A QUIC Tutorial

SIGCOMM 2020

Jana Iyengar
Fastly

Ian Swett
Google

Robin Marx
Hasselt University

Logistics

Zoom:

raise your hand for questions here

one of us will monitor

Slack:

open for clarification questions, discussion

prefer over zoom chat

Questions and interruptions welcome!

this is only as useful to you as you make it

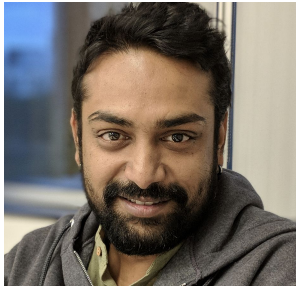
Plan

#	Start - End	Topic
1	1:40 - 1:58	QUIC's intellectual heritage
2	2:00 - 2:18	QUIC handshake, headers, connection migration
3	2:20 - 2:38	Wireshark demo and tutorial
4	2:40 - 2:58	QUIC streams, flow control, frames, packetization
5	**3:00 - 3:18**	**BREAK**
6	3:20 - 3:38	qlog and qvis demo and tutorial
7	3:40 - 3:58	QUIC loss detection and congestion control: how different from TCP?
8	4:00 - 4:18	Build your own congestion controller. Code walkthrough: quicly and quiche
9	4:20 - 4:38	Extending QUIC: transport parameters and extensions (Ack Frequency)
10	4:40 - 5:00	Open Discussion, Q & A

Who are we?

Jana Iyengar

Distinguished Engineer, Office of the CTO, Fastly
Editor of IETF QUIC specs
Chair of IRTF Internet Congestion Control research group
Working on transport since ~2000, QUIC since 2013



Ian Swett

Staff Engineer and QUIC Tech Lead, Google
Editor of IETF QUIC specs
Working on QUIC since 2012
TL for QUIC BBR and BBRv2

Robin Marx

PhD student, Hasselt University in Belgium
Focus on HTTP/2, QUIC, and HTTP/3 performance
creates qlog and qvis debugging tools
Co-founder of LuGus Studios; multiplayer game programmer

Plan

#	Start - End	Topic
1	1:40 - 1:58	QUIC’s intellectual heritage
2	2:00 - 2:18	QUIC handshake, headers, connection migration
3	2:20 - 2:38	Wireshark demo and tutorial
4	2:40 - 2:58	QUIC streams, flow control, frames, packetization
5	**3:00 - 3:18**	**BREAK**
6	3:20 - 3:38	qlog and qvis demo and tutorial
7	3:40 - 3:58	QUIC loss detection and congestion control: how different from TCP?
8	4:00 - 4:18	Build your own congestion controller. Code walkthrough: quicly and quiche
9	4:20 - 4:38	Extending QUIC: transport parameters and extensions (Ack Frequency)
10	4:40 - 5:00	Open Discussion, Q & A

What is QUIC?

A new transport protocol

built for needs of today's Internet and the modern web
not what TCP was built for

UDP-based, because UDP gets through most networks

QUIC re-creates TCP services from scratch
(loss recovery, congestion control, flow control, etc.)

Has encryption baked in, data/metadata are protected

combines transport and crypto handshakes for latency
uses TLS/1.3 for key negotiation

What is HTTP/3?

HTTP/3 is HTTP over QUIC

Feature-parity with HTTP/2

request multiplexing, header compression, push

… except for priorities

which is being dropped in HTTP/2 by HTTP working group

a common scheme is being devised for HTTP/2 and /3

A ~~QUIC~~ GQUIC history

Protocol for HTTPS transport, deployed at Google starting 2014

- Between Google services and Chrome / mobile apps
- 35% of Google's egress traffic (7% of Internet) in 2017

IETF QUIC working group formed in Oct 2016

- Modularize and standardize QUIC

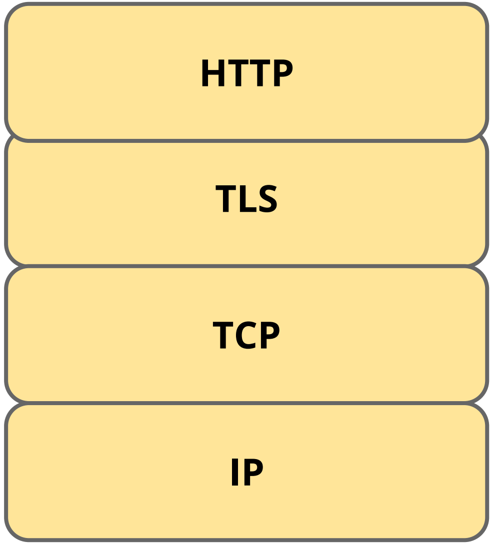
A QUIC history

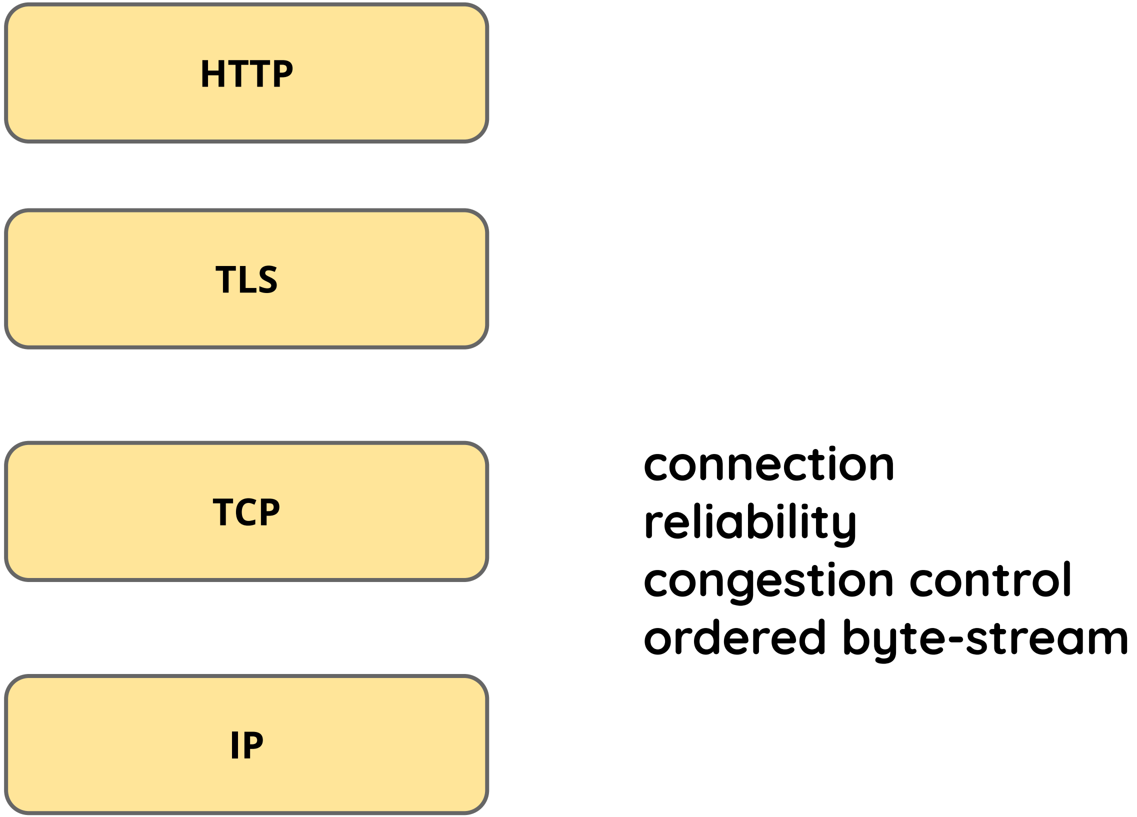
QUIC has matured significantly at the IETF

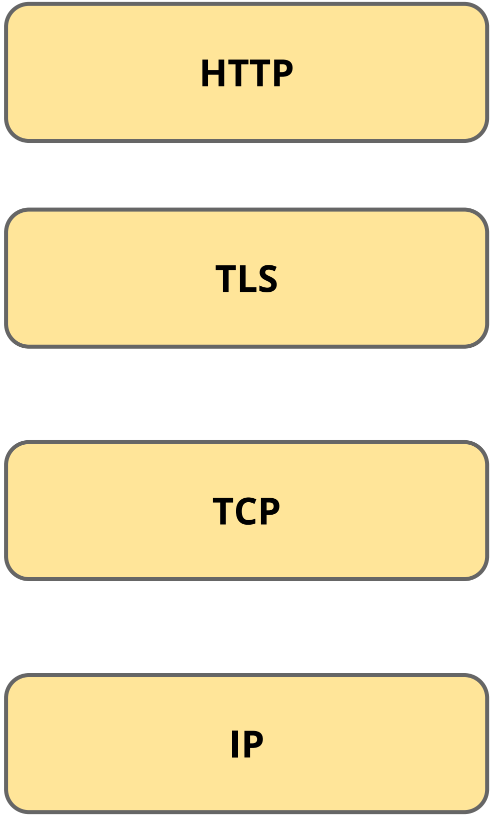
- Use of TLS 1.3
- Overhaul of the handshake
- New packet headers and structure
- Packet number encryption
- Connection IDs
- Unidirectional and bidirectional streams
- HTTP mapping

- Operator concerns

QUIC comes in a long line of work on web transport

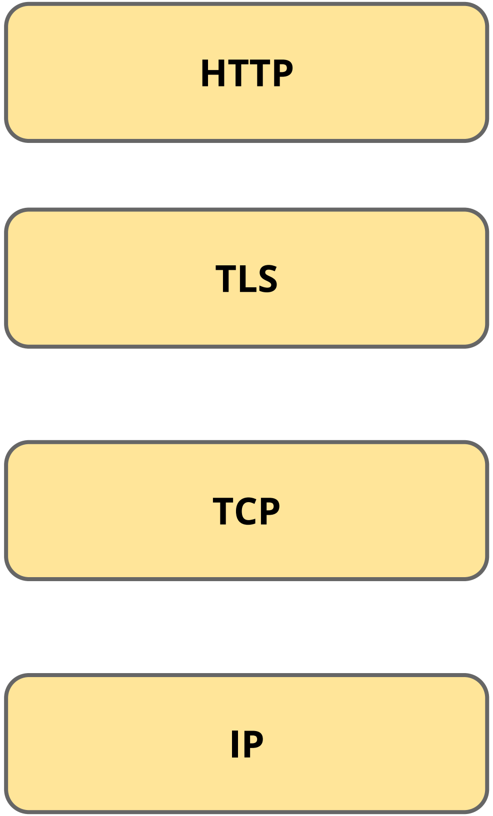

HTTP
TLS
TCP
IP


HTTP
TLS
TCP
IP
connection
reliability
congestion control
ordered byte-stream



authentication
encryption

connection
reliability
congestion control
ordered byte-stream



many many objects
needs low latency

authentication
encryption

connection
reliability
congestion control
ordered byte-stream

The HTTP Story

HTTP/1.0

: independent file transfers
(open, write, close)

The HTTP Story

HTTP/1.0
HTTP/1.1

: connection persistence

: pipelining

The HTTP Story

HTTP/1.0
HTTP/1.1

Then around 1998 ...

HTTP-NG
Activity Statement

W3C's work on HTTP Next Generation (HTTP-NG) is being managed as part of W3C's Architecture Domain.

Activity statements provide a managerial overview of W3C's work in this area. They provide information about what W3C is actively doing in a particular area and how we believe this will benefit the Web community. You will also be able to find a list of accomplishments to date and a summary of where we are headed. The area overview is often a good source of more generic information about the area and the background reading pages can help set the scene and explain any technical concepts in preparation.

Introduction

The World Wide Web is a tremendous and growing success and HTTP has been at the core of this success as the primary substrate for exchanging information on the Web. However, HTTP/1.1 is becoming strained modularity wise as well as performance wise and those problems are to be addressed by HTTP-NG.

Modularity is an important kind of simplicity, and HTTP/1.x isn't very modular. If we look carefully at HTTP/1.x, we can see it addresses three layers of concerns, but in a way that does not cleanly separate those layers: message transport, general-purpose remote method invocation, and a particular set of methods historically focused on document processing (broadly construed to include things like forms processing and searching).

The lack of modularity makes the specification and evolution of HTTP more difficult than necessary and also causes problems for other applications. Applications are being layered on top of HTTP, and these applications are thus forced to include a lot of

MUX Overview

https://www.w3.org/Protocols/MUX/

MUX Overview

MUX is a session management protocol separating the underlying transport from the upper level application protocols. It provides a lightweight communication channel to the application layer by multiplexing data streams on top of a reliable stream oriented transport. By supporting coexistence of multiple application level protocols (e.g. HTTP and HTTP-NG), MUX will ease transitions to future Web protocols, and communications of client applets using private protocols with servers over the same connection as the HTTP conversation.

- *Why MUX?*
- *Working Drafts and Notes*
- *Related Protocols*

MUX is now part of the W3C HTTP-NG project where a Working Draft is being produced. Discussion of this draft takes place on the HTTP-NG Interest Group Mailing list.

@(#) $Id: Overview.html,v 1.37 2000/12/06 10:37:58 ylafon Exp $

Why MUX?

The Internet is suffering from the effects of the HTTP/1.0 protocol, which was designed without thorough understanding of the underlying TCP transport protocol. HTTP/1.0 opens a TCP connection for each URI retrieved (at a cost of both packets and round trip times (RTTs)), and then closes the connection. For small HTTP requests, these connections have poor performance due to TCP slow start as well as the round trips required to open and close each TCP connection.

HTTP/1.1 persistent connections and pipelining will reduce network traffic and the amount of TCP overhead caused by opening and closing TCP connections. However, the serialized behavior of HTTP/1.1 pipelining does not adequately support simultaneous rendering of inlined objects - part of most Web pages today; nor does it provide suitable fairness between protocol flows, or allow for graceful abortion of HTTP transactions without closing the TCP connection.

Current TCP implementations do not share congestion information across multiple simultaneous connections between two peers, which increases the overhead of opening new TCP connections. We expect that Transactional TCP and sharing of congestion information in TCP control blocks will improve TCP performance by using less RTTs, making it more suitable for HTTP transactions.

“[...] poor performance due to [...] round trips required to open and close each TCP connection”

“[...] does not adequately support simultaneous rendering of inlined objects”

“[...] nor does it provide suitable fairness between protocol flows”

“[...] or allow for graceful abortion of HTTP transactions without closing the TCP connection”

“[...] do not share congestion information across multiple simultaneous connections”

“[...] poor performa **handshake latency** *uired to open and close each TCP connectio*

“[...] does not adequately support simultaneous rendering of inlined objects”

“[...] nor does it provide suitable fairness between protocol flows”

“[...] or allow for graceful abortion of HTTP transactions without closing the TCP connection”

“[...] do not share congestion information across multiple simultaneous connections”

“[...] poor performa

handshake latency

uired to open and close each TCP connectio

“[...] does not adequat

parallelism

rendering of inlined objects”

“[...] nor does it provide suitable fairness between protocol flows”

“[...] or allow for graceful abortion of HTTP transactions without closing the TCP connection”

“[...] do not share congestion information across multiple simultaneous connections”

“[...] poor performa **handshake latency** *uired to open and close each TCP connectio*

“[...] does not adequat **parallelism** *rendering of inlined objects”*

“[...] nor does it provi **scheduling** *n protocol flows”*

“[...] or allow for graceful abortion of HTTP transactions without closing the TCP connection”

“[...] do not share congestion information across multiple simultaneous connections”

“[...] poor performa

handshake latency

uired to open and close each TCP connectio

“[...] does not adequat

parallelism

rendering of inlined objects”

“[...] nor does it provic

scheduling

n protocol flows”

“[...] or allow for gro

request cancellation

ictions without closing the TCP connection”

“[...] do not share congestion information across multiple simultaneous connections”

"[...] poor performa

handshake latency

uired to open and close
each TCP connectio

"[...] does not adequat

parallelism

rendering of inlined objects"

"[...] nor does it provic

scheduling

n protocol flows"

"[...] or allow for gro

request cancellation

ctions without closing
the TCP connection"

"[...] do not share

many congestion controllers

simultaneous
connections"

“[...] multiplexing multiple lightweight HTTP transactions

“[…] multiplexing multiple lightweight HTTP transactions

onto the same underlying transport connection

“[...] multiplexing multiple lightweight HTTP transactions

onto the same underlying transport connection

and deploying smart output buffer management”

The Transport Story

The Transport Story

T/TCP (1994)

RFC 1644 - T/TCP -- TCP Exte

https://tools.ietf.org/html/rfc1644

```
[Docs] [txt|pdf] [Tracker]

Obsoleted by: 6247                                            HISTORIC

Network Working Group                                        R. Braden
Request for Comments: 1644                                         ISI
Category: Experimental                                       July 1994

                T/TCP -- TCP Extensions for Transactions
                        Functional Specification

Status of this Memo

   This memo describes an Experimental Protocol for the Internet
   community, and requests discussion and suggestions for improvements.
   It does not specify an Internet Standard.  Distribution is unlimited.

Abstract

   This memo specifies T/TCP, an experimental TCP extension for
   efficient transaction-oriented (request/response) service.  This
   backwards-compatible extension could fill the gap between the current
   connection-oriented TCP and the datagram-based UDP.

   This work was supported in part by the National Science Foundation
   under Grant Number NCR-8922231.

Table of Contents

 1. INTRODUCTION ................................................  2
 2.  OVERVIEW ...................................................  3
```

The Transport Story

T/TCP (1994)
TCP Session (1998)

Venkata N. Padmanabhan's Ph

Not Secure | daedalus.cs.berkeley.edu/publications/padmanab-thesis/index.html

Addressing the Challenges of Web Data Transport

Venkata N. Padmanabhan

Doctor of Philosophy in Computer Science

University of California at Berkeley

September 1998

Abstract

In just a few years since its inception, the World Wide Web has grown to be the most dominant application in the Internet. In large measure, this rapid growth is due to the Web's convenient point-and-click interface and its appealing graphical content. Since Web browsing is an interactive activity, minimizing user-perceived latency is an important goal. However, layering Web data transport on top of the TCP protocol poses several challenges to achieving this goal.

First, the transmission of a Web page from a server to a client involves the transfer of multiple distinct components, each in itself of some value to the user. To minimize user-perceived latency, it is desirable to transfer the components concurrently. TCP provides an ordered byte-stream abstraction with no mechanism to demarcate sub-streams. If a separate TCP connection is used for each component, as with HTTP/1.0, uncoordinated competition among the connections could exacerbate congestion, packet loss, unfairness, and latency.

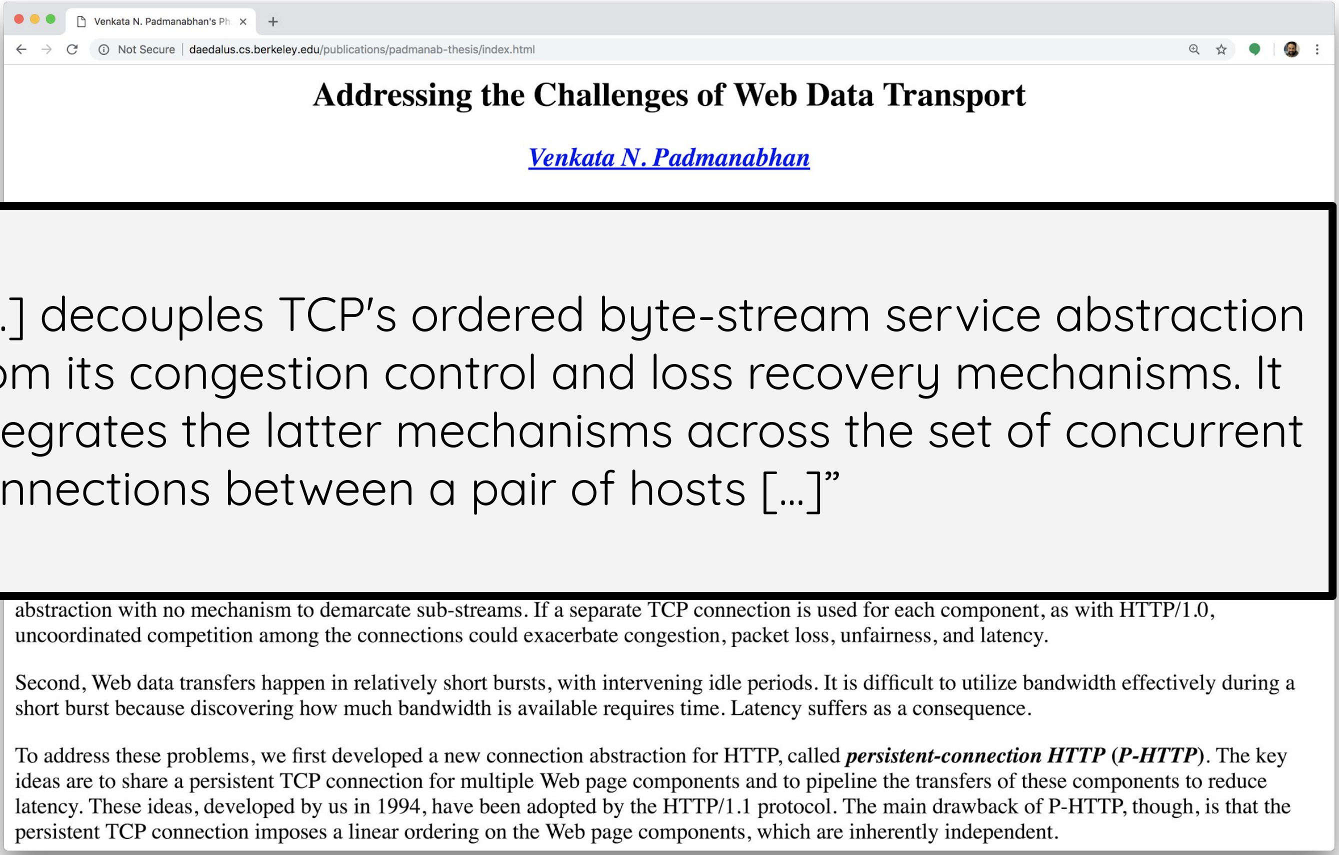
Second, Web data transfers happen in relatively short bursts, with intervening idle periods. It is difficult to utilize bandwidth effectively during a short burst because discovering how much bandwidth is available requires time. Latency suffers as a consequence.

To address these problems, we first developed a new connection abstraction for HTTP, called ***persistent-connection HTTP*** (***P-HTTP***). The key ideas are to share a persistent TCP connection for multiple Web page components and to pipeline the transfers of these components to reduce latency. These ideas, developed by us in 1994, have been adopted by the HTTP/1.1 protocol. The main drawback of P-HTTP, though, is that the persistent TCP connection imposes a linear ordering on the Web page components, which are inherently independent.

The Transport Story

T/TCP (1994)

TCP Session (1998)



Venkata N. Padmanabhan's Ph

Not Secure | daedalus.cs.berkeley.edu/publications/padmanab-thesis/index.html

Addressing the Challenges of Web Data Transport

[*Venkata N. Padmanabhan*]

abstraction with no mechanism to demarcate sub-streams. If a separate TCP connection is used for each component, as with HTTP/1.0, uncoordinated competition among the connections could exacerbate congestion, packet loss, unfairness, and latency.

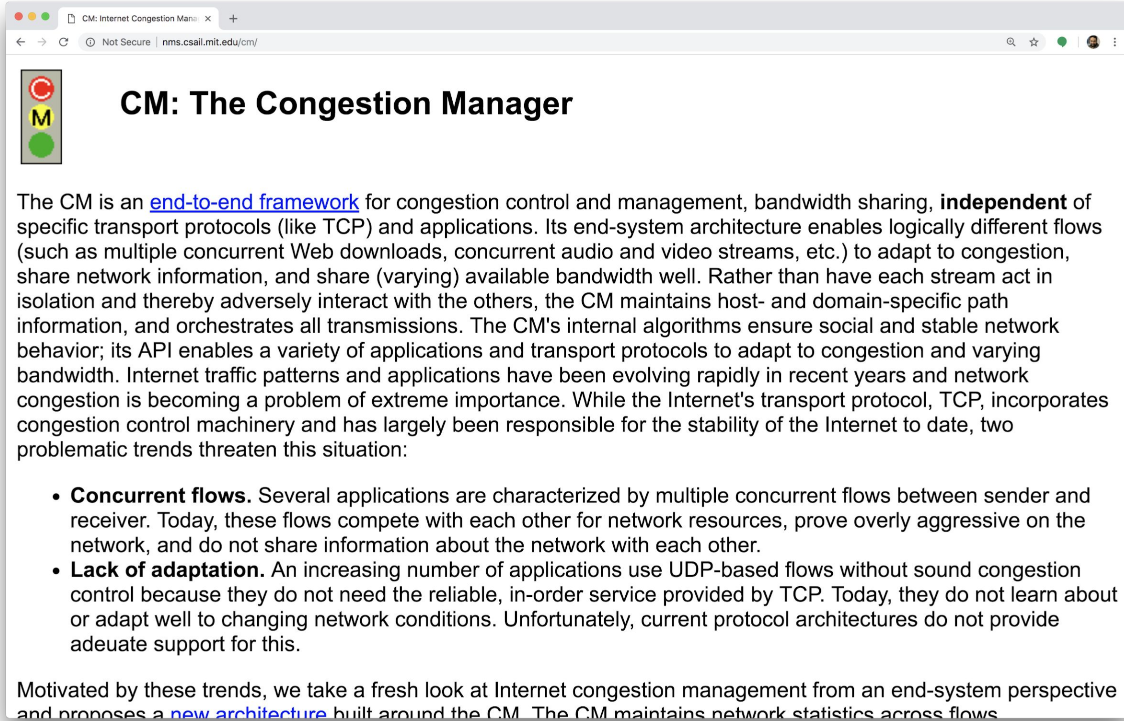
Second, Web data transfers happen in relatively short bursts, with intervening idle periods. It is difficult to utilize bandwidth effectively during a short burst because discovering how much bandwidth is available requires time. Latency suffers as a consequence.

To address these problems, we first developed a new connection abstraction for HTTP, called ***persistent-connection HTTP*** (***P-HTTP***). The key ideas are to share a persistent TCP connection for multiple Web page components and to pipeline the transfers of these components to reduce latency. These ideas, developed by us in 1994, have been adopted by the HTTP/1.1 protocol. The main drawback of P-HTTP, though, is that the persistent TCP connection imposes a linear ordering on the Web page components, which are inherently independent.

"[…] decouples TCP's ordered byte-stream service abstraction from its congestion control and loss recovery mechanisms. It integrates the latter mechanisms across the set of concurrent connections between a pair of hosts […]"

The Transport Story

T/TCP (1994)
TCP Session (1997)
Congestion Manager (1998)



CM: The Congestion Manager

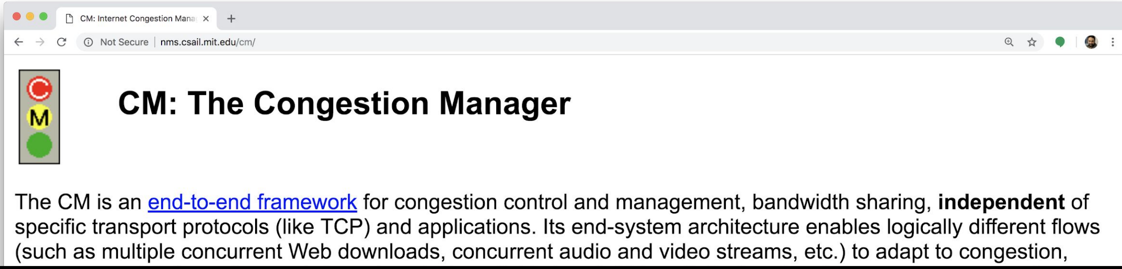
The CM is an end-to-end framework for congestion control and management, bandwidth sharing, **independent** of specific transport protocols (like TCP) and applications. Its end-system architecture enables logically different flows (such as multiple concurrent Web downloads, concurrent audio and video streams, etc.) to adapt to congestion, share network information, and share (varying) available bandwidth well. Rather than have each stream act in isolation and thereby adversely interact with the others, the CM maintains host- and domain-specific path information, and orchestrates all transmissions. The CM's internal algorithms ensure social and stable network behavior; its API enables a variety of applications and transport protocols to adapt to congestion and varying bandwidth. Internet traffic patterns and applications have been evolving rapidly in recent years and network congestion is becoming a problem of extreme importance. While the Internet's transport protocol, TCP, incorporates congestion control machinery and has largely been responsible for the stability of the Internet to date, two problematic trends threaten this situation:

- **Concurrent flows.** Several applications are characterized by multiple concurrent flows between sender and receiver. Today, these flows compete with each other for network resources, prove overly aggressive on the network, and do not share information about the network with each other.
- **Lack of adaptation.** An increasing number of applications use UDP-based flows without sound congestion control because they do not need the reliable, in-order service provided by TCP. Today, they do not learn about or adapt well to changing network conditions. Unfortunately, current protocol architectures do not provide adeuate support for this.

Motivated by these trends, we take a fresh look at Internet congestion management from an end-system perspective and proposes a new architecture built around the CM. The CM maintains network statistics across flows,

The Transport Story

T/TCP (1994)
TCP Session (1997)
Congestion Manager (1998)


CM: Internet Congestion Mana
Not Secure | nms.csail.mit.edu/cm/

CM: The Congestion Manager

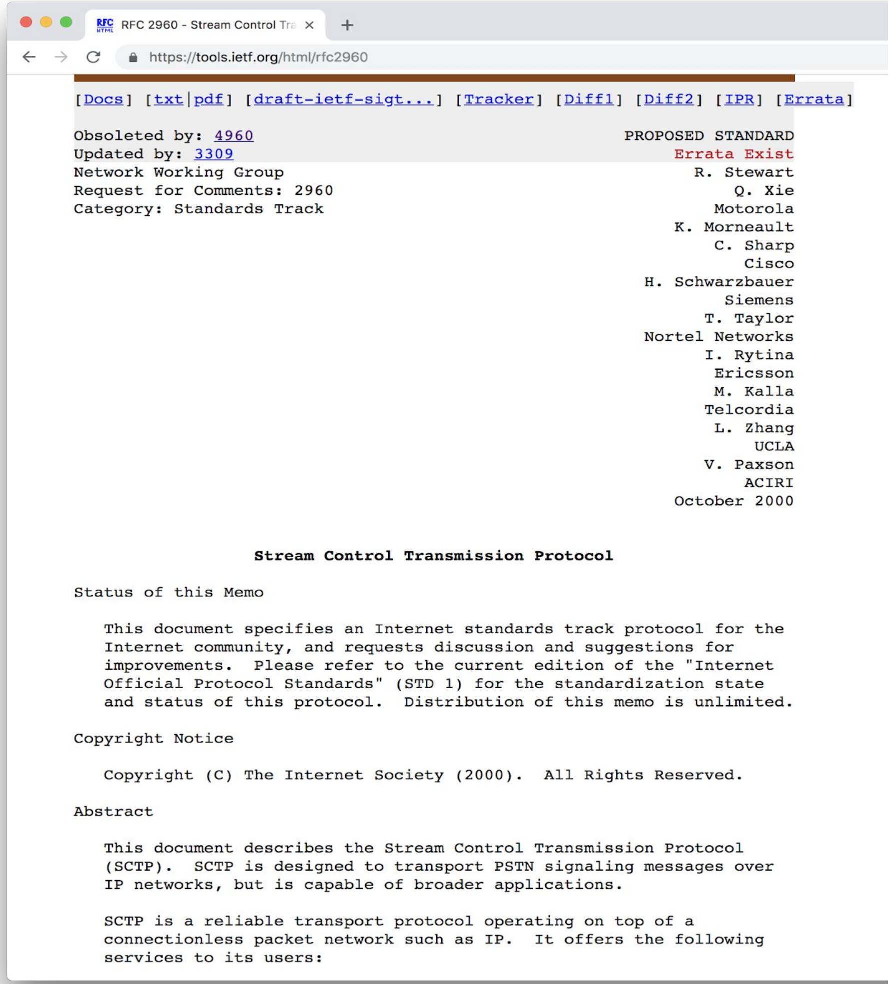
The CM is an end-to-end framework for congestion control and management, bandwidth sharing, **independent** of specific transport protocols (like TCP) and applications. Its end-system architecture enables logically different flows (such as multiple concurrent Web downloads, concurrent audio and video streams, etc.) to adapt to congestion,

"[...] framework integrates congestion management across all applications and transport protocols [...]"

"[...] an ensemble of concurrent TCP connections can effectively share bandwidth and obtain consistent performance [...]"

The Transport Story

T/TCP (1994)
TCP Session (1997)
Congestion Manager (1998)
SCTP (2000)



Obsoleted by: 4960 PROPOSED STANDARD
Updated by: 3309 Errata Exist
Network Working Group R. Stewart
Request for Comments: 2960 Q. Xie
Category: Standards Track Motorola
K. Morneault
C. Sharp
Cisco
H. Schwarzbauer
Siemens
T. Taylor
Nortel Networks
I. Rytina
Ericsson
M. Kalla
Telcordia
L. Zhang
UCLA
V. Paxson
ACIRI
October 2000

Stream Control Transmission Protocol

Status of this Memo

This document specifies an Internet standards track protocol for the Internet community, and requests discussion and suggestions for improvements. Please refer to the current edition of the "Internet Official Protocol Standards" (STD 1) for the standardization state and status of this protocol. Distribution of this memo is unlimited.

Copyright Notice

Copyright (C) The Internet Society (2000). All Rights Reserved.

Abstract

This document describes the Stream Control Transmission Protocol (SCTP). SCTP is designed to transport PSTN signaling messages over IP networks, but is capable of broader applications.

SCTP is a reliable transport protocol operating on top of a connectionless packet network such as IP. It offers the following services to its users:

The Transport Story

T/TCP (1994)
TCP Session (1997)
Congestion Manager (1998)
SCTP (2000)
...
...

Rise of the Middle

mid 1990s: the network started to change

- Network Address Translators (NATs): IP address scarcity
- Firewalls: Protection and policy
- Protocol accelerators (PEPs): Improve transfer perf

Eroding End-to-End

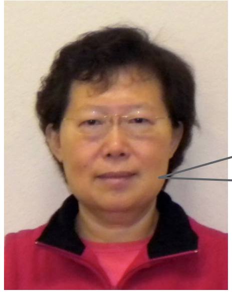
Network devices started to read/modify end-to-end information

- NATs: transport port number, checksum
- Others: most transport header fields, state machine

Eroding End-to-End

Network devices started to read/modify end-to-end information

- NATs: transport port number, checksum
- Others: most transport header fields, state machine



“Middleboxes”

Middleboxes

“[…] intermediary device performing functions other than the normal, standard functions of an IP router on the datagram path between a source host and destination host” - RFC 3234

Middleboxes:
Accidental architectural control points of the Internet

The Transport Story

T/TCP (1994)
TCP Session (1997)
Congestion Manager (1998)
SCTP (2000)
...
...

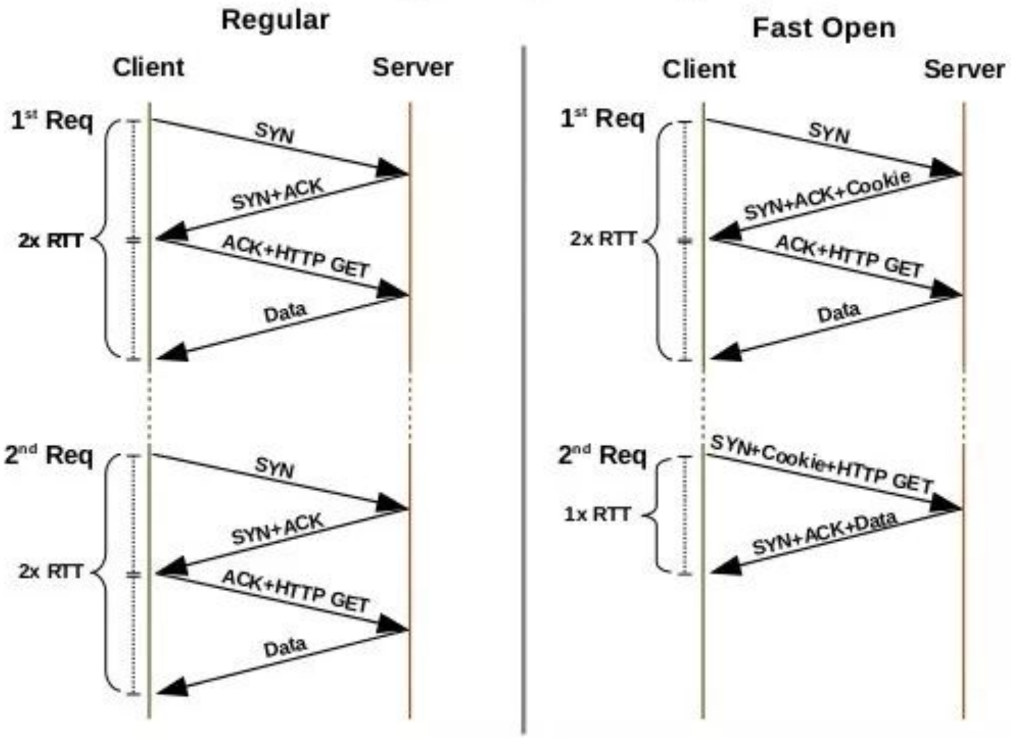
SST (UDP-based) (2007)
Minion (TCP and TLS based) (2011)

The Transport Story

T/TCP (1994)
TCP Session (1997)
Congestion Manager (1998)
SCTP (2000)
...
...

SST (UDP-based) (2007)
Minion (TCP and TLS based) (

TCP Fast Open (2009 - 2014)



The Transport Story

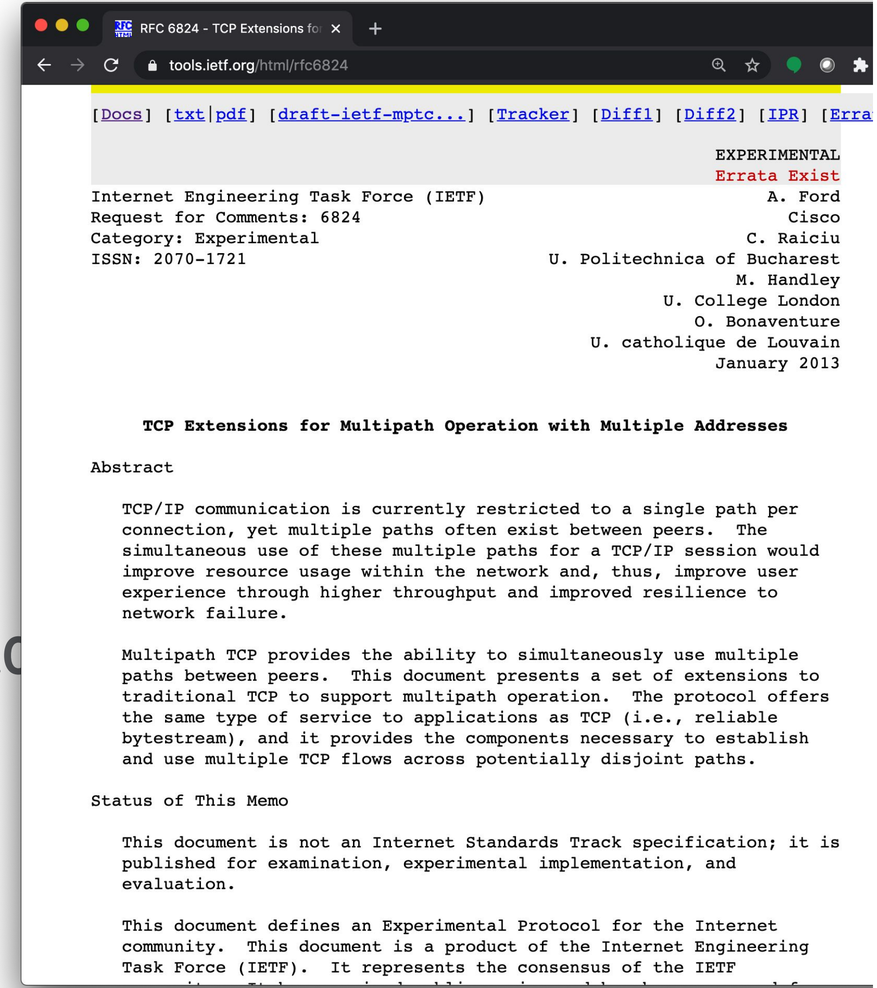
T/TCP (1994)
TCP Session (1997)
Congestion Manager (1998)
SCTP (2000)

...

...

SST (UDP-based) (2007)
Minion (TCP and TLS based) (20

TCP Fast Open (2009 - 2014)
MPTCP (2009 - 2013)



RFC 6824 - TCP Extensions for

tools.ietf.org/html/rfc6824

[Docs] [txt|pdf] [draft-ietf-mptc...] [Tracker] [Diff1] [Diff2] [IPR] [Erra

EXPERIMENTAL
Errata Exist

Internet Engineering Task Force (IETF)	A. Ford
Request for Comments: 6824	Cisco
Category: Experimental	C. Raiciu
ISSN: 2070-1721	U. Politechnica of Bucharest
	M. Handley
	U. College London
	O. Bonaventure
	U. catholique de Louvain
	January 2013

TCP Extensions for Multipath Operation with Multiple Addresses

Abstract

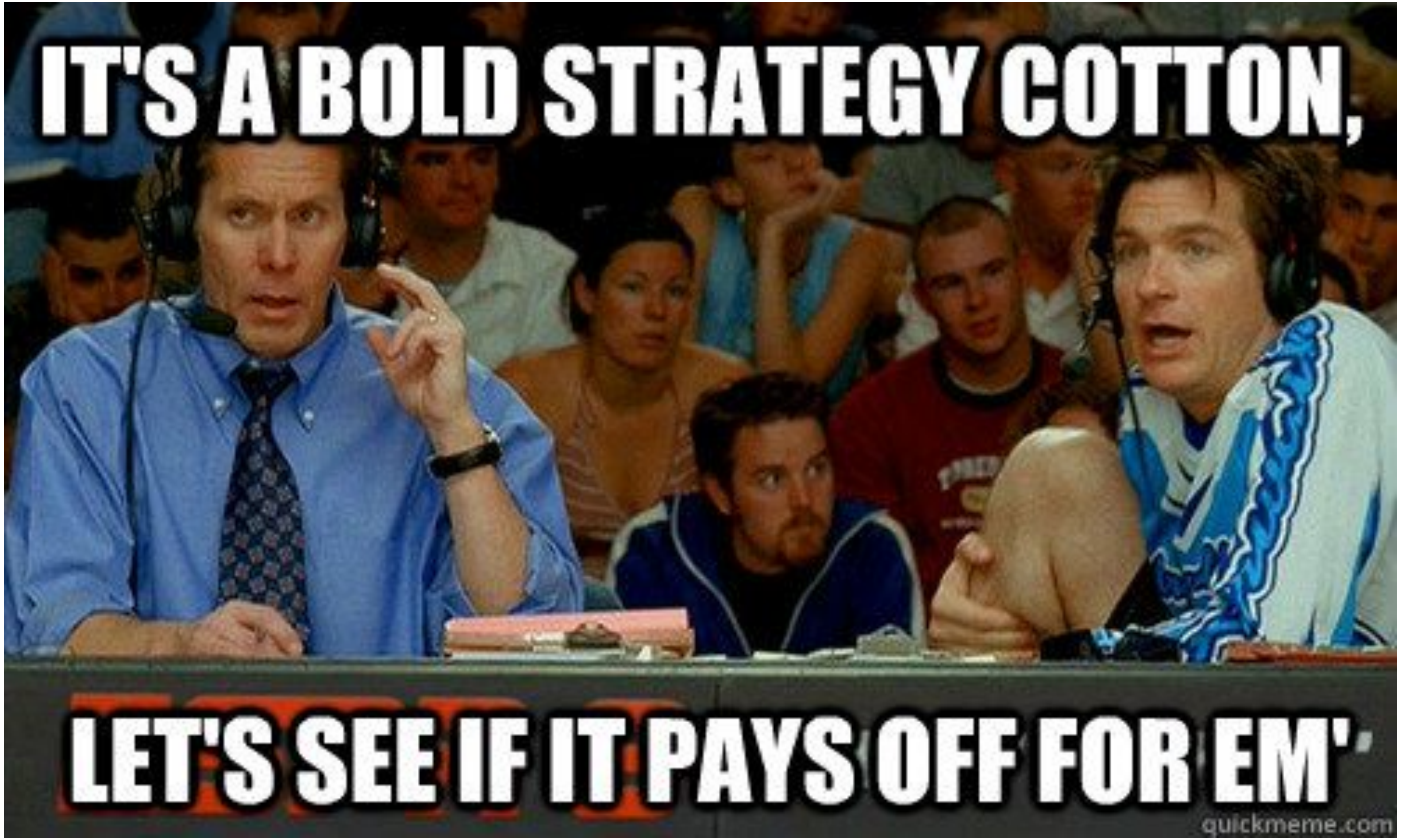
TCP/IP communication is currently restricted to a single path per connection, yet multiple paths often exist between peers. The simultaneous use of these multiple paths for a TCP/IP session would improve resource usage within the network and, thus, improve user experience through higher throughput and improved resilience to network failure.

Multipath TCP provides the ability to simultaneously use multiple paths between peers. This document presents a set of extensions to traditional TCP to support multipath operation. The protocol offers the same type of service to applications as TCP (i.e., reliable bytestream), and it provides the components necessary to establish and use multiple TCP flows across potentially disjoint paths.

Status of This Memo

This document is not an Internet Standards Track specification; it is published for examination, experimental implementation, and evaluation.

This document defines an Experimental Protocol for the Internet community. This document is a product of the Internet Engineering Task Force (IETF). It represents the consensus of the IETF

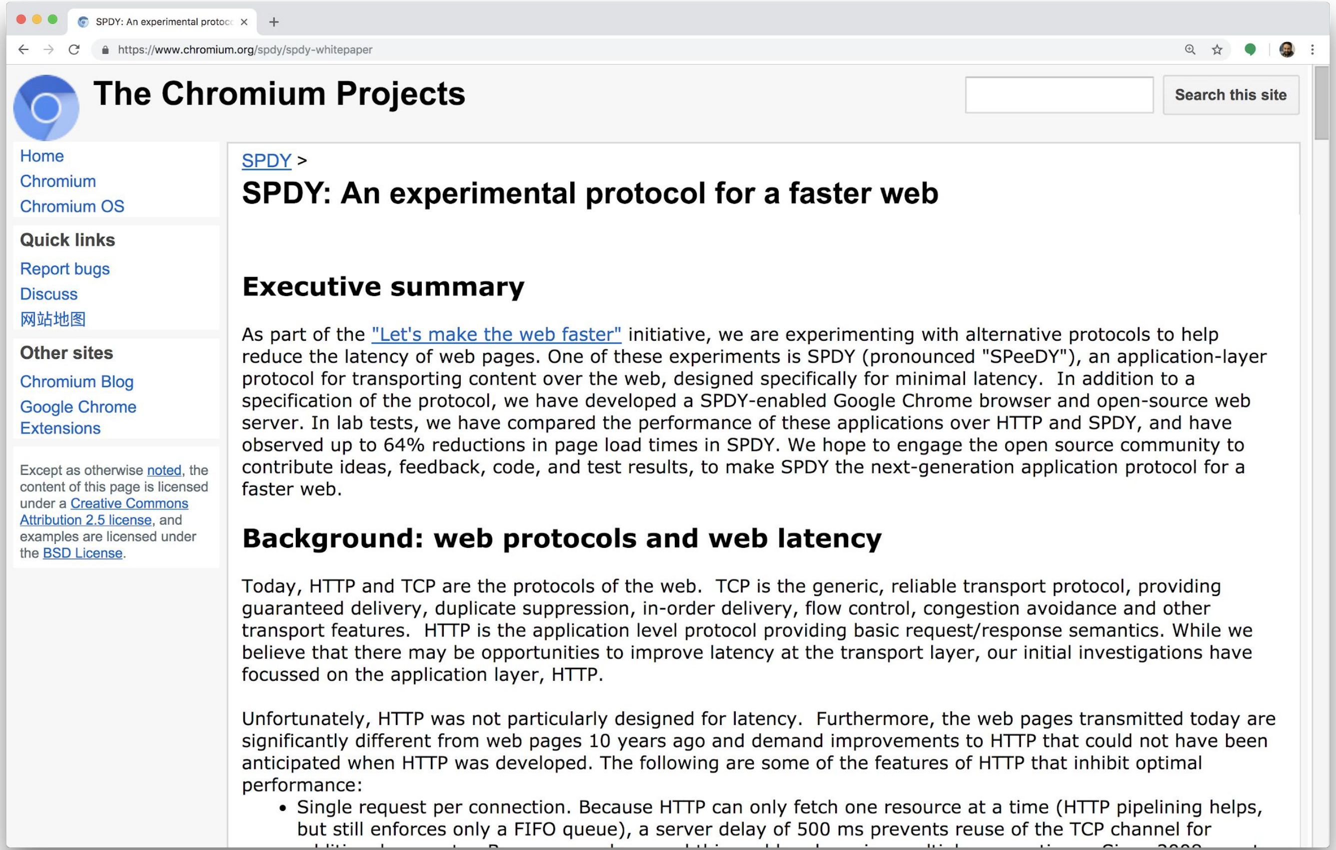

IT'S A BOLD STRATEGY COTTON,
LET'S SEE IF IT PAYS OFF FOR EM'
quickmeme.com

The HTTP Story (contd.)

HTTP/1.0
HTTP/1.1
HTTPng (?)

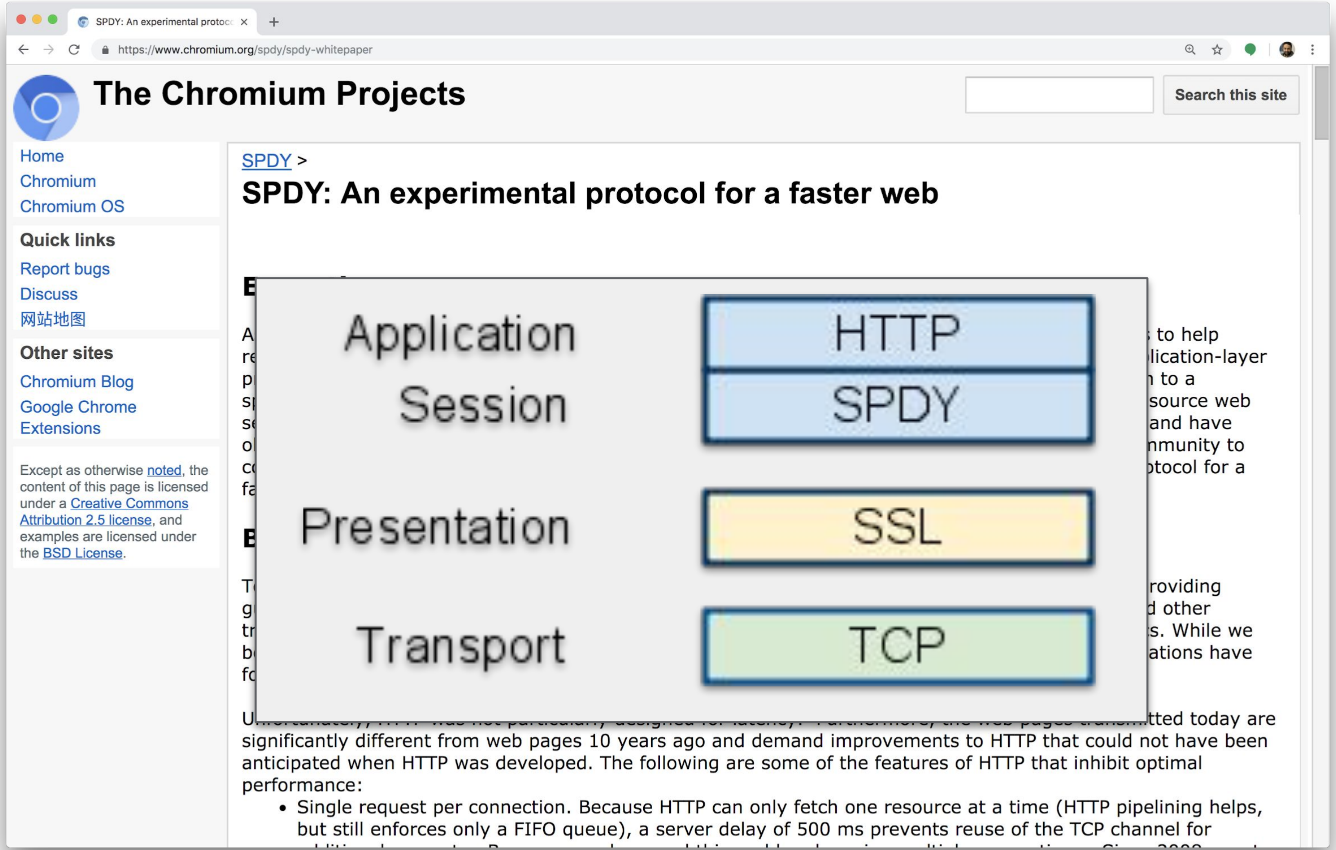
The HTTP Story

HTTP/1.0
HTTP/1.1
HTTPng (?)
...
...
...
...
...
SPDY



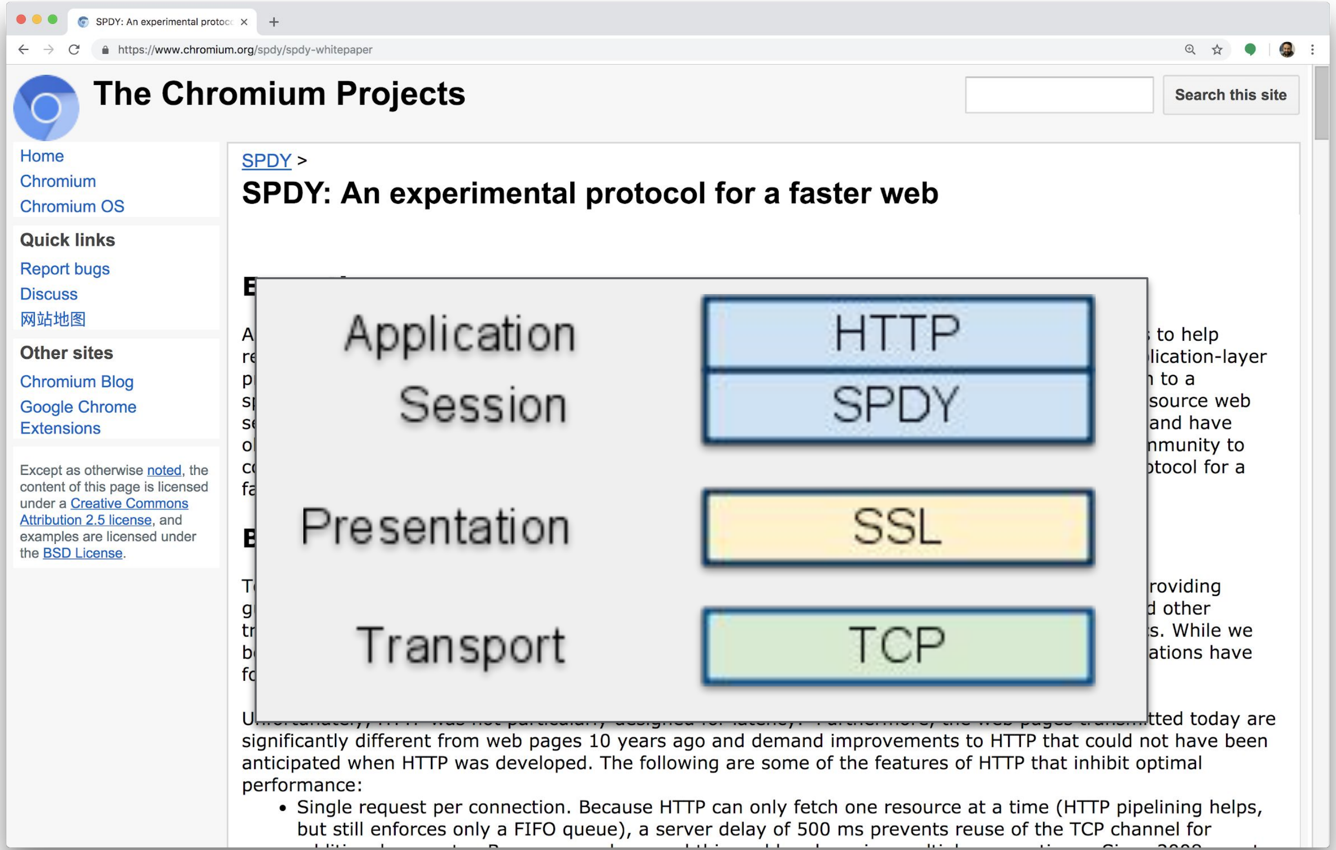
The HTTP Story

HTTP/1.0
HTTP/1.1
HTTPng (?)
...
...
...
...
...
SPDY



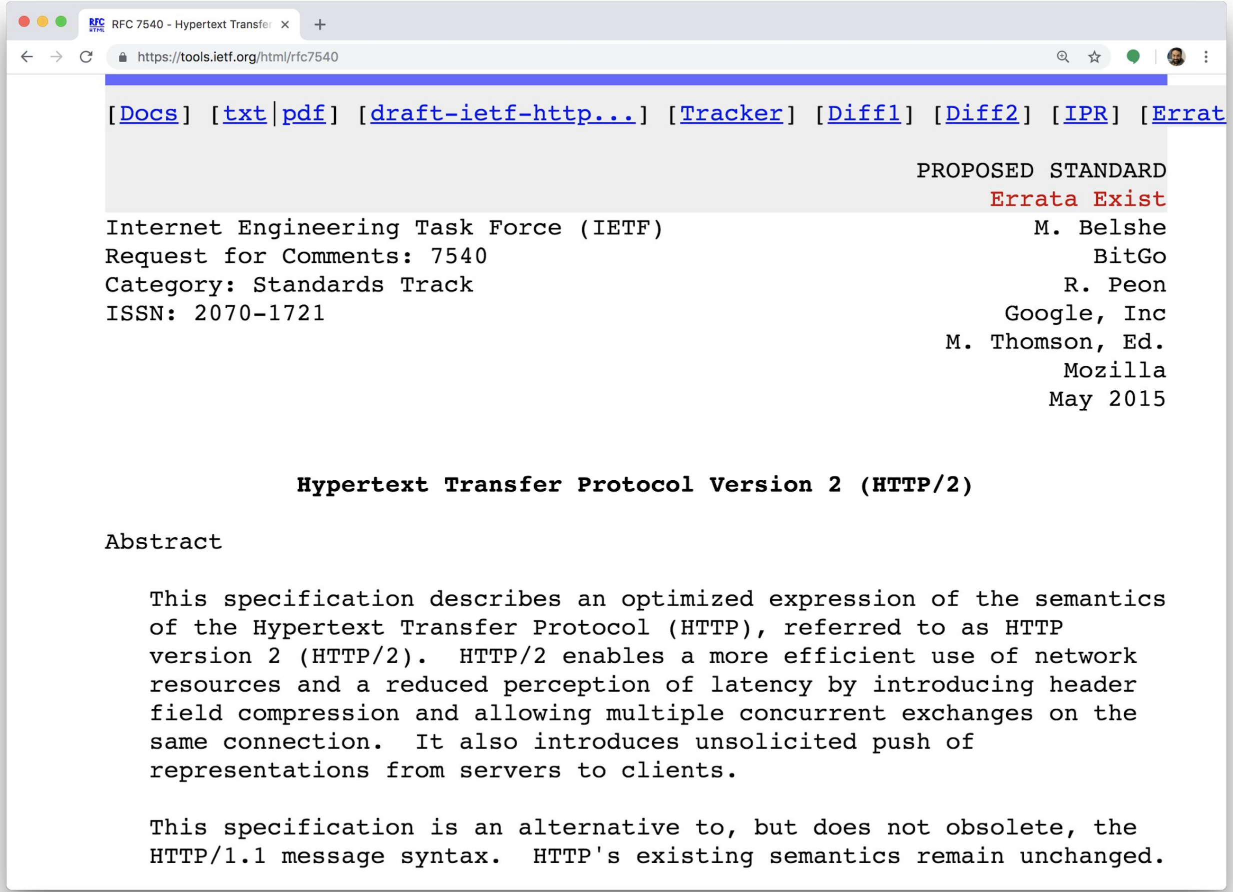
The HTTP Story

HTTP/1.0
HTTP/1.1
HTTPng (?)
...
...
...
...
...
SPDY
- streams
- multiplexing
- flow control
- priorities



The HTTP Story

HTTP/1.0
HTTP/1.1
HTTPng (?)
...
...
...
...
...
HTTP/2
- streams
- multiplexing
- flow control
- priorities



[Docs] [txt|pdf] [draft-ietf-http...] [Tracker] [Diff1] [Diff2] [IPR] [Errat

PROPOSED STANDARD
Errata Exist

Internet Engineering Task Force (IETF)
Request for Comments: 7540
Category: Standards Track
ISSN: 2070-1721

M. Belshe
BitGo
R. Peon
Google, Inc
M. Thomson, Ed.
Mozilla

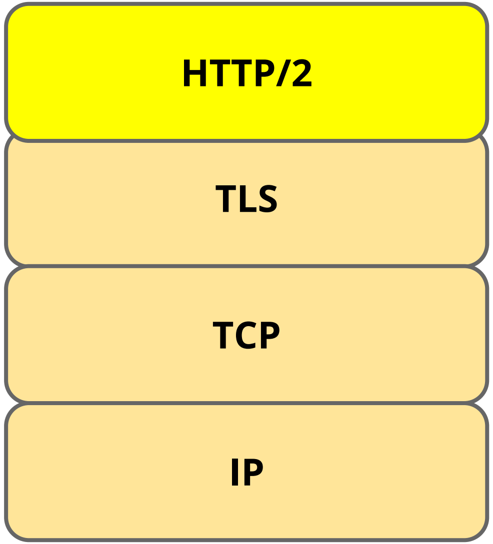
May 2015

Hypertext Transfer Protocol Version 2 (HTTP/2)

Abstract

This specification describes an optimized expression of the semantics of the Hypertext Transfer Protocol (HTTP), referred to as HTTP version 2 (HTTP/2). HTTP/2 enables a more efficient use of network resources and a reduced perception of latency by introducing header field compression and allowing multiple concurrent exchanges on the same connection. It also introduces unsolicited push of representations from servers to clients.

This specification is an alternative to, but does not obsolete, the HTTP/1.1 message syntax. HTTP's existing semantics remain unchanged.


HTTP/2
TLS
TCP
IP

Persisting Demands on Web Architecture

Latency = $$

Everything's going over HTTP

Video over HTTP, DNS over HTTP, …

HTTP needs to scale

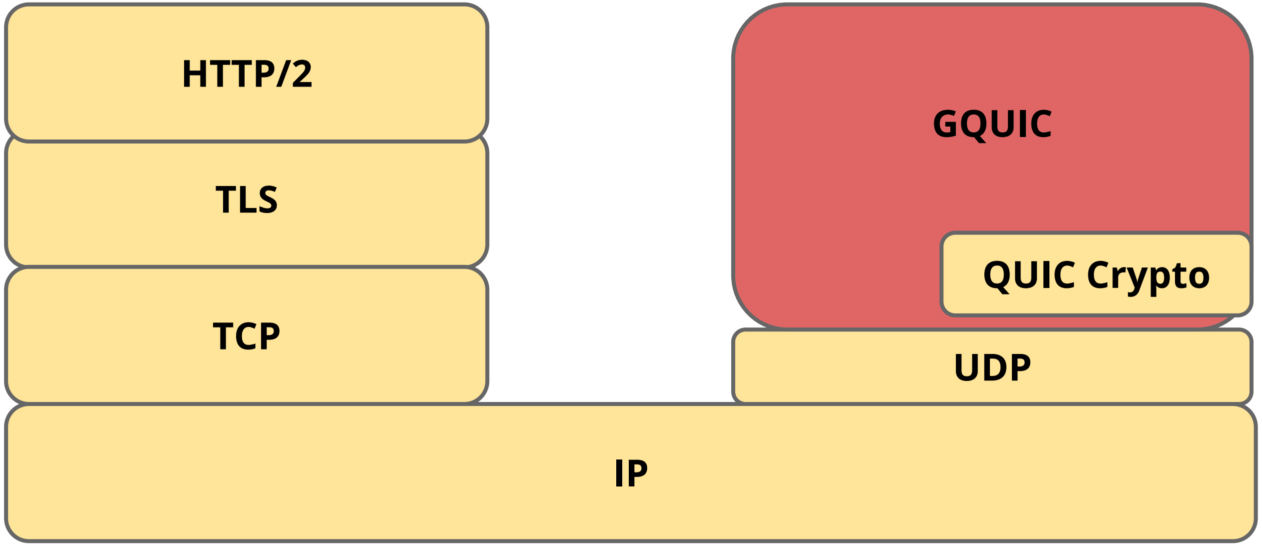
laterally: wide API, broad applicability

vertically: no delays and inefficiencies

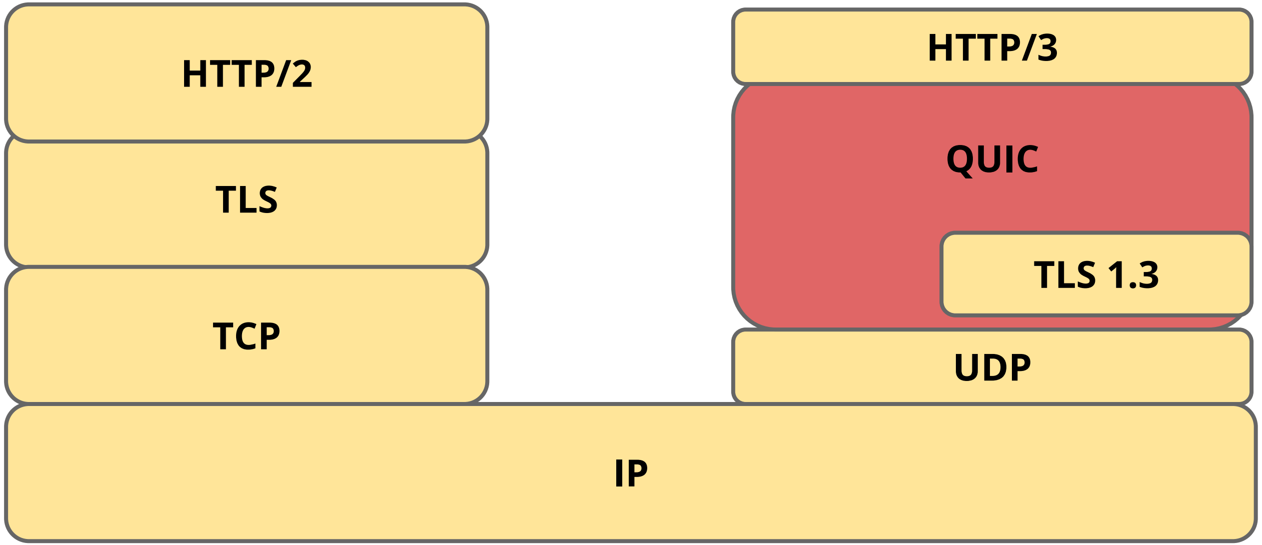

MONTY
Monty Python
EVERY MILLISECOND
IS SACRED

How do we eliminate inefficiences in the web stack?
In HTTP?
In TLS?
In TCP?
In the layering?

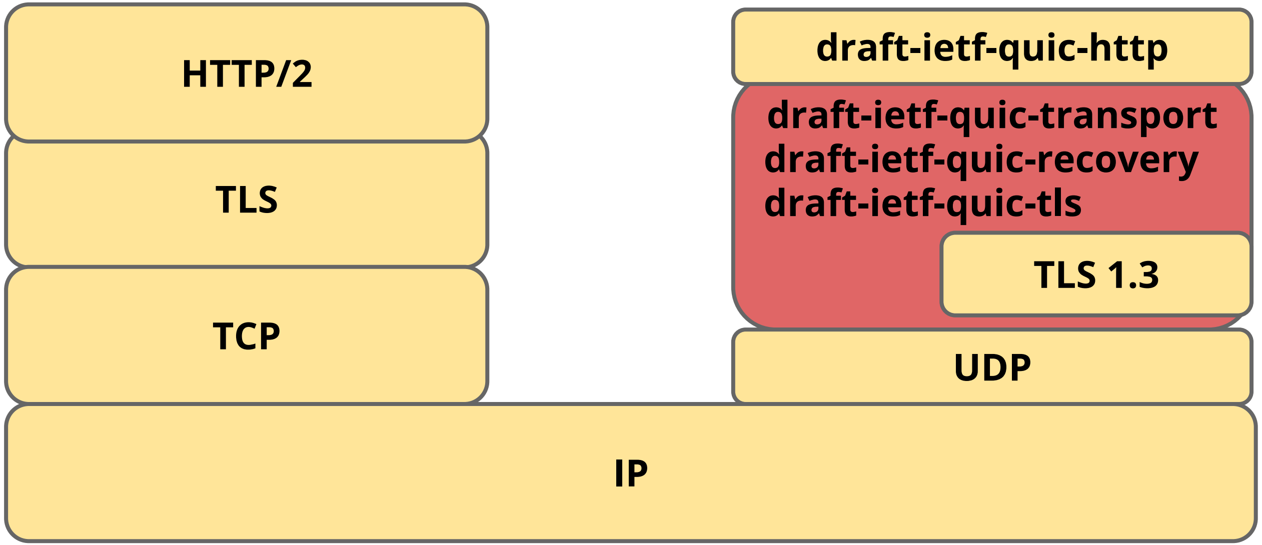
Google’s GQUIC Experiment



The QUIC Standard



The QUIC Standard



Why does industry care?

Site performance

reduces page load latency, improves video QoE

Userspace transport

offers control, agility

enables architecture exploration, such as “Direct Server Return”

Deployment agility of new features

ossification protection with versioning, encryption, GREASEing

QUIC Status

IETF:

specifications in-progress, RFCs likely in 2021

Implementations:

Apple, Facebook, Fastly, Firefox, F5, Google, Microsoft …

Server deployments have been going on for a while

Akamai, Cloudflare, Facebook, Fastly, Google …

Clients are at different stages of deployment

Chrome, Firefox, Edge, Safari

iOS, MacOS

Plan

#	Start - End	Topic
1	1:40 - 1:58	QUIC’s intellectual heritage
2	**2:00 - 2:18**	**QUIC handshake, headers, connection migration**
3	2:20 - 2:38	Wireshark demo and tutorial
4	2:40 - 2:58	QUIC streams, flow control, frames, packetization
5	**3:00 - 3:18**	**BREAK**
6	3:20 - 3:38	qlog and qvis demo and tutorial
7	3:40 - 3:58	QUIC loss detection and congestion control: how different from TCP?
8	4:00 - 4:18	Build your own congestion controller. Code walkthrough: quicly and quiche
9	4:20 - 4:38	Extending QUIC: transport parameters and extensions (Ack Frequency)
10	4:40 - 5:00	Open Discussion, Q & A

Why QUIC?

Low latency

Encrypted Transport

Resilient Connections

Why QUIC?

Low latency

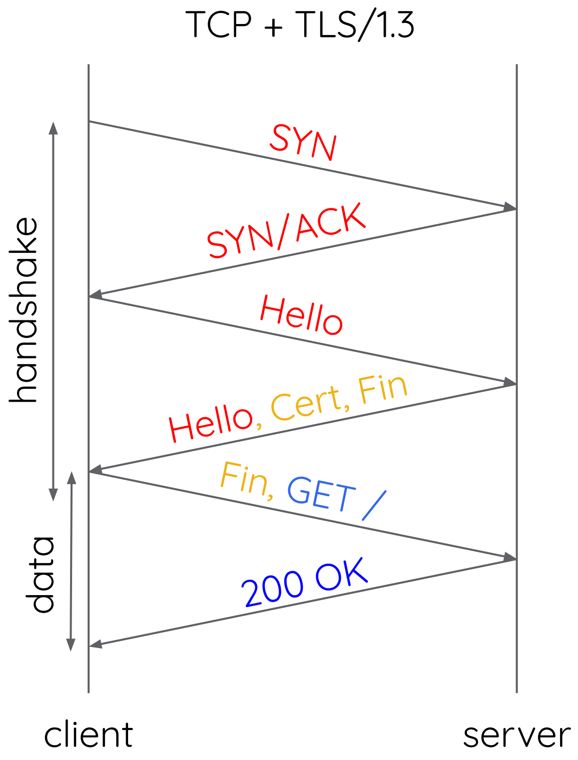
- eliminates latency of new connections to recently visited sites
- eliminates head-of-line blocking in TLS and TCP

Encrypted Transport

Resilient Connections

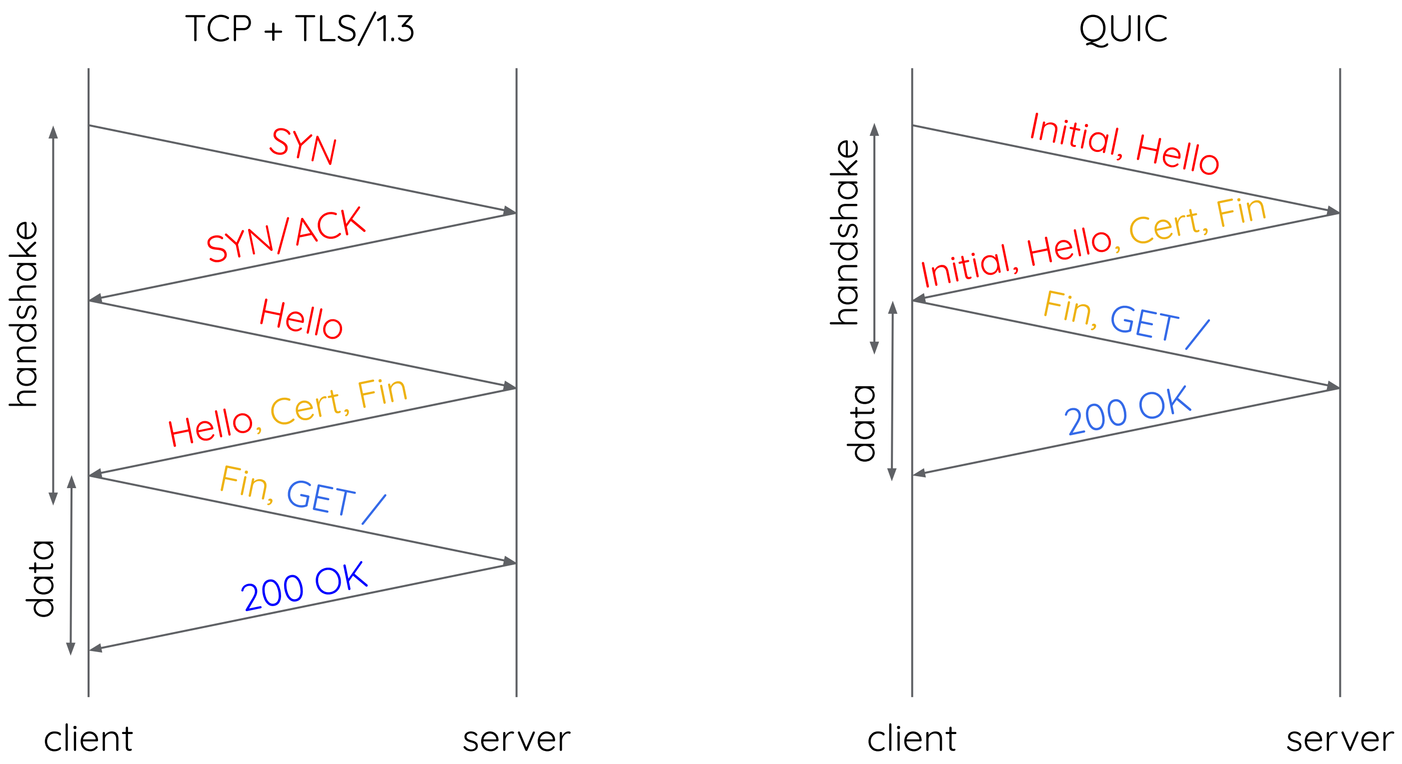
Low-latency handshake

First connection to server



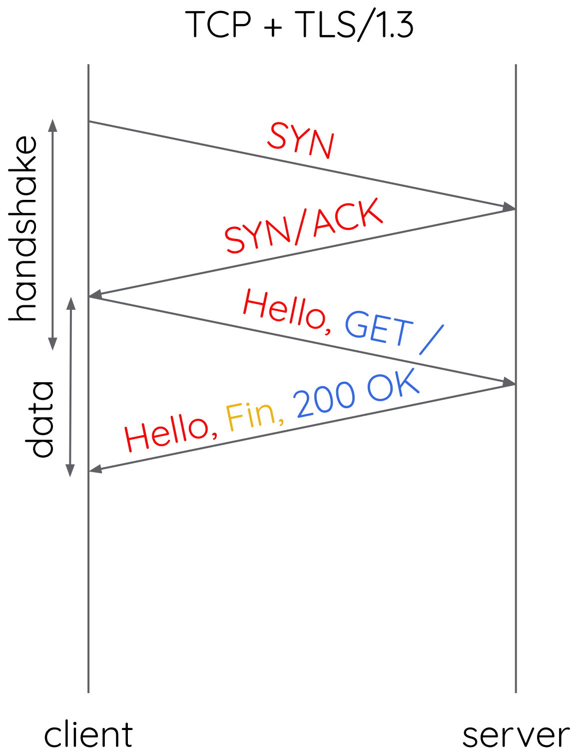
Low-latency handshake

First connection to server



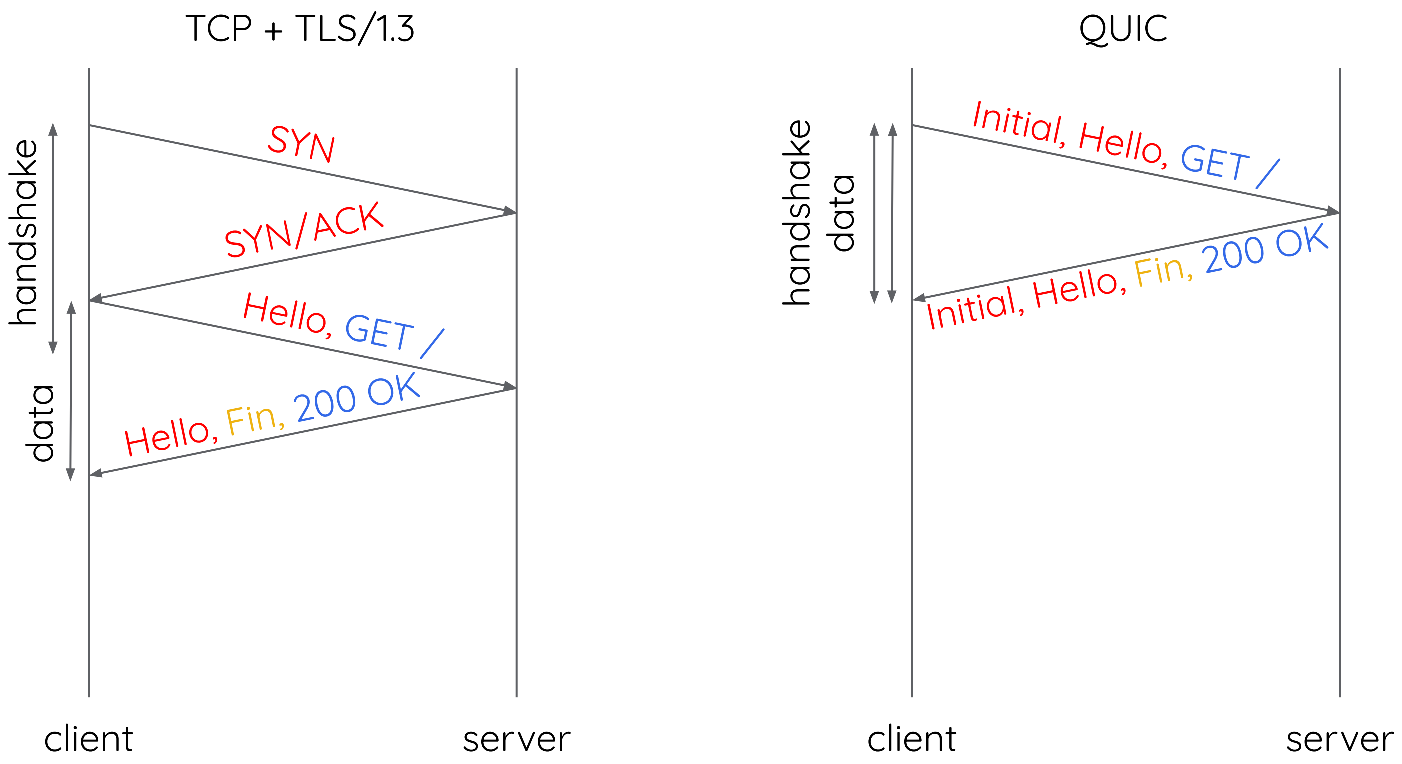
Low-latency handshake

Subsequent connection to the same server



Low-latency handshake

Subsequent connection to the same server



QUIC Handshake

Transport options exchanged in Transport Parameters

- Flow control limits, etc
- Sent as extension to TLS handshake

Connection IDs exchanged during handshake

- Each endpoint chooses CID (and length) to be used towards it

TLS handshake carried in QUIC packets

Ultimately, QUIC packets flow on wire

- Carrying TLS messages, including QUIC options
- Why? So that QUIC options are protected

Why QUIC?

Low latency

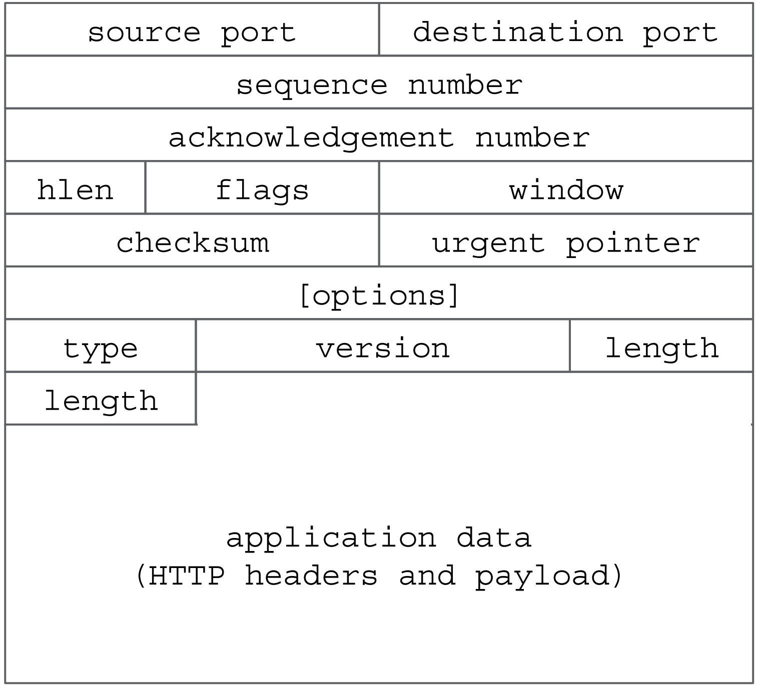
Encrypted transport

- encryption and privacy are fundamental to QUIC
- connections protected from tamper and disruption
- most of the headers not even visible to third parties

Resilient Connections

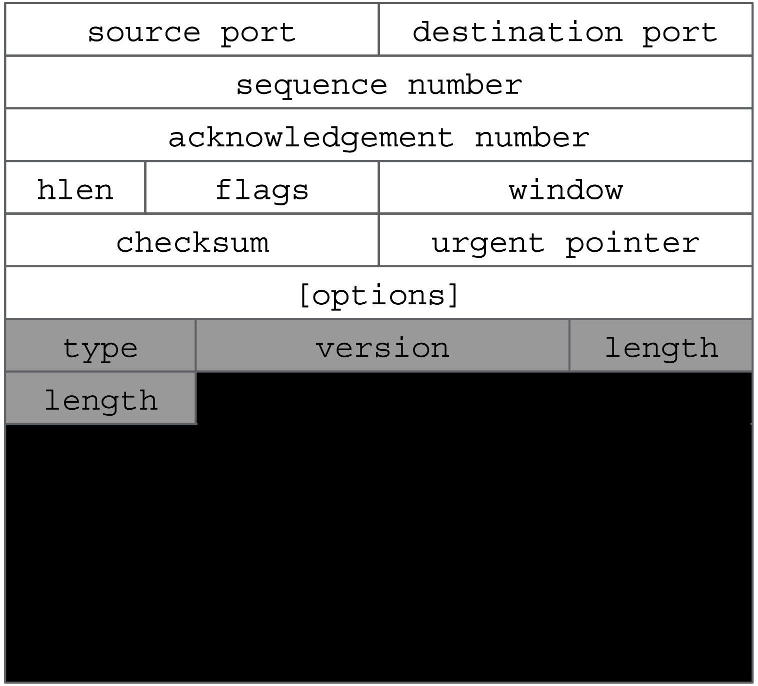
Encrypted transport

HTTP with TLS/TCP



Encrypted transport

HTTP with TLS/TCP



Identifying HTTPS-Protected Netflix Videos in Real-Time

Andrew Reed, Michael Kranch
Dept. of Electrical Engineering and Computer Science
United States Military Academy at West Point
West Point, New York, USA
{andrew.reed, michael.kranch}@usma.edu

ABSTRACT

After more than a year of research and development, Netflix recently upgraded their infrastructure to provide HTTPS encryption of video streams in order to protect the privacy of their viewers. Despite this upgrade, we demonstrate that it is possible to accurately identify Netflix videos from passive traffic capture in real-time with very limited hardware requirements. Specifically, we developed a system that can report the Netflix video being delivered by a TCP connection using only the information provided by TCP/IP headers.

protected Netflix videos. We then improve upon the previous work by fully automating the fingerprint creation process, thereby enabling us to create an extensive collection of Netflix fingerprints which we then use to conduct a robust assessment of the attack. Finally, we developed a network appliance that can, in real-time, identify HTTPS-protected Netflix videos using IP and TCP headers obtained from passive capture of network traffic.

Our primary contributions are:

- A dataset that contains the fingerprints for 42,027 Netflix

CODASPY '17

Identifying HTTPS-Protected Netflix Videos in Real-Time

Andrew Reed, Michael Kranch
Dept. of Electrical Engineering and Computer Science
United States Military Academy at West Point
West Point, New York, USA
{andrew.reed, michael.kranch}@usma.edu

we developed a system that can report the Netflix video being delivered by a TCP connection using only the information provided by TCP/IP headers.

encryption of video streams in order to protect the privacy of their viewers. Despite this upgrade, we demonstrate that it is possible to accurately identify Netflix videos from passive traffic capture in real-time with very limited hardware requirements. Specifically, we developed a system that can report the Netflix video being delivered by a TCP connection using only the information provided by TCP/IP headers.

fingerprints which we then use to conduct a robust assessment of the attack. Finally, we developed a network appliance that can, in real-time, identify HTTPS-protected Netflix videos using IP and TCP headers obtained from passive capture of network traffic.

Our primary contributions are:

- A dataset that contains the fingerprints for 42,027 Netflix

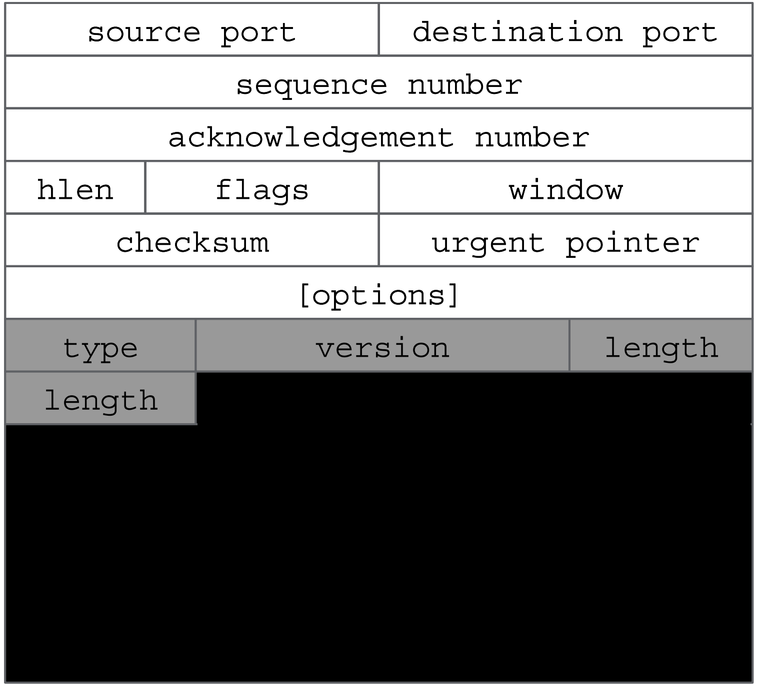
Protocol design maxim

"the ultimate defense of the end to end mode is end to end encryption"

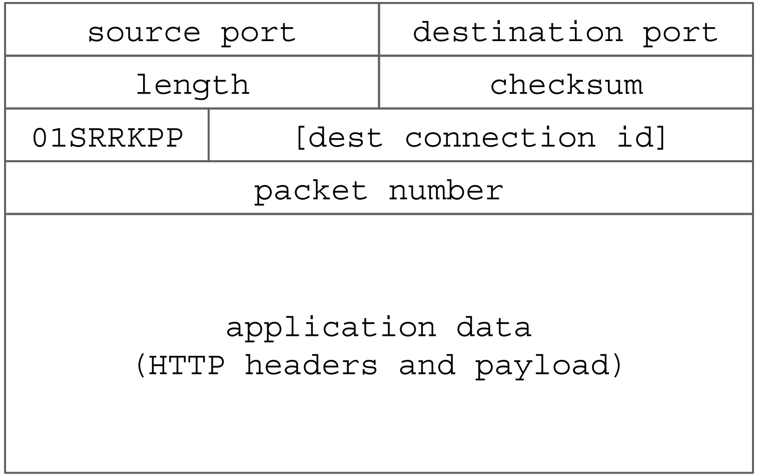
David Clark, J. Wroclawski, K. Sollins, and R. Braden, *Tussle in Cyberspace: Defining Tomorrow's Internet*. IEEE/ACM ToN, 2005.

Encrypted transport

HTTP with TLS/TCP

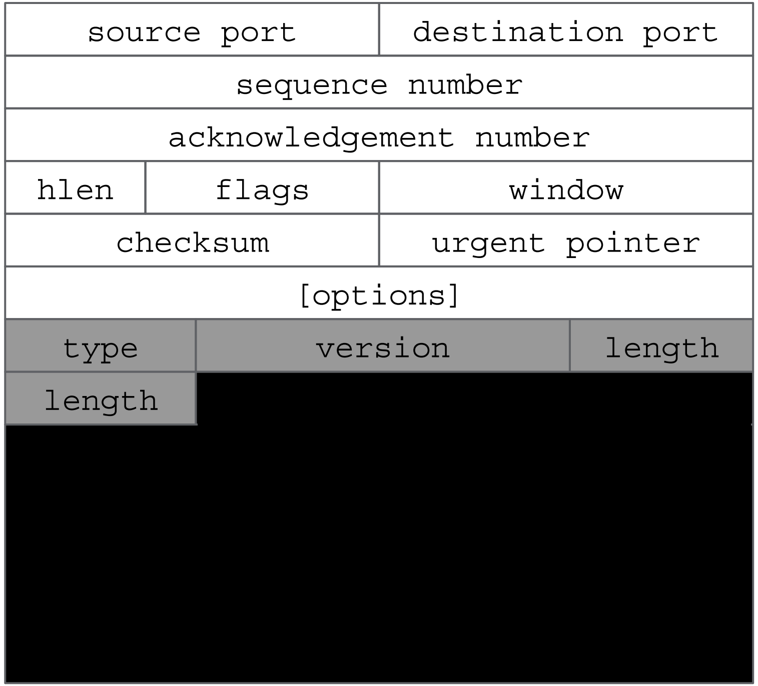


HTTP with QUIC

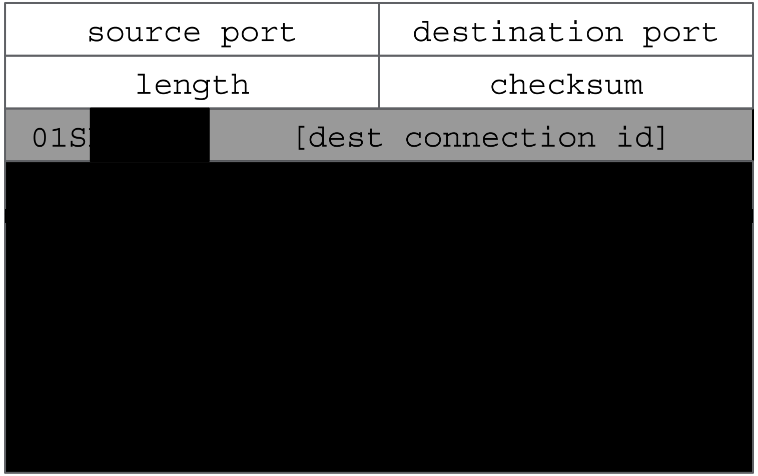


Encrypted transport

HTTP with TLS/TCP



HTTP with QUIC



Why QUIC?

Low latency

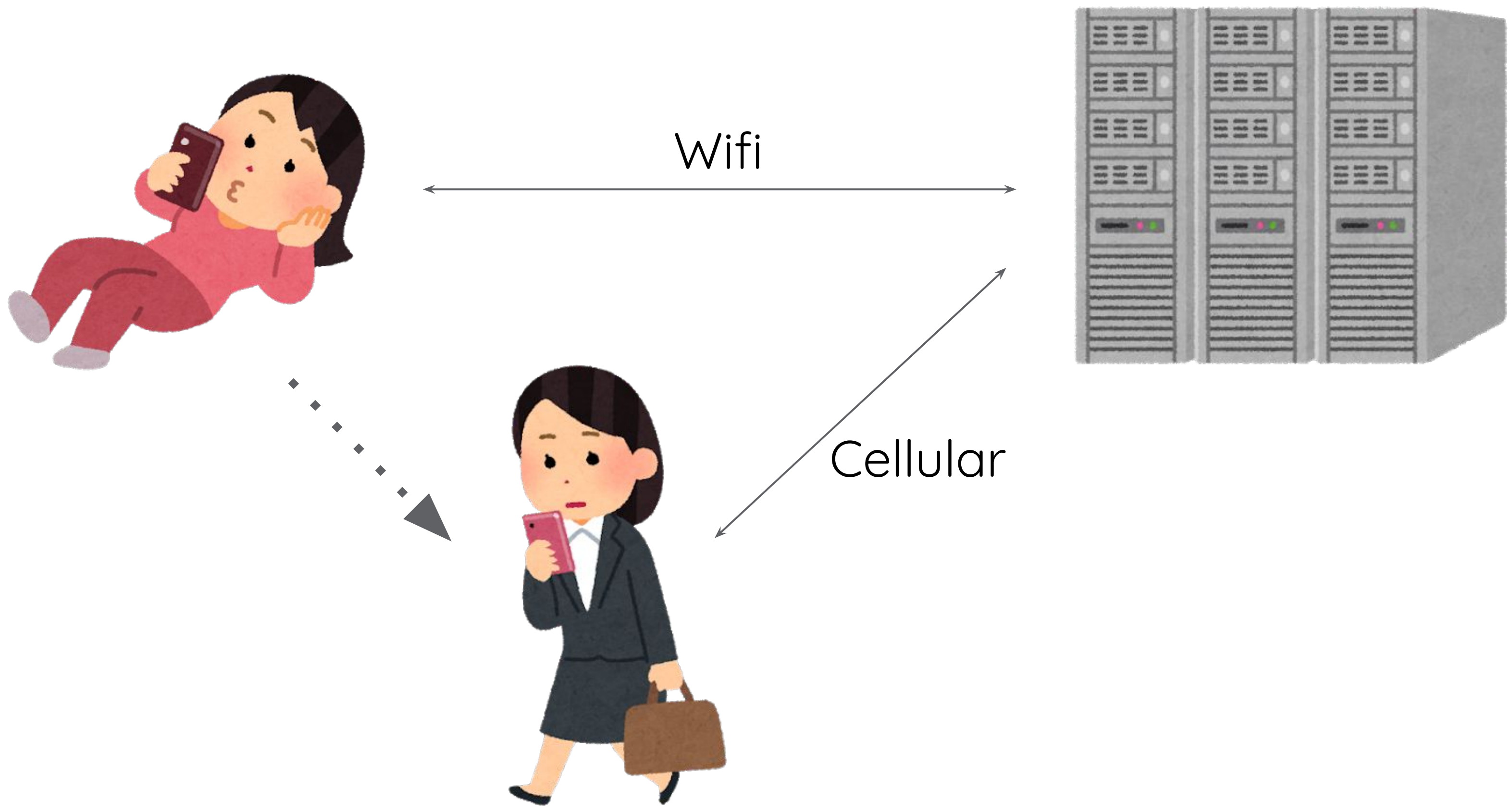
Encrypted transport

Resilient connections

- connection migration for “parking lot” problem
  - using 18-byte connection IDs
- improved loss recovery, helping connections over “bad” networks

Connection migration

Connection migration

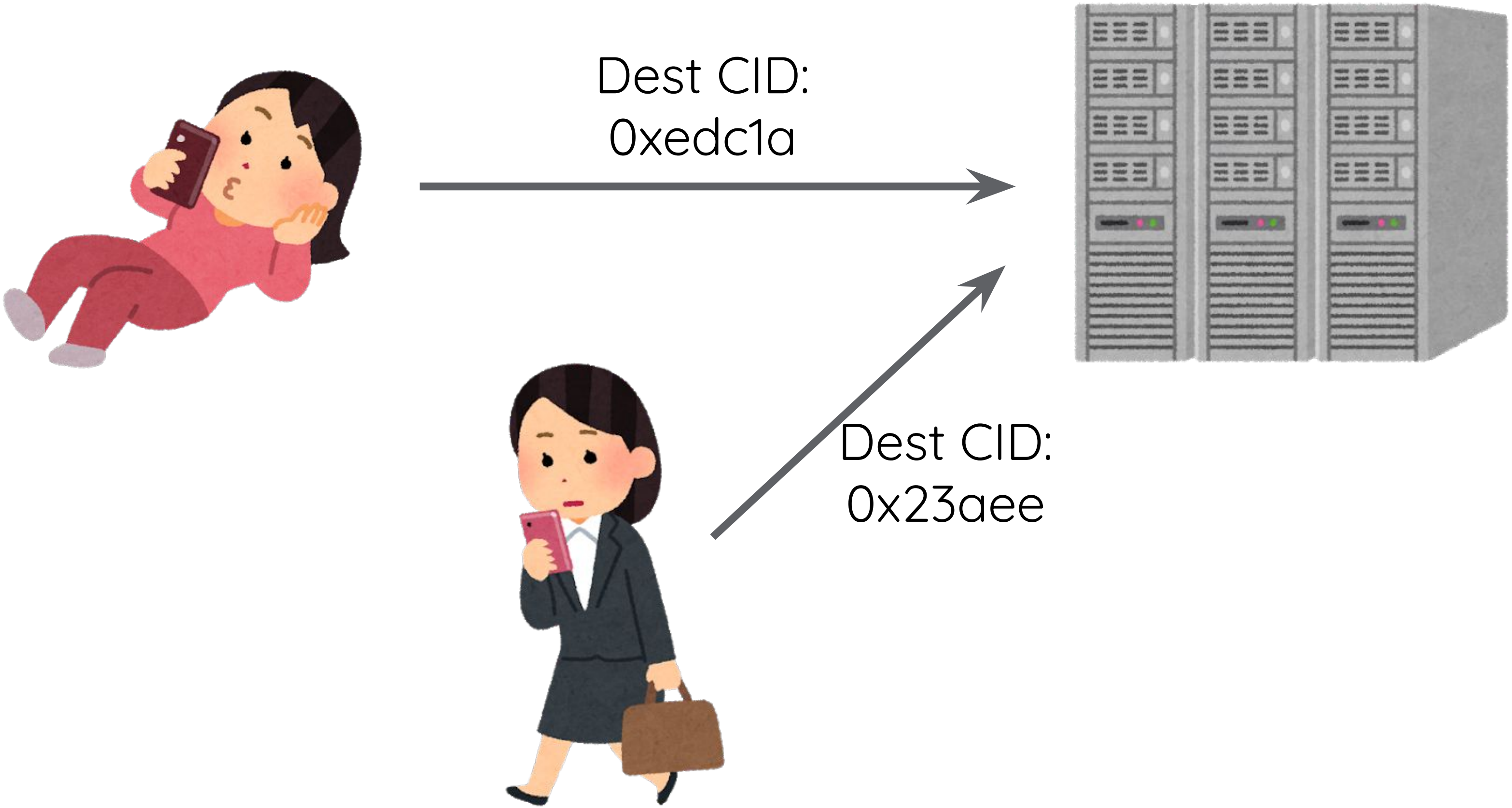


Tracking migrating connections

Tracking migrating connections

NCID: 0x1aeec
NCID: 0x23aee
NCID: 0xfe62a

Tracking migrating connections



Packet Number Encryption

Packet number used as a nonce for packet encryption

- nonce = used once
- receiver needs it to decrypt the packet
- monotonically increasing, for loss detection and compression

Packet Number Encryption

Packet number used as a nonce for packet encryption

nonce = used once

receiver needs it to decrypt the packet

monotonically increasing, for loss detection and compression

Visible packet number allows for correlation across networks

also, any visible bits ossify in the network

Packet Number Encryption

Packet number used as a nonce for packet encryption

- nonce = used once
- receiver needs it to decrypt the packet
- monotonically increasing, for loss detection and compression

Visible packet number allows for correlation across networks

- also, any visible bits ossify in the network

Encrypting packet number would require (another) nonce

Packet Number Encryption

Packet number used as a nonce for packet encryption

- nonce = used once
- receiver needs it to decrypt the packet
- monotonically increasing, for loss detection and compression

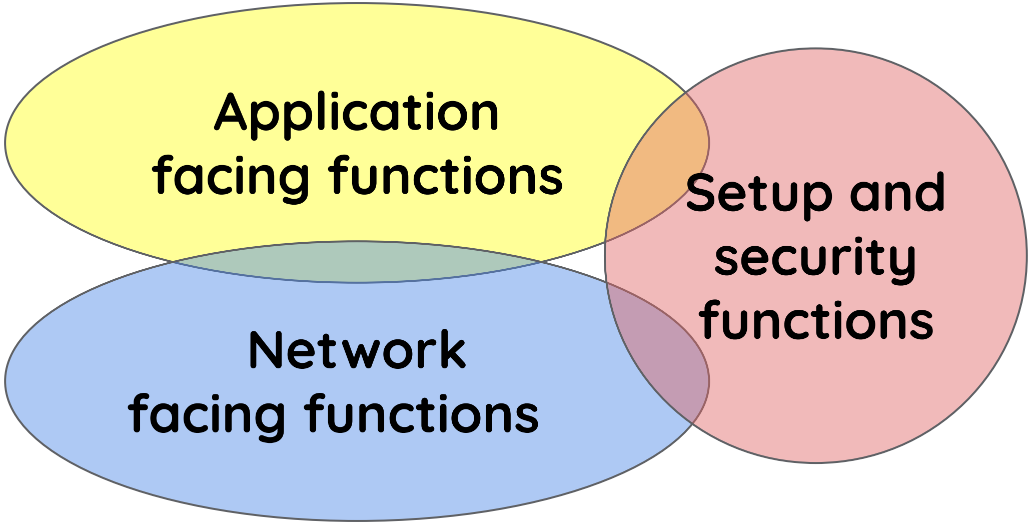
Visible packet number allows for correlation across networks

- also, any visible bits ossify in the network

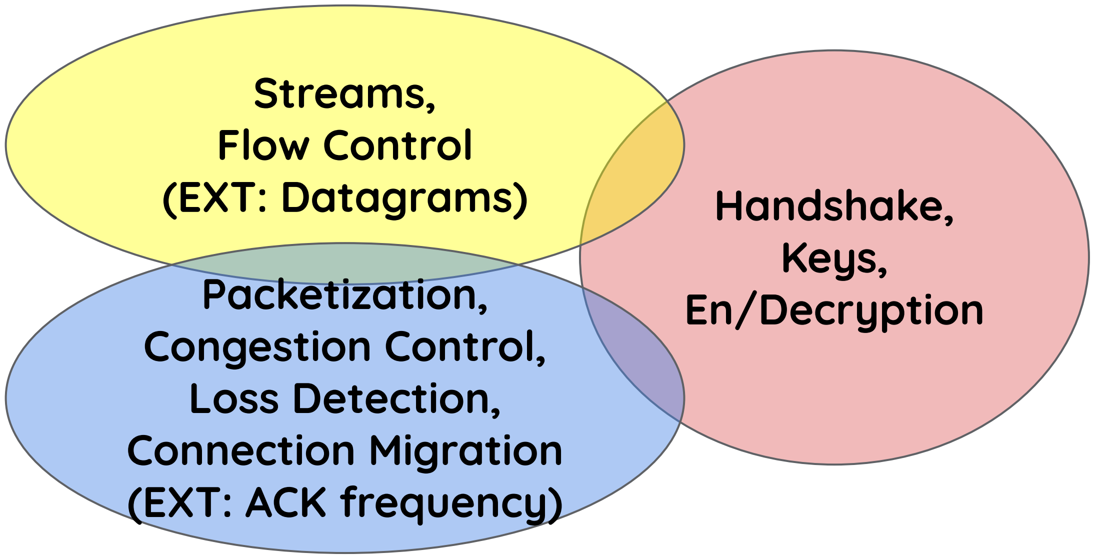
Encrypting packet number would require (another) nonce

- **Idea:** encrypted bytes from the packet are random…
- therefore, can be nonce!

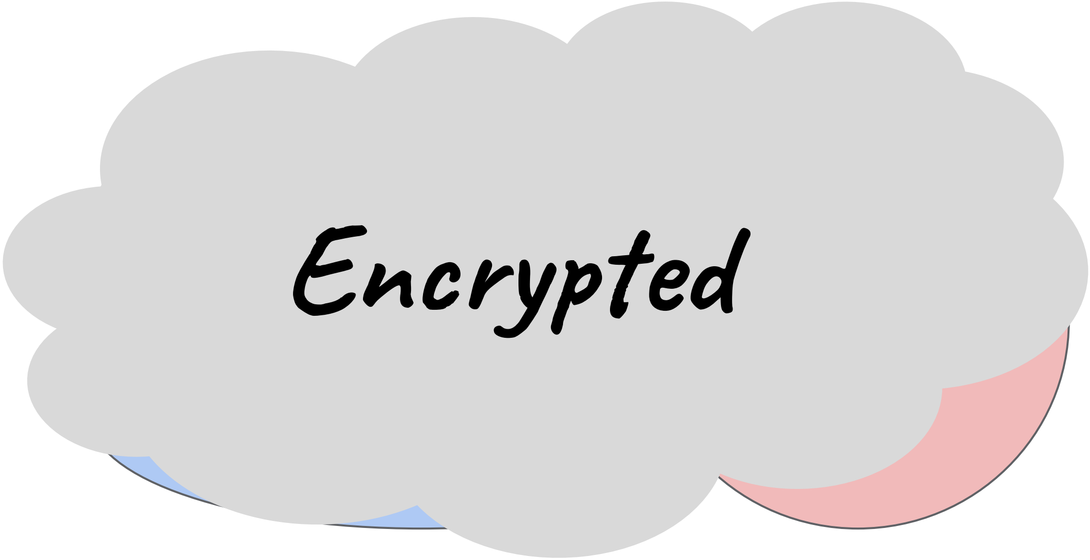
Functional Decomposition of Transport



Functional Decomposition of QUIC



Functional Decomposition of QUIC



QUIC Packet Format

Long header

Short header

QUIC Packet Format

Long header

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+
|1|1|T T|X X X X|
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                         Version (32)                          |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|DCIL(4)|SCIL(4)|
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|               Destination Connection ID (0/32..144)         ...
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                 Source Connection ID (0/32..144)            ...
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

QUIC Packet Format

Long header

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+
|1|1|T T|X X X X|
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                         Version (32)                          |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|DCIL(4)|SCIL(4)|
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|               Destination Connection ID (0/32..144)         ...
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                 Source Connection ID (0/32..144)            ...
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

0x0	Initial
0x1	0-RTT
0x2	Handshake
0x3	Retry

QUIC Packet Format

Long header

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+
|1|1|T T|X X X X|
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                         Version (32)                          |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|DCIL(4)|SCIL(4)|
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|               Destination Connection ID (0/32..144)         ...
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                 Source Connection ID (0/32..144)            ...
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Short header

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+
|0|1|S|R|R|K|P P|
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                Destination Connection ID (0..144)           ...
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                     Packet Number (8/16/24/32)              ...
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                     Protected Payload (*)                   ...
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Plan

#	Start - End	Topic
1	1:40 - 1:58	QUIC's intellectual heritage
2	2:00 - 2:18	QUIC handshake, headers, connection migration
3	**2:20 - 2:38**	**Wireshark demo and tutorial**
4	2:40 - 2:58	QUIC streams, flow control, frames, packetization
5	**3:00 - 3:18**	**BREAK**
6	3:20 - 3:38	qlog and qvis demo and tutorial
7	3:40 - 3:58	QUIC loss detection and congestion control: how different from TCP?
8	4:00 - 4:18	Build your own congestion controller. Code walkthrough: quicly and quiche
9	4:20 - 4:38	Extending QUIC: transport parameters and extensions (Ack Frequency)
10	4:40 - 5:00	Open Discussion, Q & A

Wireshark QUIC Demo

Requires up-to-date Wireshark version (v3.3.0-rc)

Get it at: https://www.wireshark.org/download/automated

Requires traffic decryption KEYS

Most QUIC stacks support SSLKEYLOGFILE

https://wiki.wireshark.org/TLS

https://lekensteyn.nl/files/wireshark-tls-debugging-sharkfest19eu.pdf

Easy to get pcaps to play with via QUIC Interop Runner

https://interop.seemann.io

Wireshark currently lacks (full) HTTP/3 support

Should be added in the coming months (August-September 2020)

Plan

#	Start - End	Topic
1	1:40 - 1:58	QUIC’s intellectual heritage
2	2:00 - 2:18	QUIC handshake, headers, connection migration
3	2:20 - 2:38	Wireshark demo and tutorial
4	**2:40 - 2:58**	**QUIC streams, flow control, frames, packetization**
5	**3:00 - 3:18**	**BREAK**
6	3:20 - 3:38	qlog and qvis demo and tutorial
7	3:40 - 3:58	QUIC loss detection and congestion control: how different from TCP?
8	4:00 - 4:18	Build your own congestion controller. Code walkthrough: quicly and quiche
9	4:20 - 4:38	Extending QUIC: transport parameters and extensions (Ack Frequency)
10	4:40 - 5:00	Open Discussion, Q & A

Streams and Flow Control

Streams are a lightweight abstraction

each is a separate "ordered stream of bytes"
streams are independent wrt ordering/retransmission
⇒ QUIC "removes" Head-of-Line blocking
data needs to be multiplexed onto underlying connection

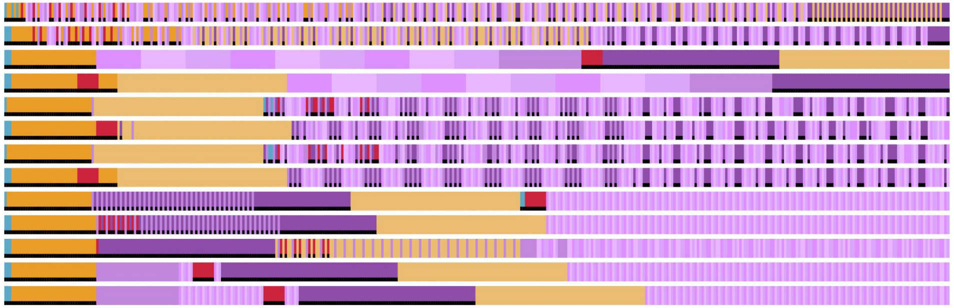
~~Two~~ Four types of stream

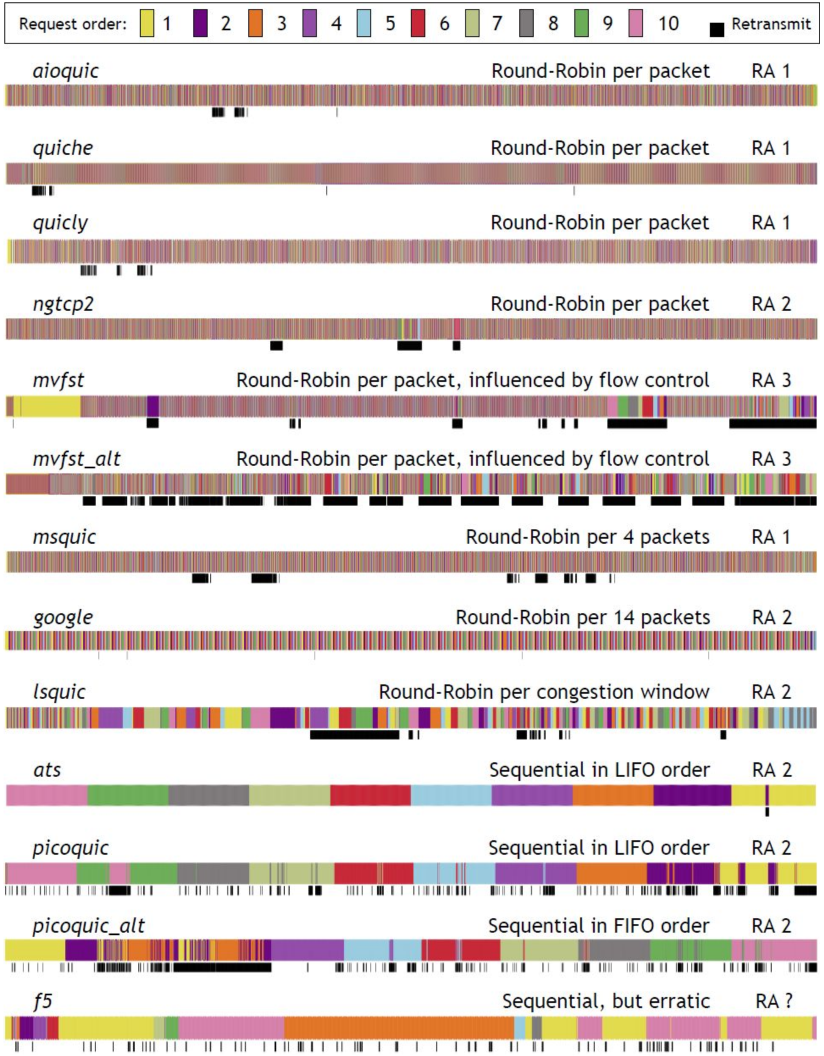
unidirectional stream	server initiated
bidirectional stream	client initiated

Receiver applies flow control to limit data sent in streams

stream flow control limits bytes sent on a stream
connection flow control limits bytes sent across all streams
stream *count* flow control limits amount of streams

Stream multiplexing and prioritization





https://h3.edm.uhasselt.be

QUIC Packet Format

Long header

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+
|1|1|T T|X X X X|
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                         Version (32)                          |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|DCIL(4)|SCIL(4)|
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|               Destination Connection ID (0/32..144)         ...
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                 Source Connection ID (0/32..144)            ...
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Short header

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+
|0|1|S|R|R|K|P P|
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                Destination Connection ID (0..144)           ...
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                     Packet Number (8/16/24/32)              ...
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                     Protected Payload (*)                   ...
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Frames

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                          Frame 1 (*)                        ...
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                          Frame 2 (*)                        ...
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
                               ...
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                          Frame N (*)                        ...
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Frames

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                          Frame 1 (*)                        ...
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                          Frame 2 (*)                        ...
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
                               ...
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                          Frame N (*)                        ...
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                       Frame Type (i)                        ...
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                   Type-Dependent Fields (*)                 ...
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Frames

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                          Frame 1 (*)                        ...
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                          Frame 2 (*)                        ...
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
                               ...
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                          Frame N (*)                        ...
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                      Frame Type (i)                         ...
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                 Type-Dependent Fields (*)                   ...
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Type Value	Frame Type Name
0x00	PADDING
0x01	PING
0x02 - 0x03	ACK
0x04	RESET_STREAM
0x05	STOP_SENDING
0x06	CRYPTO
0x07	NEW_TOKEN
0x08 - 0x0f	STREAM
0x10	MAX_DATA
0x11	MAX_STREAM_DATA
0x12 - 0x13	MAX_STREAMS
0x14	DATA_BLOCKED
0x15	STREAM_DATA_BLOCKED
0x16 - 0x17	STREAMS_BLOCKED
0x18	NEW_CONNECTION_ID
0x19	RETIRE_CONNECTION_ID
0x1a	PATH_CHALLENGE
0x1b	PATH_RESPONSE
0x1c - 0x1d	CONNECTION_CLOSE

Frames

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                          Frame 1 (*)                        ...
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                          Frame 2 (*)                        ...
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
                               ...
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                          Frame N (*)                        ...
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                       Frame Type (i)                        ...
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                 Type-Dependent Fields (*)                   ...
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Type Value	Frame Type Name
0x00	PADDING
0x01	PING
0x02 - 0x03	ACK
0x04	RESET_STREAM
0x05	STOP_SENDING
0x06	CRYPTO
0x07	NEW_TOKEN
0x08 - 0x0f	STREAM
0x10	MAX_DATA
0x11	MAX_STREAM_DATA
0x12 - 0x13	MAX_STREAMS
0x14	DATA_BLOCKED
0x15	STREAM_DATA_BLOCKED
0x16 - 0x17	STREAMS_BLOCKED
0x18	NEW_CONNECTION_ID
0x19	RETIRE_CONNECTION_ID
0x1a	PATH_CHALLENGE
0x1b	PATH_RESPONSE
0x1c - 0x1d	CONNECTION_CLOSE

Frames

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                          Frame 1 (*)                        ...
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                          Frame 2 (*)                        ...
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
                               ...
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                          Frame N (*)                        ...
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

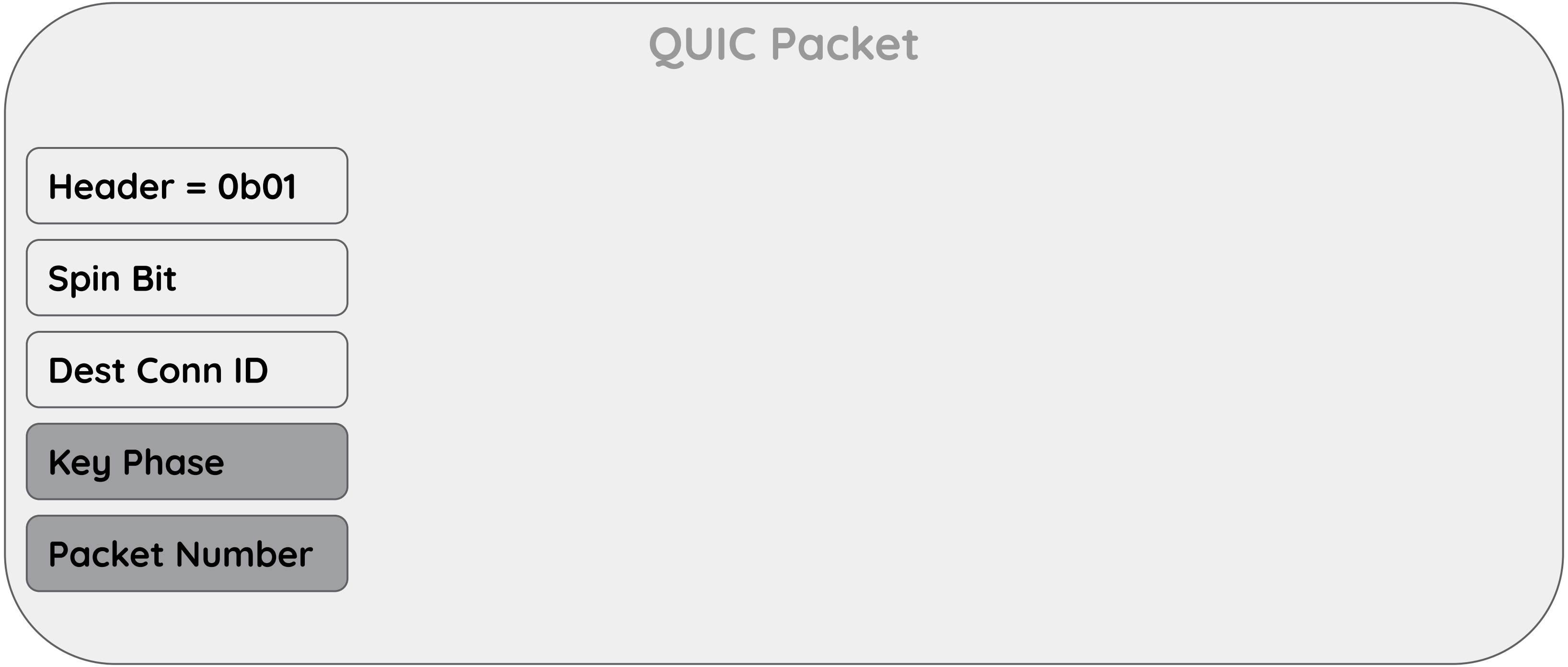
```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                       Frame Type (i)                        ...
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                 Type-Dependent Fields (*)                   ...
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Type Value	Frame Type Name
0x00	PADDING
0x01	PING
$\boxed{\text{0x02 - 0x03}}$	$\boxed{\text{ACK}}$
0x04	RESET_STREAM
0x05	STOP_SENDING
$\boxed{\text{0x06}}$	$\boxed{\text{CRYPTO}}$
0x07	NEW_TOKEN
$\boxed{\text{0x08 - 0x0f}}$	$\boxed{\text{STREAM}}$
0x10	MAX_DATA
0x11	MAX_STREAM_DATA
0x12 - 0x13	MAX_STREAMS
0x14	DATA_BLOCKED
0x15	STREAM_DATA_BLOCKED
0x16 - 0x17	STREAMS_BLOCKED
0x18	NEW_CONNECTION_ID
0x19	RETIRE_CONNECTION_ID
0x1a	PATH_CHALLENGE
0x1b	PATH_RESPONSE
0x1c - 0x1d	CONNECTION_CLOSE

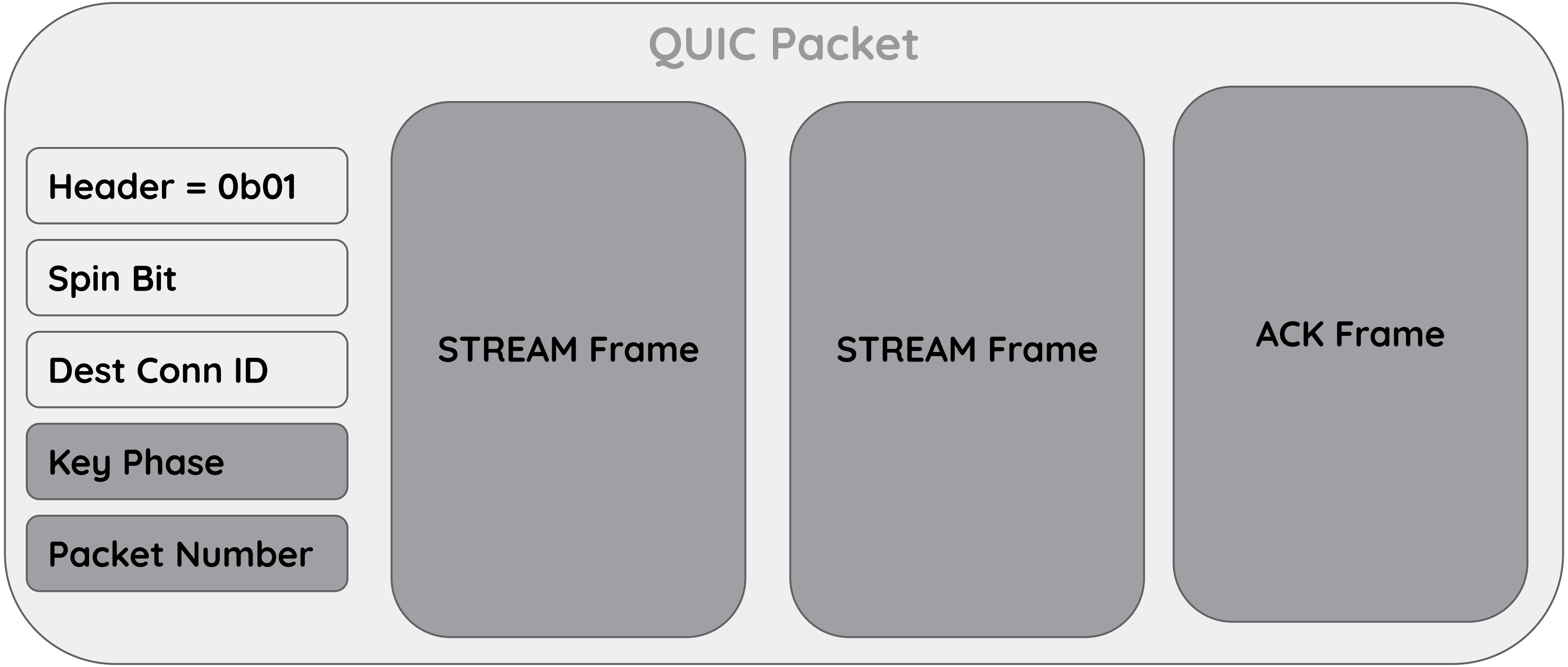
STREAM Frame

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                         Stream ID (i)                       ...
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                         [Offset (i)]                        ...
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                         [Length (i)]                        ...
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                        Stream Data (*)                      ...
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

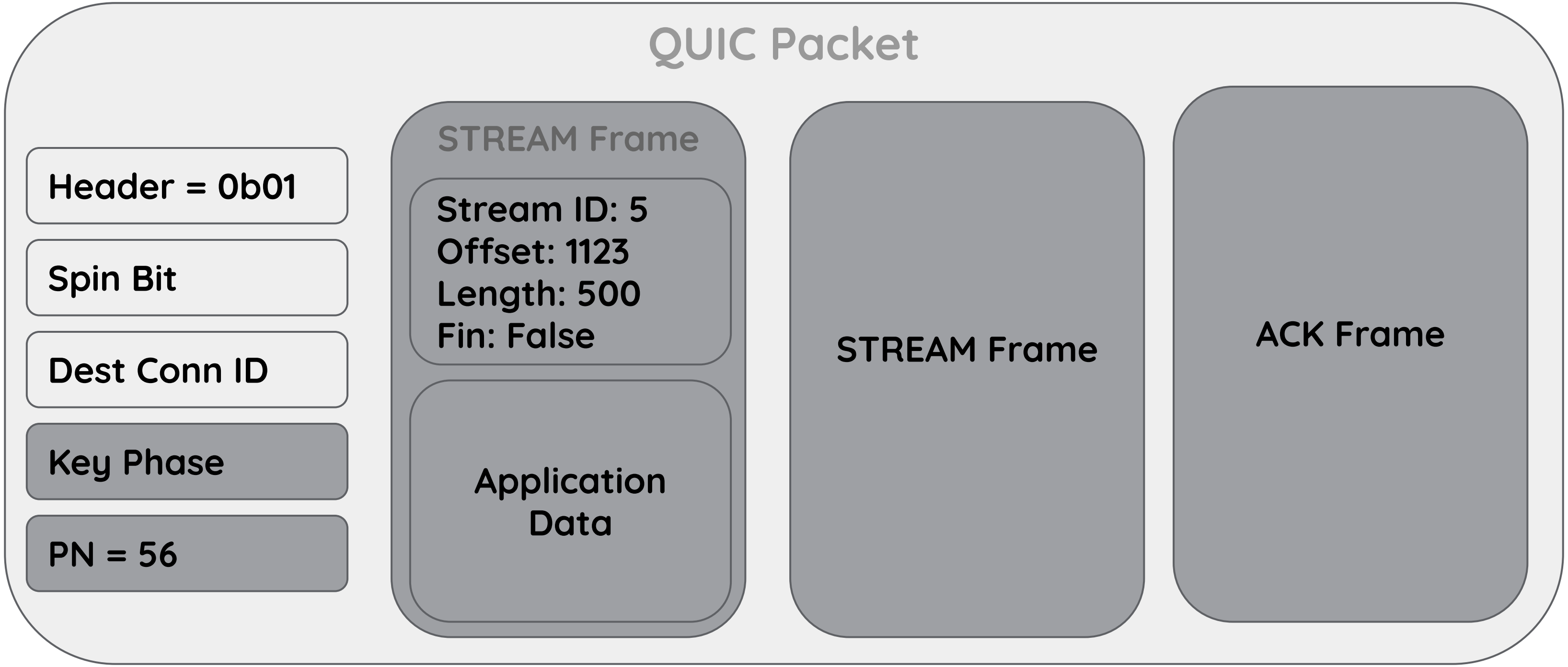
QUIC Packetization: Example



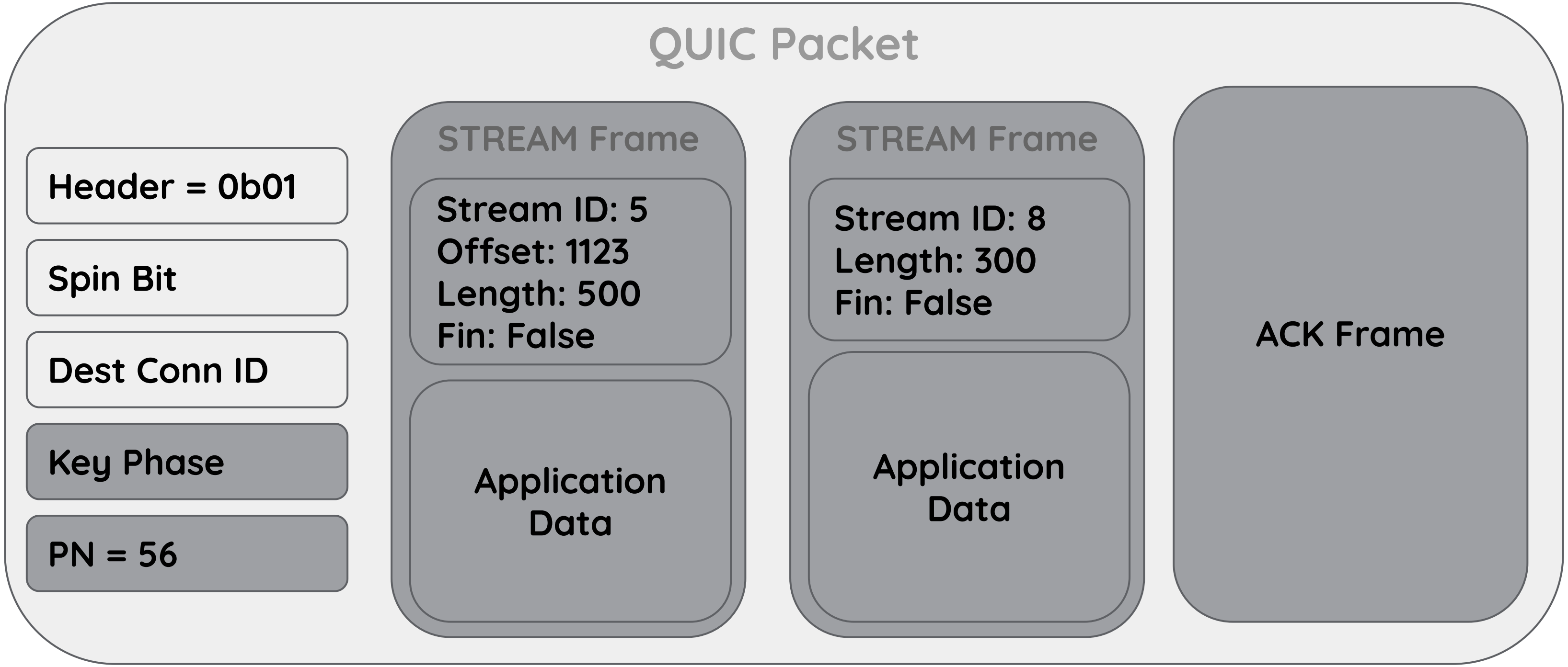
QUIC Packetization: Example



QUIC Packetization: Example



QUIC Packetization: Example



ACK Frame

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                     Largest Acknowledged (i)                ...
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                          ACK Delay (i)                      ...
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                       ACK Range Count (i)                   ...
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                       First ACK Range (i)                   ...
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                          ACK Ranges (*)                     ...
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                          [ECN Counts]                       ...
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Highest Packet Number seen so far

Time since Largest Acked was received

Contiguous from Largest Acked

QUIC Packetization: Ack Example

Packets received: 1 … 125
Time since largest received: 25ms

represented as a shifted value (default 3, negotiable)

25ms = 3125us << 3

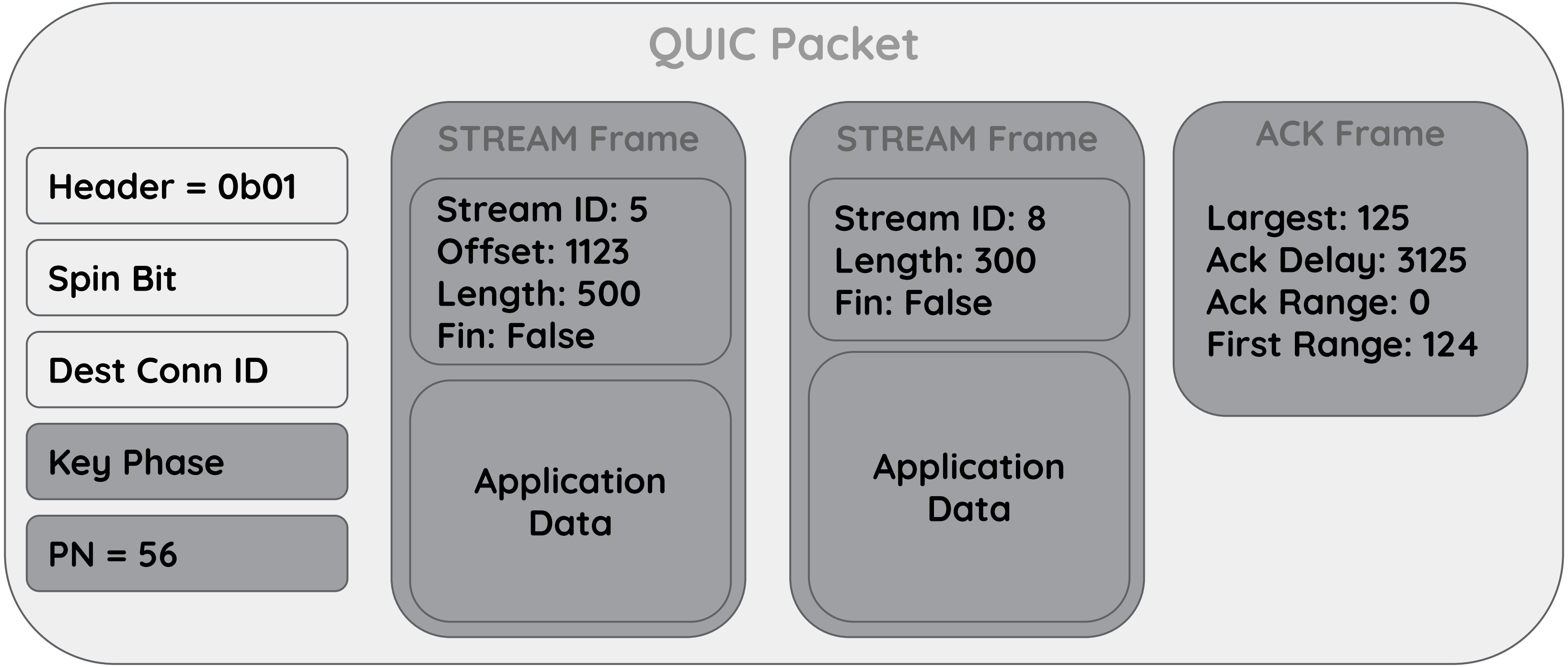
ACK fields

Largest packet received so far: 125

First Ack Range: 124

Ack Range Count: 0

QUIC Packetization: Example



QUIC Packetization: Loss Example

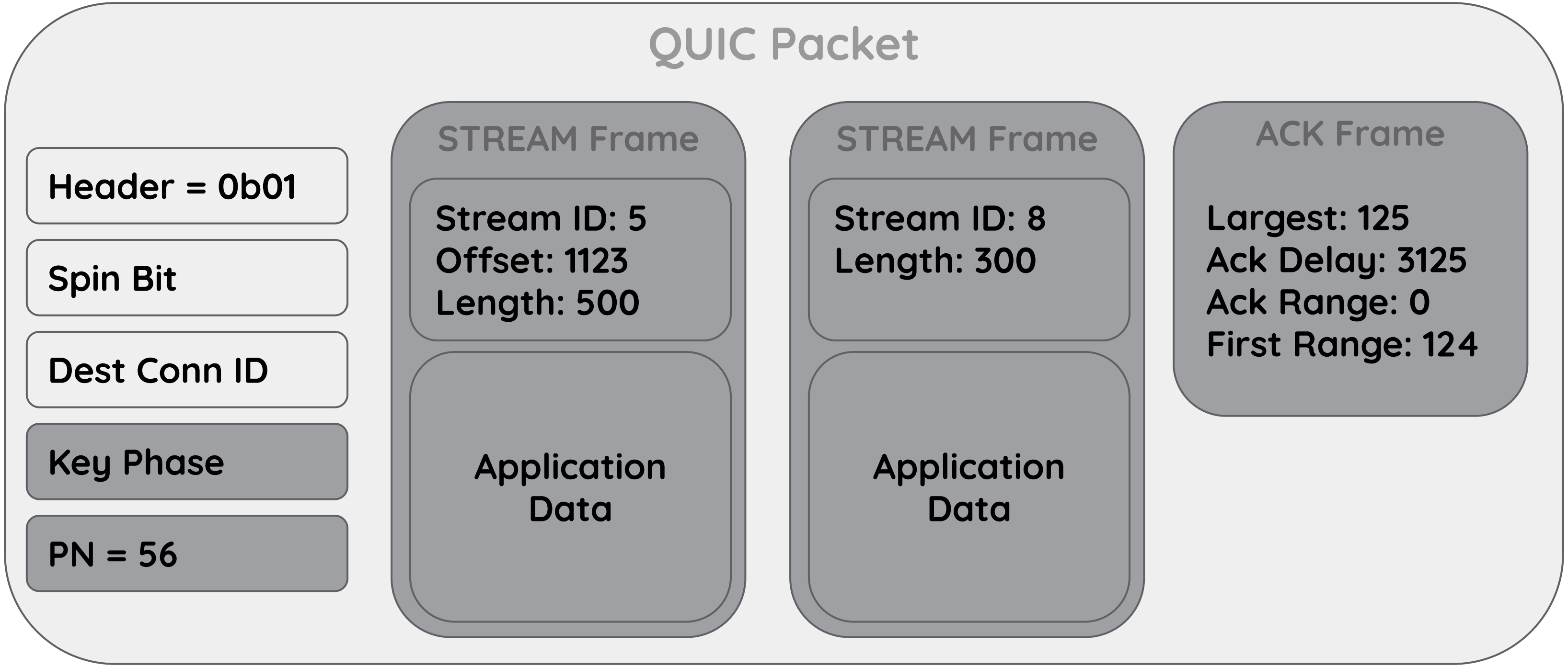
Packet 56 dropped

Also, Stream 8 was cancelled

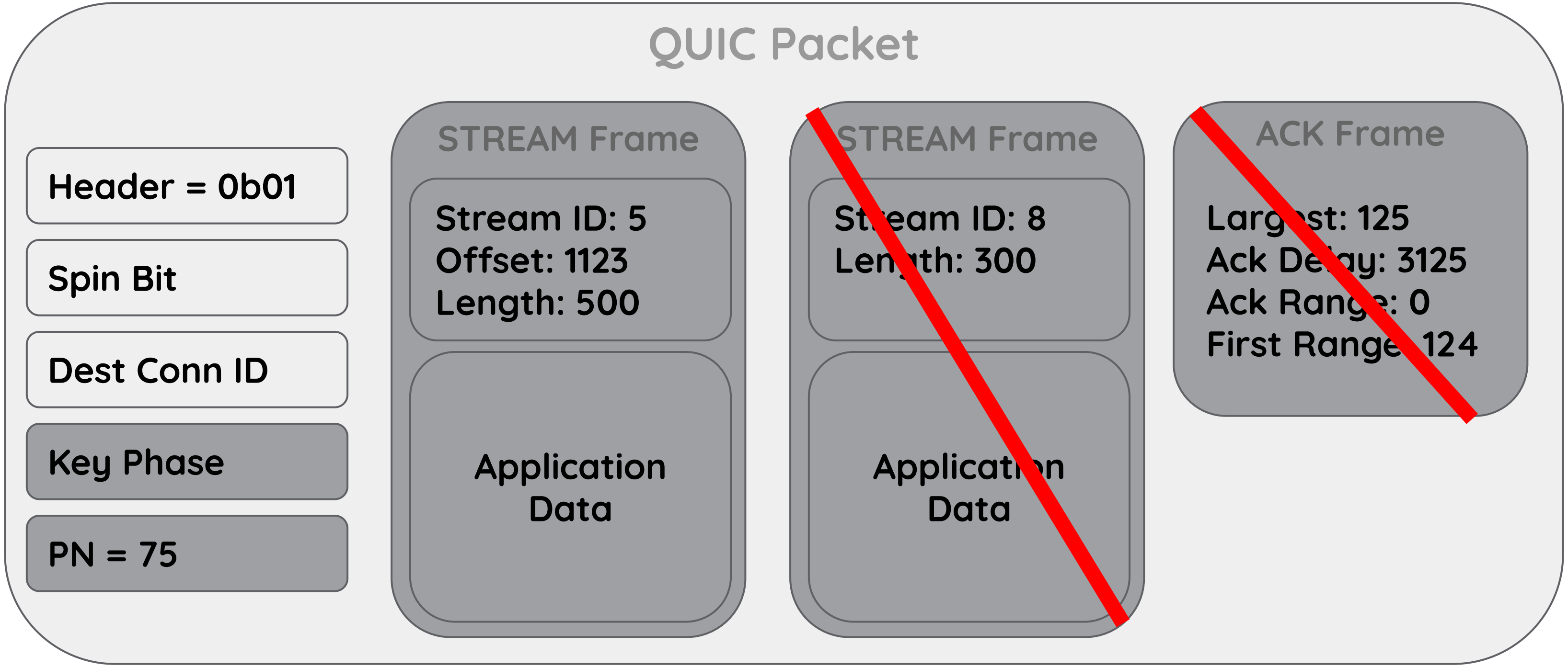
QUIC loss detection marks packet 56 as lost

- let’s say last packet sent was packet number 74

QUIC Packetization: Example



QUIC Packetization: Example



Plan

#	Start - End	Topic
1	1:40 - 1:58	QUIC's intellectual heritage
2	2:00 - 2:18	QUIC handshake, headers, connection migration
3	2:20 - 2:38	Wireshark demo and tutorial
4	2:40 - 2:58	QUIC streams, flow control, frames, packetization
5	**3:00 - 3:18**	**BREAK**
6	3:20 - 3:38	qlog and qvis demo and tutorial
7	3:40 - 3:58	QUIC loss detection and congestion control: how different from TCP?
8	4:00 - 4:18	Build your own congestion controller. Code walkthrough: quicly and quiche
9	4:20 - 4:38	Extending QUIC: transport parameters and extensions (Ack Frequency)
10	4:40 - 5:00	Open Discussion, Q & A

Plan

#	Start - End	Topic
1	1:40 - 1:58	QUIC's intellectual heritage
2	2:00 - 2:18	QUIC handshake, headers, connection migration
3	2:20 - 2:38	Wireshark demo and tutorial
4	2:40 - 2:58	QUIC streams, flow control, frames, packetization
5	**3:00 - 3:18**	**BREAK**
6	**3:20 - 3:38**	**qlog and qvis demo and tutorial**
7	3:40 - 3:58	QUIC loss detection and congestion control: how different from TCP?
8	4:00 - 4:18	Build your own congestion controller. Code walkthrough: quicly and quiche
9	4:20 - 4:38	Extending QUIC: transport parameters and extensions (Ack Frequency)
10	4:40 - 5:00	Open Discussion, Q & A

QUIC debugging challenges

Problem: End-to-end Encryption

Store full packet captures = large
Need to store decryption keys somewhere = insecure
Decryption shows -everything-, including user data = bad for privacy

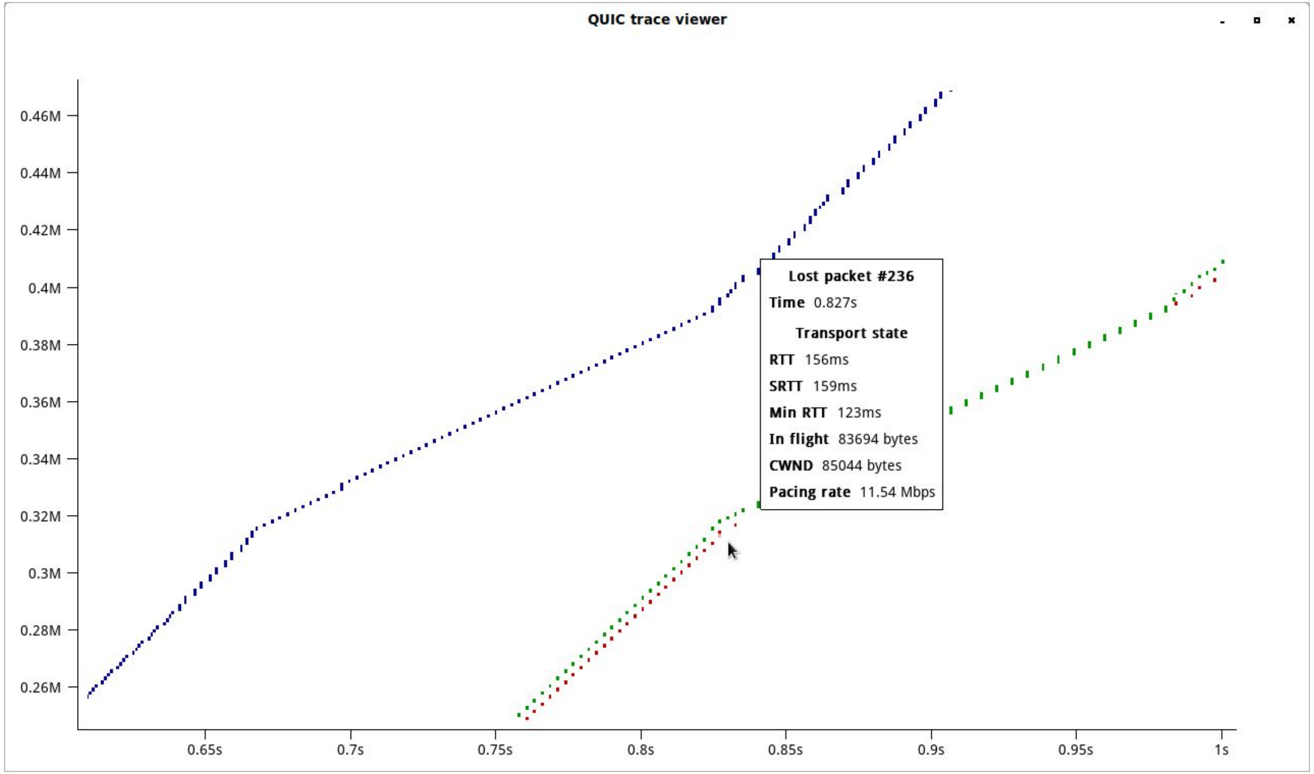
Solution: structured endpoint logging

Choose which data to log (manage file size + privacy)
Include internal state not sent on network

Still easy to create cross-implementation tooling

Network operators don't like this "solution"
- See spinbit, loss bits, etc.

QUIC tooling : quictrace



https://github.com/google/quic-trace

QUIC tooling demo : qlog and qvis

qlog: “ad-hoc standard” logging format

12/18 stacks support it, at least 3 more have plans
JSON-based
https://tools.ietf.org/html/draft-marx-qlog-main-schema-01

qvis: visualization toolsuite

5 different tools / visualizations
also supports pcap files and Chrome’s NetLog files
https://qvis.edm.uhasselt.be

Example files available at:

https://qlog.edm.uhasselt.be/sigcomm/tutorial.html

See also “Visualizing QUIC and HTTP/3 with qlog and qvis” demo Tuesday and Wednesday

Plan

#	Start - End	Topic
1	1:40 - 1:58	QUIC's intellectual heritage
2	2:00 - 2:18	QUIC handshake, headers, connection migration
3	2:20 - 2:38	Wireshark demo and tutorial
4	2:40 - 2:58	QUIC streams, flow control, frames, packetization
5	**3:00 - 3:18**	**BREAK**
6	3:20 - 3:38	qlog and qvis demo and tutorial
7	**3:40 - 3:58**	**QUIC loss detection and congestion control: how different from TCP?**
8	4:00 - 4:18	Build your own congestion controller. Code walkthrough: quicly and quiche
9	4:20 - 4:38	Extending QUIC: transport parameters and extensions (Ack Frequency)
10	4:40 - 5:00	Open Discussion, Q & A

Terminology

ACK - Acknowledgement, multiple different formats
Loss Detection - Detect which packets were lost
Recover - Retransmit lost data and have it acknowledged
RTO - Retransmission Timeout
TLP - Tail loss probe, fires before RTO
PTO - Probe Timeout: QUIC’s merge of TLP and RTO

Loss Detection and Loss Recovery

TCP - RFC793

Receiver sends an ACK every time a packet is received

- Increases ‘Acknowledgment Number’ when the data is received in order
- ‘Acknowledgement Number’ commonly known as Cumulative ACK

Sender retransmits following piece of data on 3 identical ACKs

- One round trip later, hopefully that(and more data) is acknowledged

Can only recover from one lost packet per round trip

RTO fires SRTT+4*RTTVar later if 3 dupacks are not received

- RTO collapses the congestion window to the min
- Immediately declares all packets lost

TCP SACK - RFC2018

Multiple byte ranges beyond the cumulative ACK offset

FACK (forward acknowledgement) loss detection

Loss is detected when a packet sent 3 packets later is ACKed OR 3 Dupack

Adaptive reordering thresholds common

```
                  +--------+--------+
                  | Kind=5 | Length |
+--------+--------+--------+--------+
|      Left Edge of 1st Block       |
+--------+--------+--------+--------+
|      Right Edge of 1st Block      |
+--------+--------+--------+--------+
|                                   |
/            . . .                  /
|                                   |
+--------+--------+--------+--------+
|      Left Edge of nth Block       |
+--------+--------+--------+--------+
|      Right Edge of nth Block      |
+--------+--------+--------+--------+
```

Recover from multiple losses in a round trip

TCP RACK - draft-ietf-tcpm-rack

Track transmission time of packets, in addition to sequence number

Avoids RTOs when retransmitted data is lost

Uses Time and Packet thresholds

Replaces Early Retransmit with a timer

First IETF spec to describe TLP

Previously only in a paper, though widely used

Built on existing TCP signals

QUIC - draft-ietf-quic-recovery

Key Differences

- Monotonically Increasing Packet Numbers
- ACK blocks instead of SACK blocks
- PTO replaces TLP and RTO
- ACK Delay and max_ack_delay
- Separate Packet Number Spaces
- Persistent Congestion

QUIC packet numbers are monotonic and unique

QUIC uses Packet Numbers, opposed to TCP Sequence Numbers

- Indicate transmission order, not delivery order
- Removes TCP and SCTP's **retransmission ambiguity**

QUIC packets are never* retransmitted

- Lost data or frames sent in a new packet

```
Short Header Packet {
    Header Form (1) = 0,
    Fixed Bit (1) = 1,
    Spin Bit (1),
    Reserved Bits (2),
    Key Phase (1),
    Packet Number Length (2),
    Destination Connection ID (0..160),
    Packet Number (8..32),
    Packet Payload (..),
}
```

Ack blocks

Sequence of N ACK blocks

- Number of blocks limited by datagram size
- Can detect a practically unlimited number of losses

Includes the most recently received packets

- Constant forward progress

No Reneging

- Simplifies implementations

```
ACK Frame {
      Type (i) = 0x02..0x03,
      Largest Acknowledged (i),
      ACK Delay (i),
      ACK Range Count (i),
      First ACK Range (i),
      ACK Range (..) ...,
      [ECN Counts (..)],
   }
```

PTO

Combines TLP and RTO into one mechanism

```
Period = smoothed_rtt + max(4*rttvar, kGranularity) +
           max_ack_delay
```

Updated every time a packet is sent or ACK is received

Set from the last ack-eliciting sent packet

Prefer sending new data to avoid spurious transmissions

ACK Delay and Maximum Ack Delay

ACK Delay in the ACK frame communicates introduced delay

Used in calculations of Smoothed RTT and RTTVar

max_ack_delay specifies the maximum intended ACK Delay

Communicated in Transport Parameters during the TLS handshake

Enables removing MinRTO

Similar to the TCP MAD

```
ACK Frame {
     Type (i) = 0x02..0x03,
     Largest Acknowledged (i),
     ACK Delay (i),
     ACK Range Count (i),
     First ACK Range (i),
     ACK Range (..) ...,
     [ECN Counts (..)],
   }
```

Separate Packet Number Spaces

QUIC has Initial, Handshake and ApplicationData packet number spaces

ApplicationData = 0-RTT and 1-RTT packets

After handshake confirmation, only ApplicationData is active

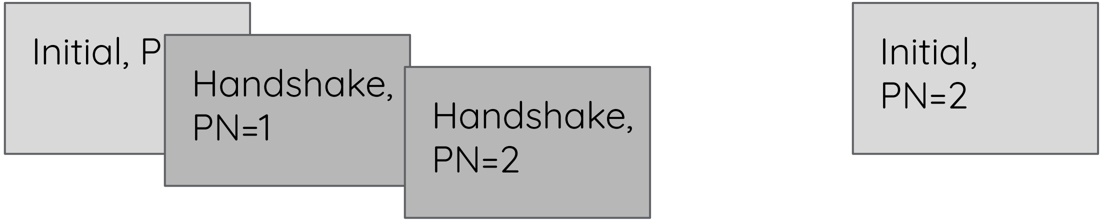
Loss detection is per-PN space

Loss detection requires the peer to acknowledge a subsequent packet

Acknowledging a packet requires decryption keys

Congestion control and RTT span PN spaces

Congestion control and RTT measurements are on a path



Congestion Control

NewReno style congestion control

AIMD - Additive increase, multiplicative decrease

Slow Start

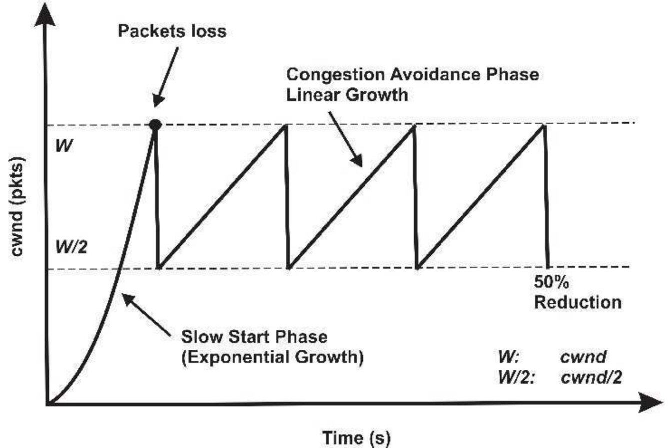
```
congestion_window += packet.size
```

Congestion avoidance (1 packet increase per acknowledged CWND)

```
congestion_window += max_datagram_size * packet.size
            / congestion_window
```

Upon Loss

```
congestion_window /= 2
```



Persistent Congestion replaces RTO response

PTO does not change the congestion window upon expiry

Unlike TCP's RTO, which collapses the congestion window

Instead, QUIC waits until packets are lost over 3 * PTO period

Similar to TCP sending TLPs twice before firing RTO

Why time instead of sequential PTOs?

Because the window isn't reduced, applications can continue to send

This can indefinitely delay the first PTO in some circumstances

Signals are generic

Signals

OnPacketSent
OnPacketsAcked
CongestionEvent - Upon lost packet or ECN

Other commonly implemented Congestion Controllers

Cubic
BBR
BBRv2

Plan

#	Start - End	Topic
1	1:40 - 1:58	QUIC’s intellectual heritage
2	2:00 - 2:18	QUIC handshake, headers, connection migration
3	2:20 - 2:38	Wireshark demo and tutorial
4	2:40 - 2:58	QUIC streams, flow control, frames, packetization
5	**3:00 - 3:18**	**BREAK**
6	3:20 - 3:38	qlog and qvis demo and tutorial
7	3:40 - 3:58	QUIC loss detection and congestion control: how different from TCP?
8	**4:00 - 4:18**	**Build your own congestion controller. Code walkthrough: quicly, quiche**
9	4:20 - 4:38	Extending QUIC: transport parameters and extensions (Ack Frequency)
10	4:40 - 5:00	Open Discussion, Q & A

Congestion Controllers in quicly (C)

Interface for congestion controllers:
https://github.com/h2o/quicly/blob/master/include/quicly/cc.h

Example implementations:
https://github.com/h2o/quicly/blob/master/lib/cc-reno.c
https://github.com/h2o/quicly/blob/master/lib/cc-cubic.c

Pacing is a TODO in quicly, might change the interface

Contributions welcome!
    Cubic is thanks to Leo Blöcher (Aachen University)

Congestion Controllers in Quicly (C)

```c
struct st_quicly_cc_impl_t {

   quicly_cc_type_t type;

   void (*cc_on_acked)(quicly_cc_t *cc, const quicly_loss_t *loss, uint32_t bytes,
                       uint64_t largest_acked, uint32_t inflight,
                       int64_t now, uint32_t max_udp_payload_size);

   void (*cc_on_lost)(quicly_cc_t *cc, const quicly_loss_t *loss, uint32_t bytes,
                      uint64_t lost_pn, uint64_t next_pn, int64_t now,
                      uint32_t max_udp_payload_size);

   void (*cc_on_persistent_congestion)(quicly_cc_t *cc, const quicly_loss_t *loss, int64_t now);
};
```

Congestion Controllers in Chromium Quiche(C++)

Events from SendAlgorithmInterface

```
virtual void OnPacketSent(QuicTime sent_time,
                          QuicByteCount bytes_in_flight,
                          QuicPacketNumber packet_number,
                          QuicByteCount bytes,
                          HasRetransmittableData is_retransmittable) = 0;


virtual void OnCongestionEvent(bool rtt_updated,
                               QuicByteCount prior_in_flight,
                               QuicTime event_time,
                               const AckedPacketVector& acked_packets,
                               const LostPacketVector& lost_packets) = 0;
```

Congestion Controllers in Chromium Quiche(C++)

When and how fast to send from [SendAlgorithmInterface]

```
virtual bool CanSend(QuicByteCount bytes_in_flight) = 0;
```

ie: `return bytes_in_flight < congestion_window`

```
virtual QuicBandwidth PacingRate(QuicByteCount bytes_in_flight) const = 0;
```

Window-based: `C*(congestion_window / smoothed_rtt)`
Bandwidth-based: `C*bandwidth`

Congestion Controllers in Chromium Quiche(C++)

BbrSender

ACM Queue 2016 Paper

Plan

#	Start - End	Topic
1	1:40 - 1:58	QUIC's intellectual heritage
2	2:00 - 2:18	QUIC handshake, headers, connection migration
3	2:20 - 2:38	Wireshark demo and tutorial
4	2:40 - 2:58	QUIC streams, flow control, frames, packetization
5	**3:00 - 3:18**	**BREAK**
6	3:20 - 3:38	qlog and qvis demo and tutorial
7	3:40 - 3:58	QUIC loss detection and congestion control: how different from TCP?
8	4:00 - 4:18	Build your own congestion controller. Code walkthrough: quicly and quiche
9	**4:20 - 4:38**	**Extending QUIC: transport parameters and extensions (Ack Frequency)**
10	4:40 - 5:00	Open Discussion, Q & A

Extending QUIC

QUIC uses transport parameters to negotiate extensions

- They may be a simple bool indicating support
- Or they one or more values

New Frames in QUIC MUST be negotiated

- Otherwise the peer will close the connection upon receipt

Example 1: Changing the ACK frequency

QUIC follows TCP RFC 5681

- Recommends ACK every 2 packets

In practice, ACK collapsing (thinning) is widespread for TCP

- at endhosts
- by middleboxes

These optimizations are critical for

- high bandwidth links
- highly asymmetric links (satellite)

QUIC packets are encrypted, so middleboxes can't do it

Proposal

Sender: Sender of ack-eliciting packets
Receiver: Sender of ACK-only frames in response
Assumption:

- Receiver is naturally incented to ACK minimally
- Sender is naturally incented to process fewer ACKs
- Sender knows its controller’s tolerance / desire

Design: Frame sent from **Sender** to **Receiver** to change receiver’s ACK behavior

Draft: draft-iyengar-quic-delayed-ack

Negotiating with a Transport Parameter

Transport Parameter: `min_ack_delay (0xde1a)`
the minimum amount of time (in microseconds)
by which the endpoint can delay an acknowledgement

Used for negotiating use of this extension

ACK_FREQUENCY Frame

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                          0xAF (i)                          ...
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                      Sequence Number (i)                   ...
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                      Packet Tolerance (i)                  ...
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                    Update Max Ack Delay (i)                ...
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| Ignore Order (8)|
+-+-+-+-+-+-+-+-+-+
```

0xAF : Frame Type

Example 2: Unreliable sub-packet payloads

QUIC provides and HTTP/3 uses **Streams**

Streams are by default reliable

DATAGRAM is a way to transport data which is unreliable by default

Limited to what fits into a single packet

Congestion controlled, but not flow-controlled

Draft: [draft-ietf-quic-datagram]

Example 2: Negotiating

Transport Parameter: `max_datagram_frame_size(0x0020)`

Maximum datagram frame size in bytes

DATAGRAM Frame

```
0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                        0x0020 (i)                        …
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                       [Length (i)]                         ...
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                    Datagram Data (*)                       ...
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

0x0020 : Frame Type

Plan

#	Start - End	Topic
1	1:40 - 1:58	QUIC’s intellectual heritage
2	2:00 - 2:18	QUIC handshake, headers, connection migration
3	2:20 - 2:38	Wireshark demo and tutorial
4	2:40 - 2:58	QUIC streams, flow control, frames, packetization
5	**3:00 - 3:18**	**BREAK**
6	3:20 - 3:38	qlog and qvis demo and tutorial
7	3:40 - 3:58	QUIC loss detection and congestion control: how different from TCP?
8	4:00 - 4:18	Build your own congestion controller. Code walkthrough: quicly and quiche
9	4:20 - 4:38	Extending QUIC: transport parameters and extensions (Ack Frequency)
10	**4:40 - 5:00**	**Open Discussion, Q & A**