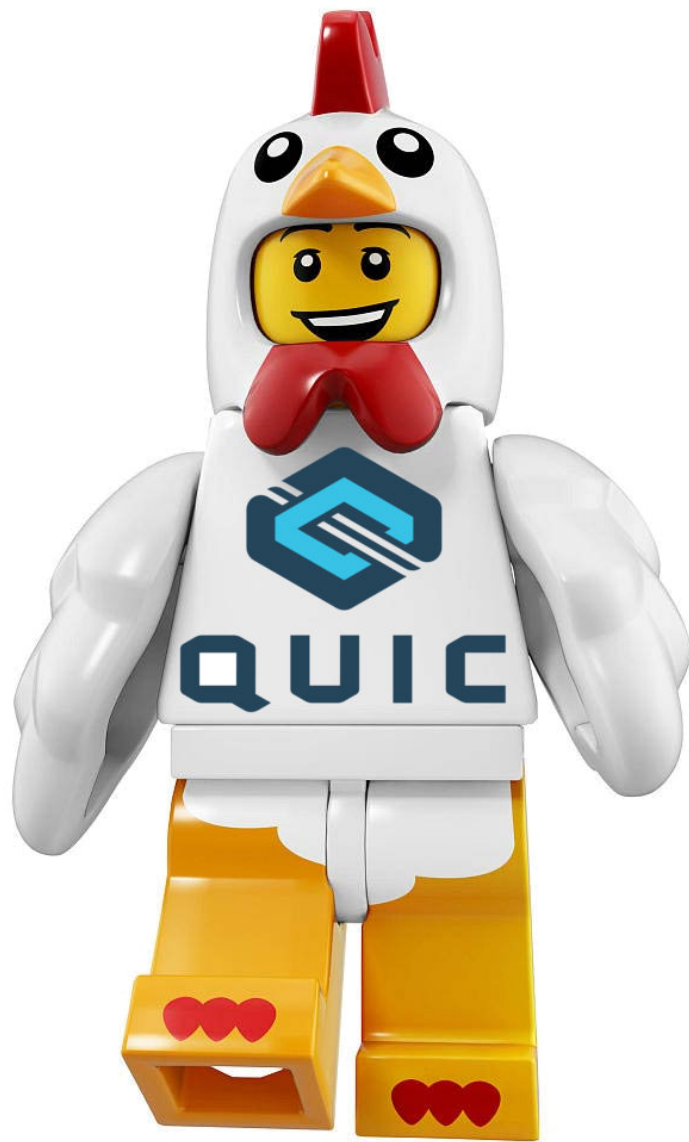




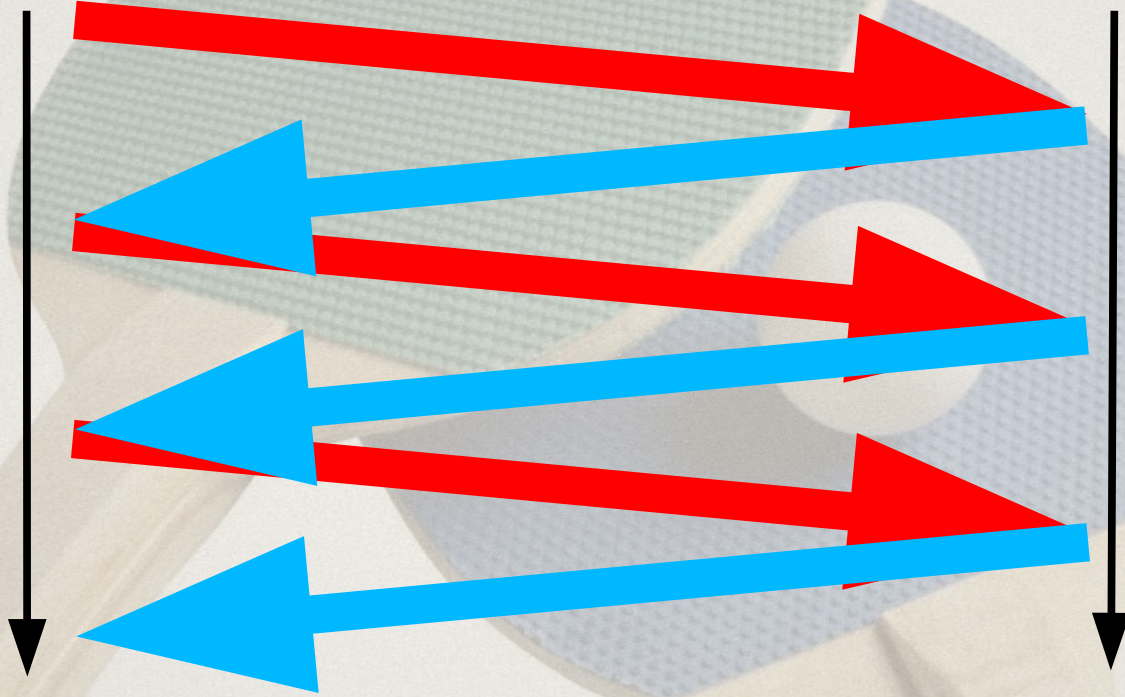
# Web Transport is HTTP

## HTTP/1

## HTTP/2

## QUIC

# Roundtrip Bonanza

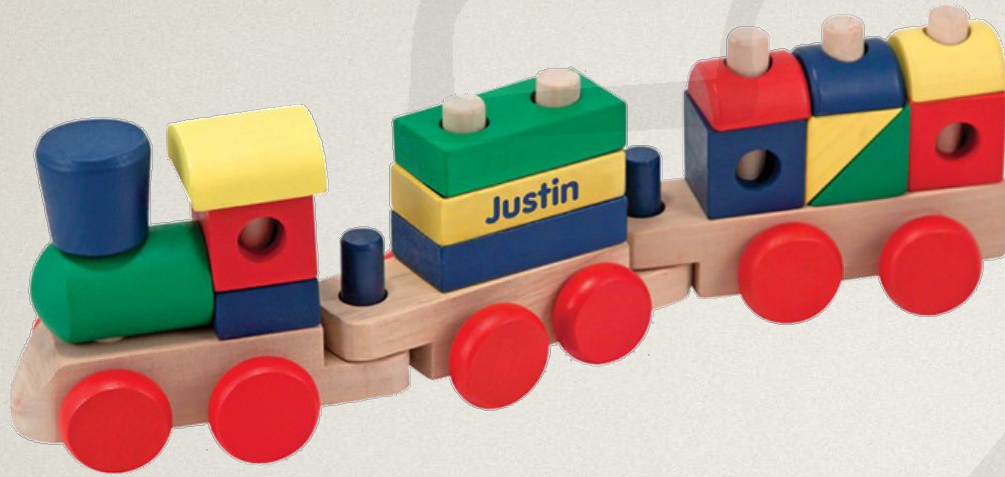




HTTP/2

**Maintain HTTP semantics,  
change how it is  
transported**

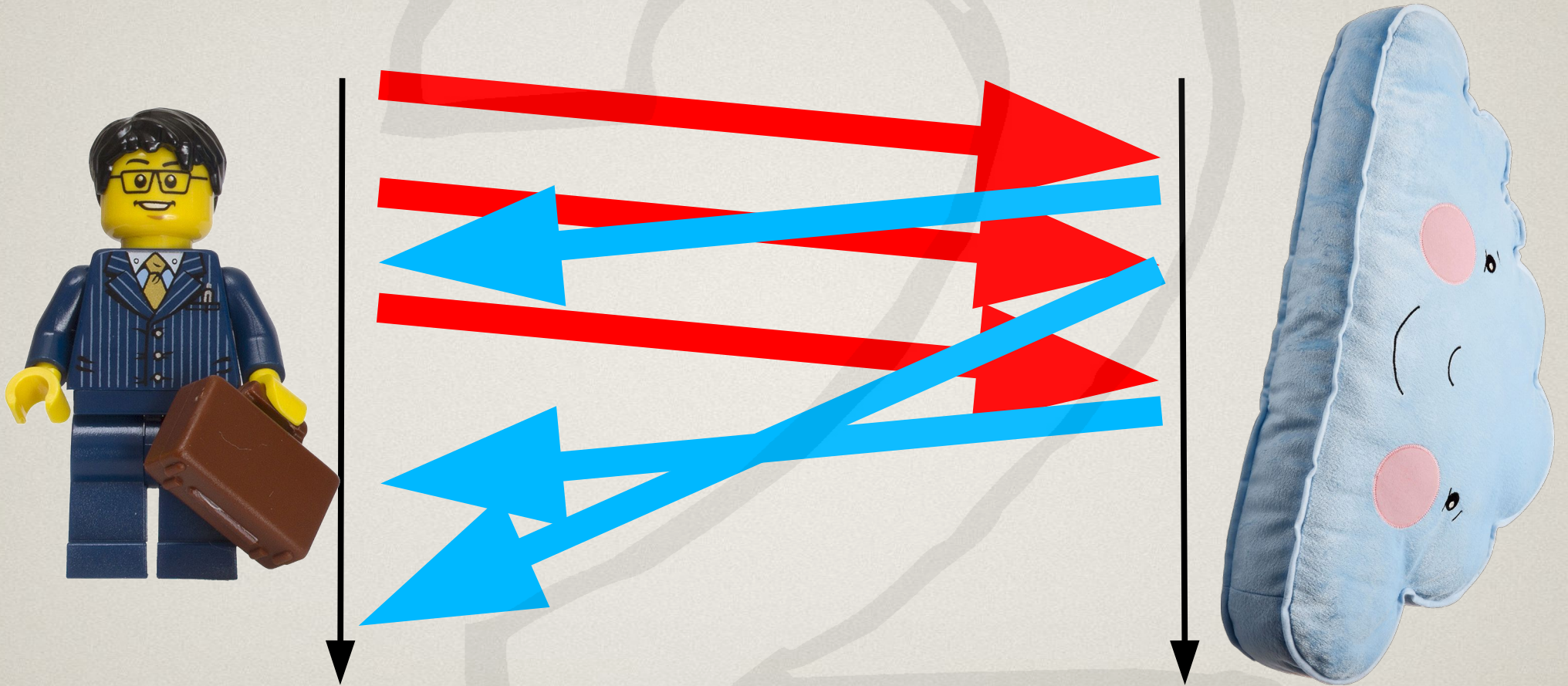
# Multiplexed (1/2)



## Multiplexed (2/2)

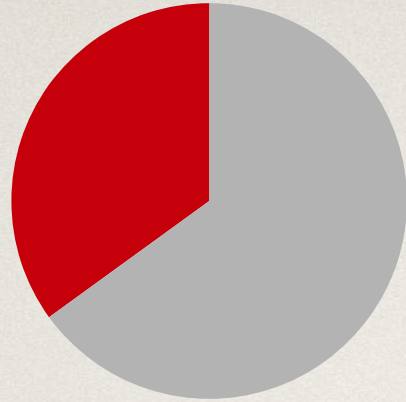


# Better utilization of bandwidth



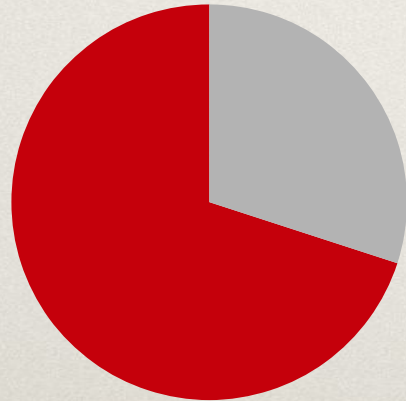
# Firefox stats – March 2018

HTTP/2:

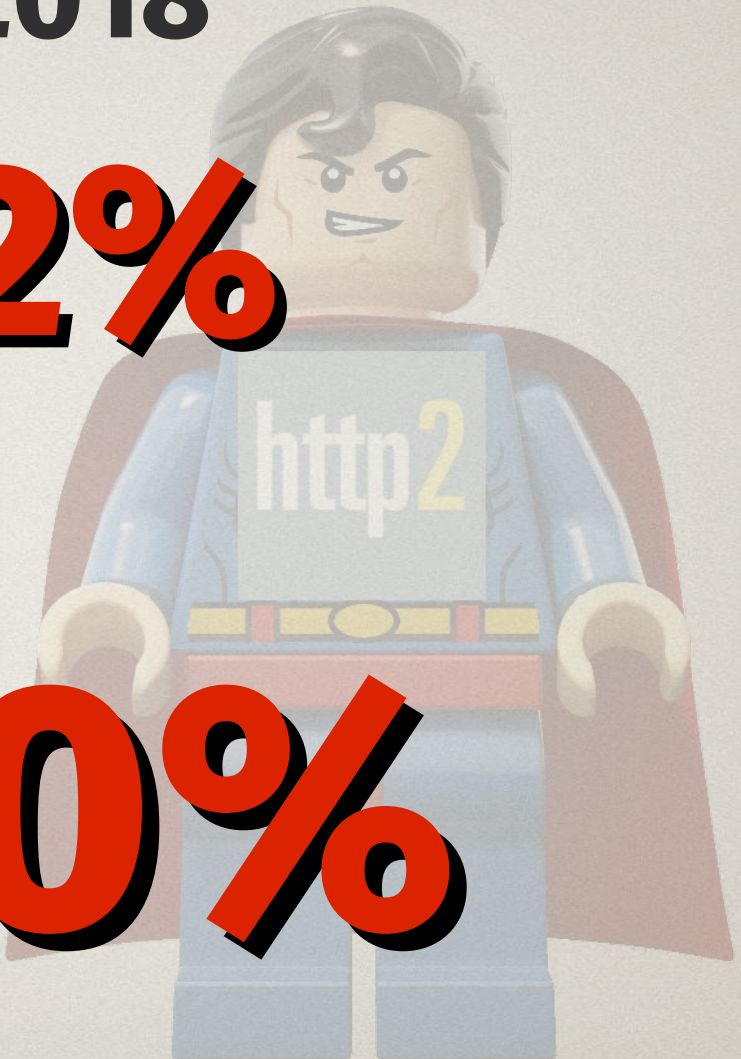


**42%**

Share of HTTPS:



**70%**



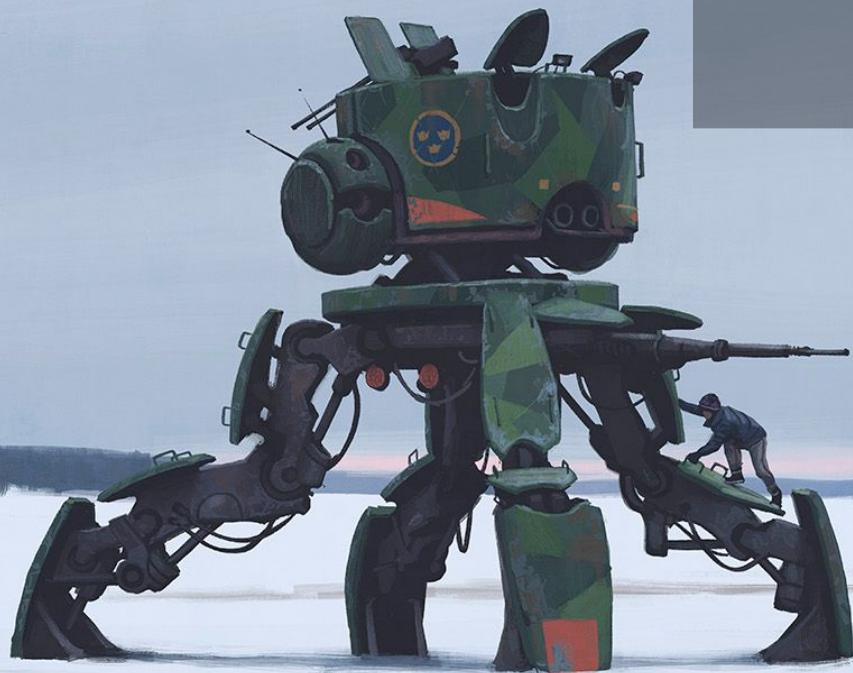
**Internetz – March 2018**

**24%** of top 10 million

*Doubled last 12 months*

**38%** of top 1000

# Winners and losers



# The remote corners of Internet

| Percentile | Desktop | Mobile |
|------------|---------|--------|
| 5          | 1       | 11     |
| 25         | 20      | 44     |
| 50         | 79      | 94     |
| 75         | 194     | 184    |
| 95         | 800     | 913    |

Milliseconds RTT

# The remote corners of Internet

| Percentile | Desktop | Mobile |
|------------|---------|--------|
| 5          | 1       | 11     |
| 25         | 20      | 44     |
| 50         | 79      | 94     |
| 75         | 194     | 184    |
| 95         | 800     | 913    |

Milliseconds RTT

# Queuing time h1 vs h2

(Time waiting internally to send off a HTTP request)

| Percentile | HTTP 1  | HTTP 2 |
|------------|---------|--------|
| 80         | 100 ms  | 2 ms   |
| 95         | 2000 ms | 16 ms  |

>100ms: H1 **20%**, H2 **3%**

# 0% packet loss



Image and data by Hooman Beheshti, Fastly

# 2% packet loss

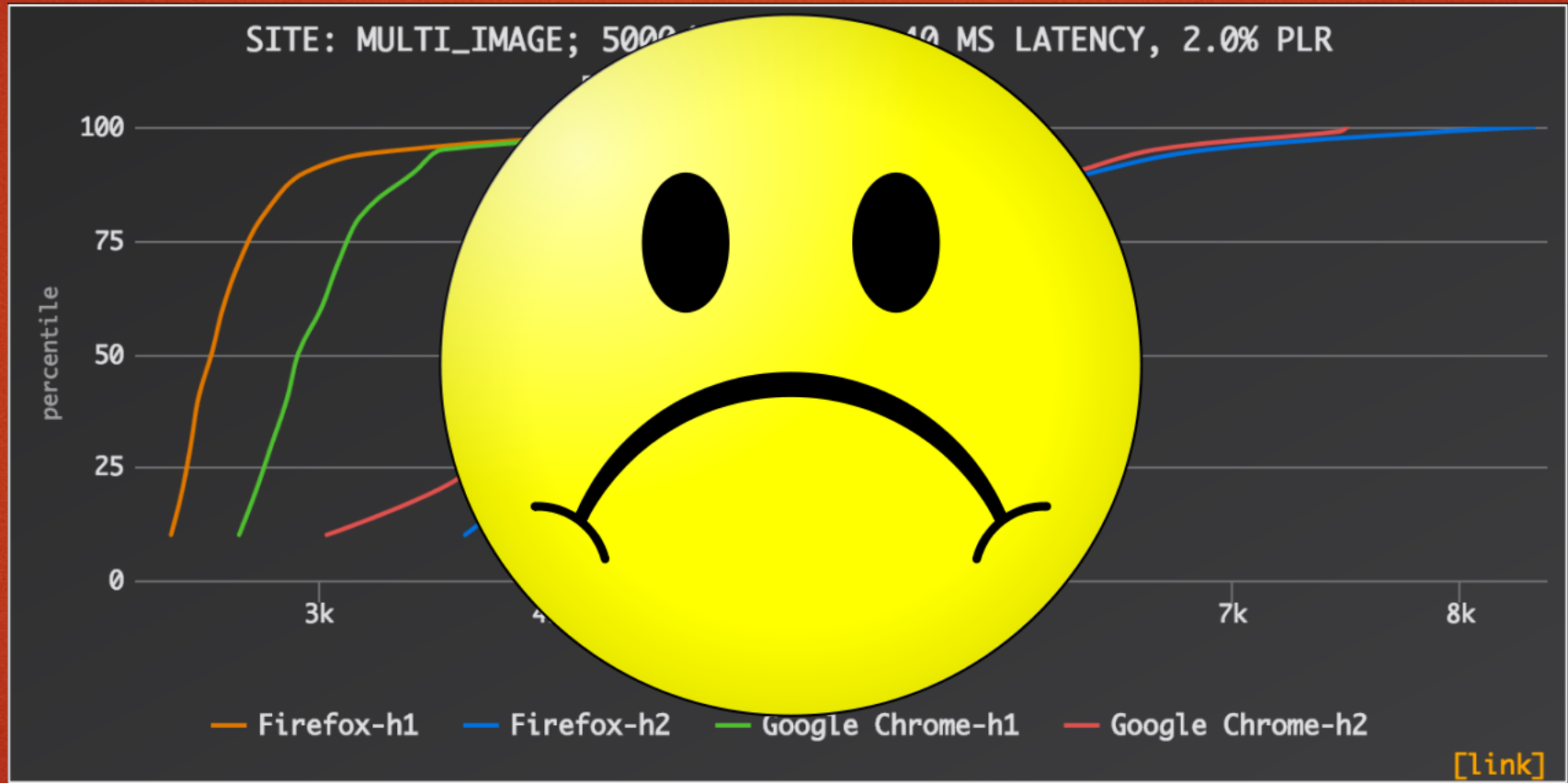


Image and data by Hooman Beheshti, Fastly

**Packet loss, hey?**

A single dropped  
packet blocks all  
streams

# Packet loss, hey?

TCP

TCP

TCP

TCP

IP

IP

IP

IP

IP

IP

# Packet loss, hey?

TLS

TLS

TLS

TCP

TCP

TCP

TCP

IP

IP

IP

IP

IP

IP

# Packet loss, hey?

HTTP/2  
frame

HTTP/2  
frame

HTTP/2  
frame

HTTP/2  
frame

HTTP/2  
frame

TLS

TLS

TLS

TCP

TCP

TCP

TCP

IP

IP

IP

IP

IP

IP

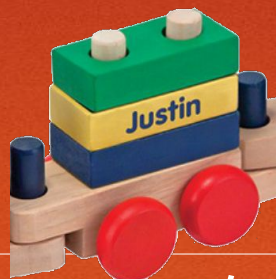
# Packet loss, hey?



HTTP/2  
frame



HTTP/2  
frame



HTTP/2  
frame



HTTP/2  
frame



HTTP/2  
frame

TLS

TLS

TLS

TCP

TCP

TCP

TCP

IP

IP

IP

IP

IP

IP

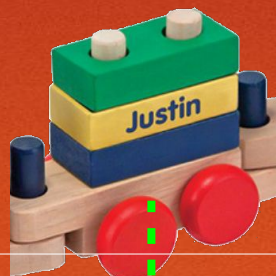
# Packet loss, hey?



HTTP/2  
frame



HTTP/2  
frame



HTTP/2  
frame



HTTP/2  
frame



HTTP/2  
frame

TLS

TLS

TLS

TCP

TCP

TCP

TCP

IP

IP

IP

IP

IP

IP



**Fixing TCP head of line blocking**

# A non-blocking TCP + TLS + HTTP/2

independent packets

... that are stream aware

Needs retransmissions/ACKs

New protocol?

Fixing TCP takes decades – if even doable



# QUIC

over UDP and end-to-end crypto

no TCP head of line blocking

Independent streams

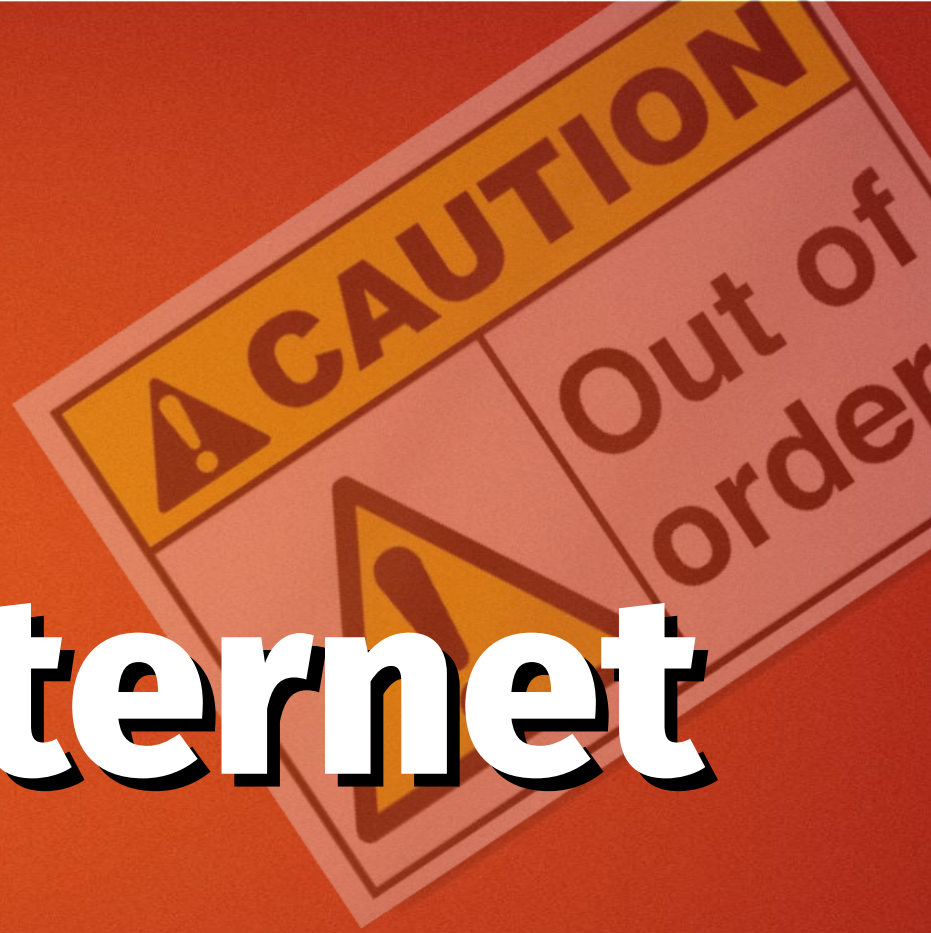
0RTT

“TCP improvements” faster



**Google-QUIC**

**7% of Internet**



QUIC

**Maintain HTTP semantics,  
change how it is  
transported**

# Packet loss, hey?



h2



h2



h2



h2



h2



h2

TLS

TLS

TLS

TLS

TLS

TLS

quic

quic

quic

quic

quic

quic

UDP

UDP

UDP

UDP

UDP

UDP

IP

IP

IP

IP

IP

IP

# Packet loss, hey?



h2



h2



h2



h2



h2



h2

TLS

TLS

TLS

TLS

TLS

TLS

quic

quic

quic

quic

quic

quic

UDP

UDP

UDP

UDP

UDP

UDP

IP

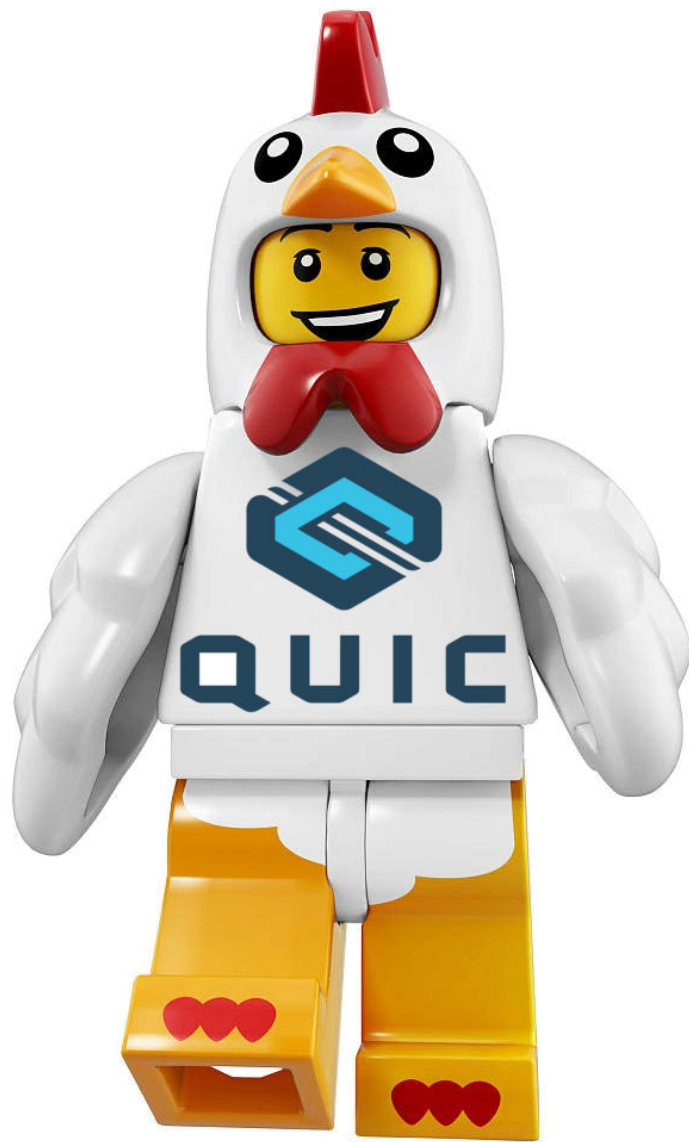
IP

IP

IP

IP

IP



# The IETF QUIC wg

Started in 2016

Massive interest

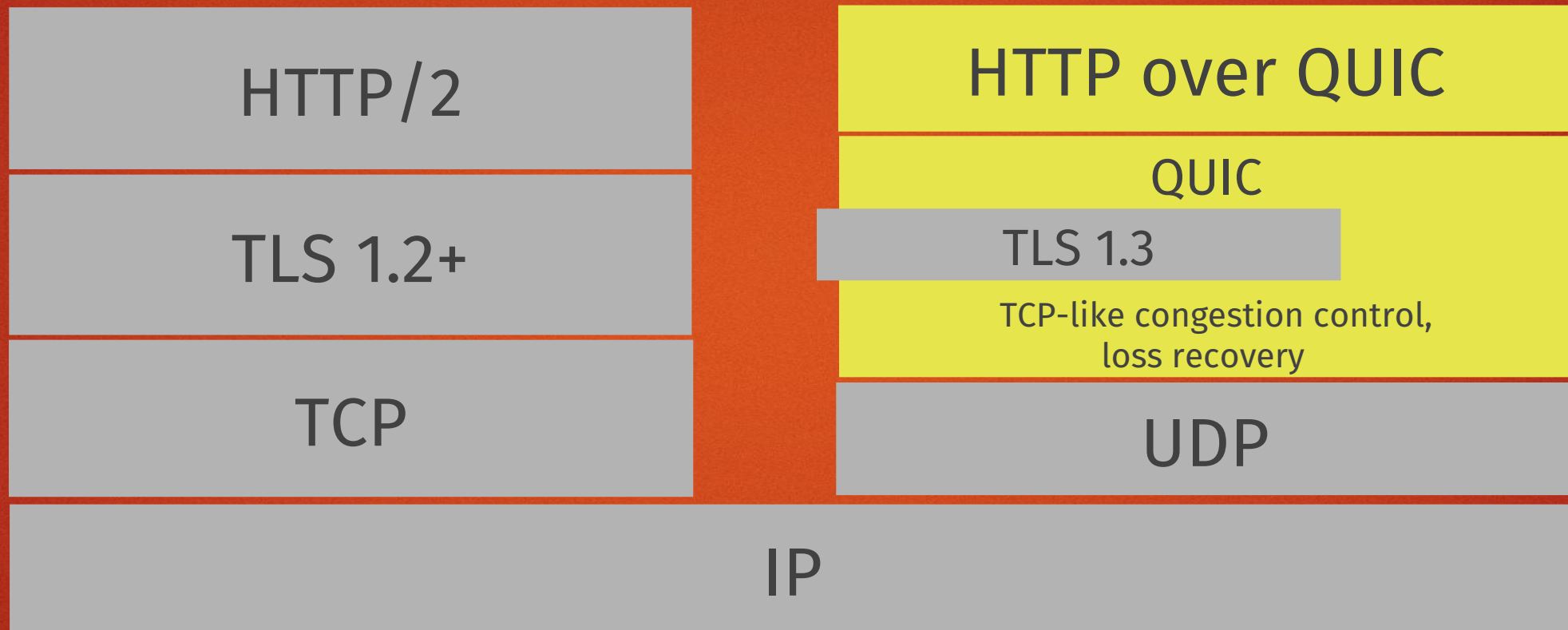
More than “h2-like”

Fifth interim in Stockholm in June ‘18

IETF-QUIC vs Google-QUIC

*Done* by the end of 2018!

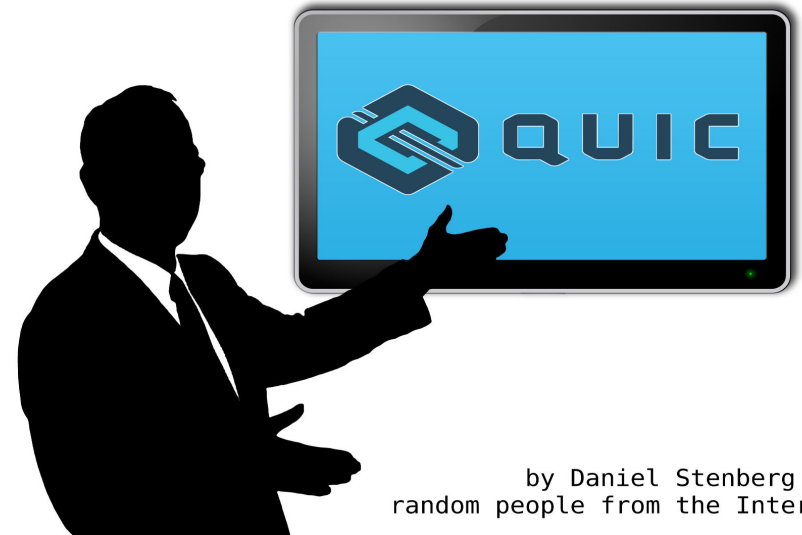
# The IETF-QUIC stack



# This is QUIC

<https://daniel.haxx.se/this-is-quic/>

# This is QUIC



by Daniel Stenberg and  
random people from the Internet

# QUIC in curl (1/2)

Not started yet

Base on nghq (based on ngtcp2) ?

Similiar integration as HTTP/2

Start out with “known QUIC peer”; add alt-svc later

TLS integration might get quirky; start simple

Test server in nghq?

# QUIC in curl (2/2)

Get started ASAP - who's in?

Initial thoughts by the next QUIC Interim in Stockholm (June 2018)