast Open (remember T/TCP?)**

Needs support in client-OS and middleboxes

Limited to one packet

## **SCTP, SST, TCP Session, ...**

Shared ideas, but many subtle differences

We're happy to steal ideas!

# QUIC Design Aspirations

- Deployability and evolvability
- Low latency connection establishment
- Multistreaming
- Better loss recovery and flexible congestion control
- Resilience to NAT-rebinding (Connection IDs vs. 4-tuple)
- Multipath for resilience and load sharing

# Deployability and Evolvability

## Uses UDP as the substrate

- enables deployment through middleboxes
- allows userspace implementation

## Version negotiation

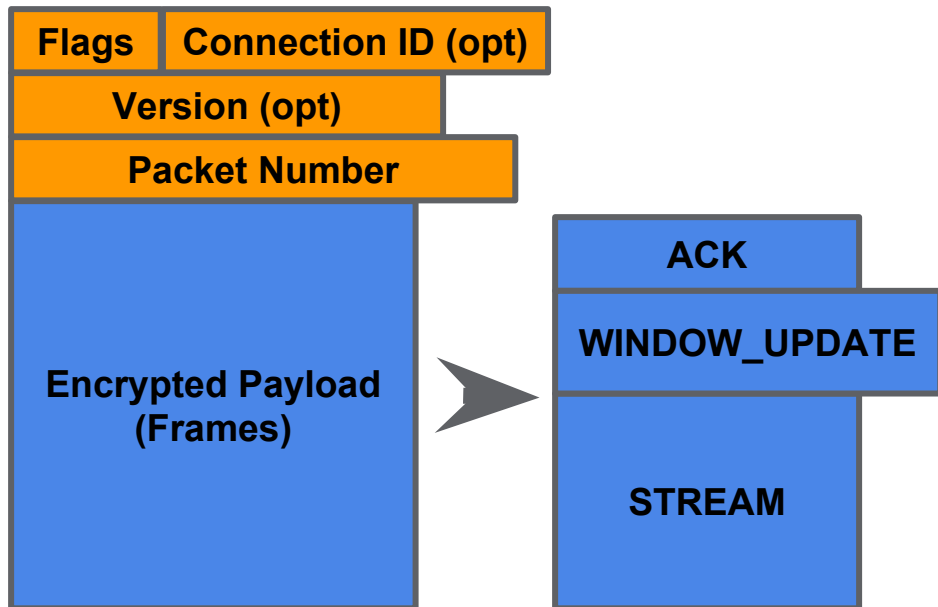
- enables protocol wire format evolution

## Fully authenticated and mostly encrypted headers

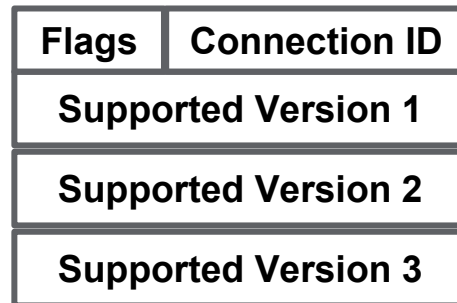
- avoids network ossification
- befuddles network operators :-)

# QUIC packets (previous)

## Regular Packets



## Version Negotiation Packet (Unencrypted)

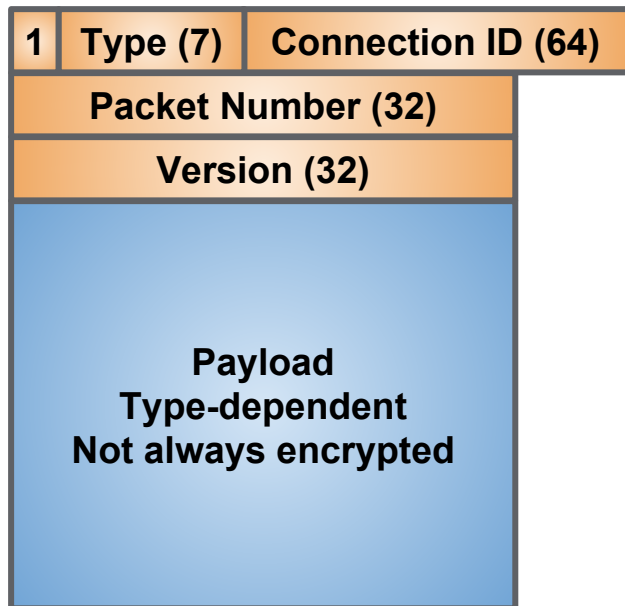


## Public Reset Packet (Unencrypted)

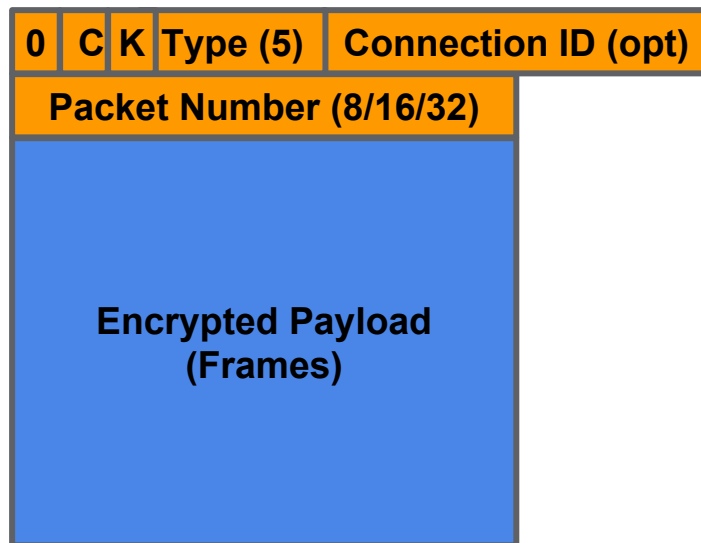


# QUIC packets (proposed)

## Long Header Packets



## Short Header Packets (optimized for packets encrypted with TLS 1-RTT key)



# Congestion Control & Loss Recovery

**QUIC builds on decades of experience with TCP**

**Incorporates TCP best practices**

TCP-like congestion control (NewReno, Cubic), FACK, TLP, F-RTO, Early Retransmit, ...

**Richer signaling than TCP**

# Richer Signaling Than TCP

**Retransmitted packets consume new sequence number**

no retransmission ambiguity

prevents loss of retransmission from causing RTO

**More verbose ACK**

TCP supports up to 3 SACK ranges

QUIC supports up to 256 ACK ranges

explicit packet receive times

enables ACK decimation

# What's the QUIC wg up to?

## Turning an amateur protocol into a professional one

A QUIC makeover

## Figuring out how to

- map HTTP cleanly to QUIC
- use TLS 1.3 with QUIC
- resolve open questions in QUIC
- make QUIC work for non-HTTP apps



## Is this just Google's QUIC?

No.

Google's QUIC was an experiment

QUIC wg uses the experiment as a starting point

Already moved miles away from experiment

A great example of running code informing protocol design.

# QUIC Implementations

**Chromium (open source)**

<https://cs.chromium.org/chromium/src/net/quic/>

**quic-go (open source implementation in Go)**

<https://github.com/lucas-clemente/quic-go>

# Debugging Tools: Wireshark

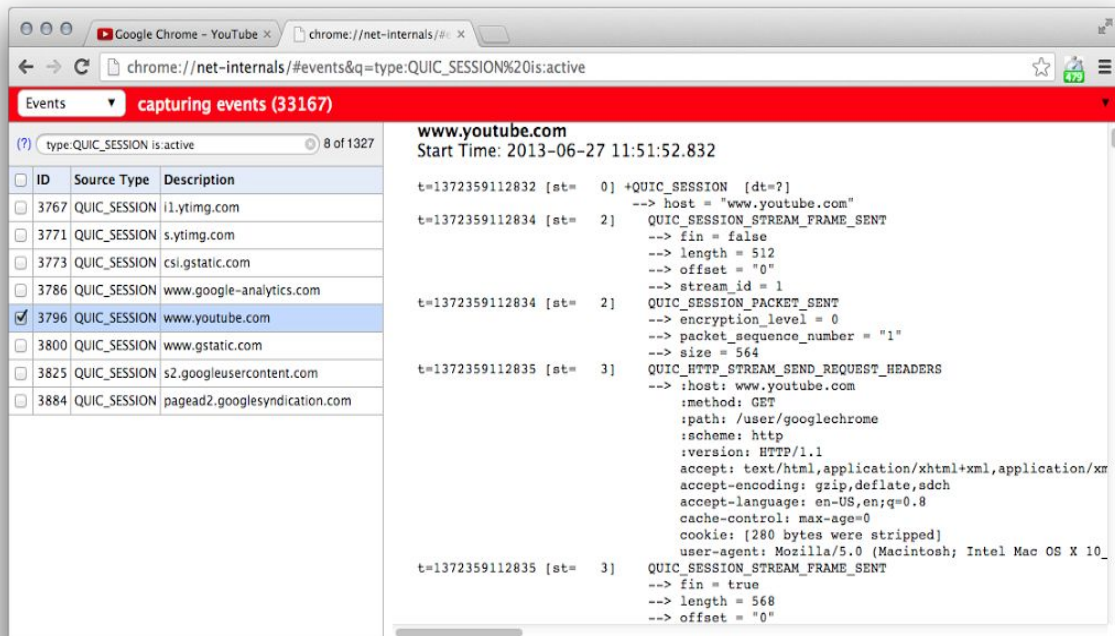
No.	Time	Source	Destination	Protoc	Length	Info
985	14.027869000	173.194.46.73	10.1.10.14	QUIC	1392	CID: 3182875774876983667, Seq: 1
986	14.028834000	10.1.10.14	173.194.46.73	QUIC	1392	CID: 3182875774876983667, Seq: 2
989	14.065914000	173.194.46.73	10.1.10.14	QUIC	1392	CID: 3182875774876983667, Seq: 2
990	14.066812000	10.1.10.14	173.194.46.73	QUIC	79	CID: 3182875774876983667, Seq: 3
991	14.194009000	10.1.10.14	173.194.46.73	QUIC	1392	CID: 3182875774876983667, Seq: 4
992	14.194164000	10.1.10.14	173.194.46.73	QUIC	350	CID: 3182875774876983667, Seq: 5
993	14.231536000	173.194.46.73	10.1.10.14	QUIC	85	CID: 3182875774876983667, Seq: 3
994	14.258228000	173.194.46.73	10.1.10.14	QUIC	353	CID: 3182875774876983667, Seq: 4
995	14.268285000	2601:6:2c01:9300:69a8:92607:f8b0:4004:a::12	10.1.10.14	QUIC	1412	CID: 2735399198252988334, Seq: 1
997	14.270807000	10.1.10.14	216.58.216.238	QUIC	1392	CID: 2060901289831796684, Seq: 1
998	14.273189000	10.1.10.14	173.194.46.76	QUIC	1392	CID: 16164325528471686122, Seq: 1
999	14.277601000	10.1.10.14	173.194.46.73	QUIC	1392	CID: 9176532438181928584, Seq: 1
1000	14.278560000	10.1.10.14	173.194.46.73	QUIC	1392	CID: 9176532438181928584, Seq: 2
1001	14.278618000	10.1.10.14	173.194.46.73	QUIC	515	CID: 9176532438181928584, Seq: 3
1002	14.284072000	10.1.10.14	173.194.46.73	QUIC	82	CID: 3182875774876983667, Seq: 6
1003	14.295209000	2607:f8b0:4004:a::12	2601:6:2c01:9300:69a8	QUIC	1412	CID: 2735399198252988334, Seq: 1
1004	14.296658000	2601:6:2c01:9300:69a8:92607:f8b0:4004:a::12	10.1.10.14	QUIC	99	CID: 2735399198252988334, Seq: 2
1005	14.309132000	216.58.216.238	10.1.10.14	QUIC	1392	CID: 2060901289831796684, Seq: 1
1006	14.312428000	173.194.46.76	10.1.10.14	QUIC	1392	CID: 16164325528471686122, Seq: 1

Filter: Expression... Clear Apply Save

Frame 981: 1392 bytes on wire (11136 bits), 1392 bytes captured (11136 bits) on interface 0 (outbound)  
Ethernet II, Src: Apple\_bc:da:74 (78:31:c1:bc:da:74), Dst: Netgear\_bf:79:04 (c4:04:15:bf:79:04)  
Internet Protocol Version 4, Src: 10.1.10.14 (10.1.10.14), Dst: 173.194.46.73 (173.194.46.73)  
User Datagram Protocol, Src Port: 51863 (51863), Dst Port: 80 (80)  
QUIC (Quick UDP Internet Connections)  
Public Flags: 0x0d  
CID: 3182875774876983667  
Version: 0024  
Sequence: 1  
Payload: 9f8da5bbb0e0724d965b22dc01a001000443484c4f130000...

# Debugging Tools: Chrome

chrome://net-internals  
(demo if time permits)



The screenshot shows the Chrome DevTools net-internals page with the 'Events' tab selected. The search filter is 'type:QUIC\_SESSION is:active' and 8 of 1327 events are shown. The selected event (ID 3796) is for 'www.youtube.com' and shows the following details:

```
www.youtube.com
Start Time: 2013-06-27 11:51:52.832

t=1372359112832 [st= 0] +QUIC_SESSION [dt=?]
--> host = "www.youtube.com"
t=1372359112834 [st= 2] QUIC_SESSION_STREAM_FRAME_SENT
--> fin = false
--> length = 512
--> offset = "0"
--> stream_id = 1
t=1372359112834 [st= 2] QUIC_SESSION_PACKET_SENT
--> encryption_level = 0
--> packet_sequence_number = "1"
--> size = 564
t=1372359112835 [st= 3] QUIC_HTTP_STREAM_SEND_REQUEST_HEADERS
--> :host: www.youtube.com
:method: GET
:path: /user/googlechrome
:scheme: http
:version: HTTP/1.1
accept: text/html,application/xhtml+xml,application/xml;
accept-encoding: gzip,deflate,sdch
accept-language: en-US,en;q=0.8
cache-control: max-age=0
cookie: [280 bytes were stripped]
user-agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_
t=1372359112835 [st= 3] QUIC_SESSION_STREAM_FRAME_SENT
--> fin = true
--> length = 568
--> offset = "0"
```