Advanced Computer Networks

CS740

Data Center Network Applications (II)

https://pages.cs.wisc.edu/~mgliu/CS740/F25/index.html

Ming Liu
mgliu@cs.wisc.edu

Outline

- Last lecture
  - Data Center Network Applications (I)

- Today
  - Data Center Network Applications (II)

- Announcements
  - In-class Exam 11/20/2025
  - Project Presentation on 12/04/2025 and 12/09/2025

Which application does this paper target?

Which application does this paper target?

Web Transport

Traditional HTTPS Stack

HTTP

TLS

TCP

IP

Traditional HTTPS Stack

HTTP

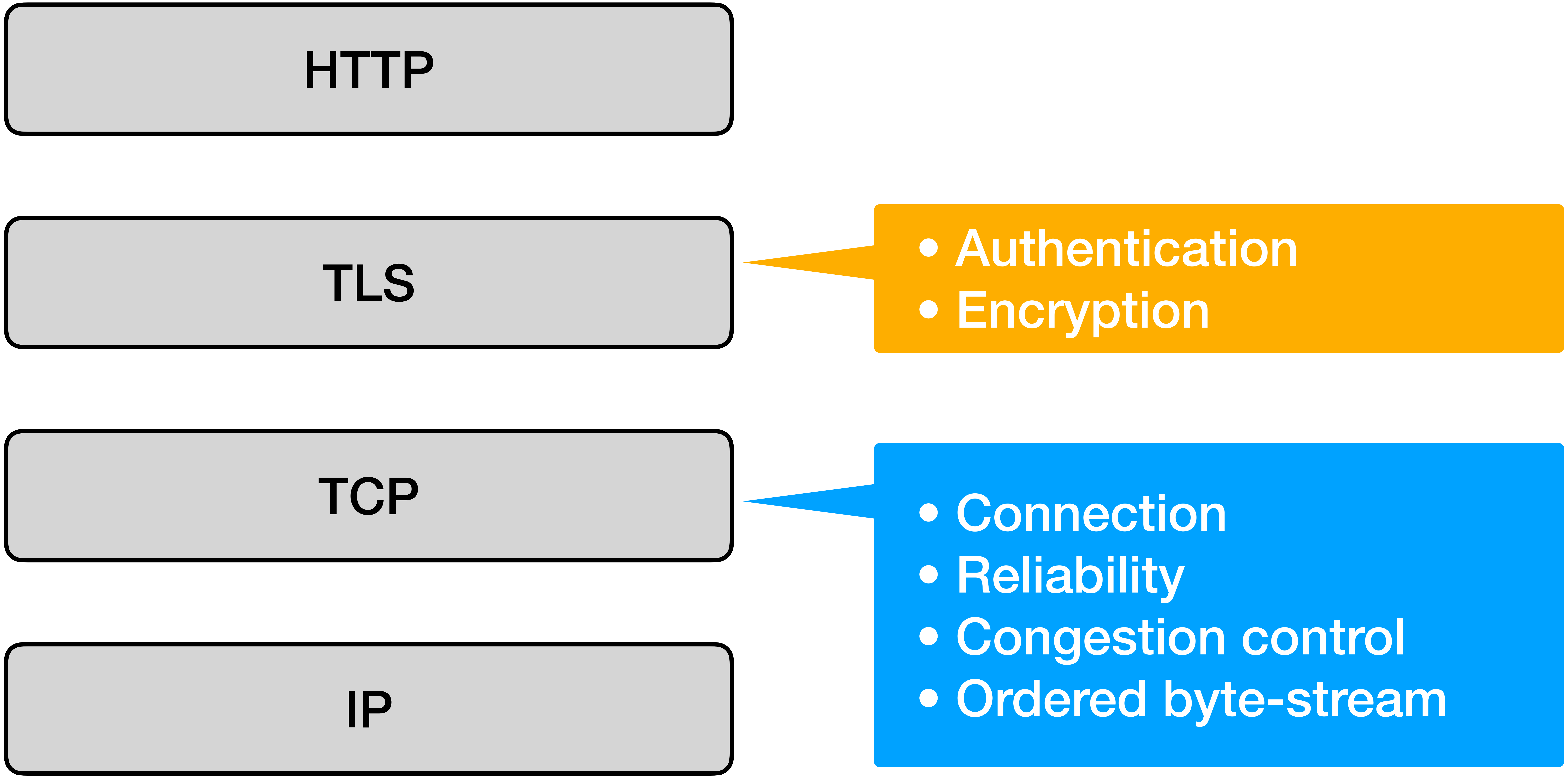
TLS

TCP

IP

- Connection
- Reliability
- Congestion control
- Ordered byte-stream

Traditional HTTPS Stack



Traditional HTTPS Stack

HTTP

- Transfer objects
- Require low latency

TLS

- Authentication
- Encryption

TCP

- Connection
- Reliability
- Congestion control
- Ordered byte-stream

IP

The HTTP Story

- Hypertext Transfer Protocol
  - Initiated by Tim Berners-Lee in 1989

The HTTP Story

- Hypertext Transfer Protocol
  - Initiated by Tim Berners-Lee in 1989

- HTTP/0.9
  - Only support the GET method

The HTTP Story

- Hypertext Transfer Protocol
  - Initiated by Tim Berners-Lee in 1989
- HTTP/0.9
  - Only support the GET method
- HTTP/1.0
  - Independent file transfers (open, write, close)

The HTTP Story

- Hypertext Transfer Protocol
  - Initiated by Tim Berners-Lee in 1989
- HTTP/0.9
  - Only support the GET method
- HTTP/1.0
  - Independent file transfers (open, write, close)
- HTTP/1.1
  - Connection persistence
  - Pipelining

However,

Final HTTP-NG Activity Statement

W3C's work on HTTP Next Generation (HTTP-NG) has been 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

Between July '97 and Dec '98, the HTTP-NG Activity explored the future development of the HTTP protocol. The motivation was the impression that HTTP/1.1 is becoming strained modularity wise as well as performance wise. The HTTP-NG Activity produced a number of proposals that successfully addressed these issues, which were presented to W3C members and at an IETF meeting in Dec. 98. At the moment, W3C does not plan any follow-up work on HTTP-NG.

Role of W3C

The work on HTTP-NG has been done at W3C by Henrik Frystyk Nielsen, Jim Gettys and Daniel Veillard who worked with other researchers from a number of companies and organizations. All the documents produced by the Protocol Design Group are already available as either W3C Technical Reports or IETF Internet Drafts.

Current Situation

At the IETF in Orlando, December 1998, we presented the initial work described in the Internet Draft " HTTP-NG Overview: Problem Statement, Requirements, and Solution Outline" along with the following IETF Internet Drafts:

- HTTP-NG Architectural Model Working Draft, 10th July 1998. This document defines a simple model for what an HTTP-ng architecture might look like, along with a set of terms for referring to parts of it.
- HTTP-NG Web Interfaces, Working Draft, 10th July 1998. This draft document describes an initial set of extensible formal object interfaces.
- HTTP-NG Binary Wire Protocol Working Draft, 10th July 1998. This document describes a binary `on-the-wire' protocol to be used when sending HTTP-NG operation invocations or terminations across a network connection.
- WebMUX Protocol Specification Working Draft, 10 July 1998. This document describes an experimental design for a multiplexing transport, intended for, but not restricted to, use with the Web.

While there was interest in the Project, the general feeling was that it was too early to bring it to IETF and that we needed to provide a plan for how to get where are today to where we would like to be. One of the arguments that we got was that people were just getting used to HTTP/1.1 and saw HTTP-NG as a 'warm-hole' into a very different Web infrastructure than what we have today.

Two results came out of this:

However,

W3C Architecture domain Activity STATEMENT

Final HTTP-NG Activity Statement

W3C's work on HTTP Next Generation (HTTP-NG) has been 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

Between July '97 and Dec '98, the HTTP-NG Activity explored the future development of the HTTP protocol. The motivation was the impression that HTTP/1.1 is becoming strained modularity wise as well as performance wise. The HTTP-NG Activity produced a number of proposals that successfully addressed these issues, which were presented to W3C members and at an IETF meeting in Dec. 98. At the moment, W3C does not plan any follow-up work on HTTP-NG.

Role of W3C

The work on HTTP-NG has been done at W3C by Henrik Frystyk Nielsen, Jim Gettys and Daniel Veillard who worked with other researchers from a number of companies and organizations. All the documents produced by the Protocol Design Group are already available as either W3C Technical Reports or IETF Internet Drafts.

Current Situation

At the IETF in Orlando, December 1998, we presented the initial work described in the Internet Draft " HTTP-NG Overview: Problem Statement, Requirements, and Solution Outline" along with the following IETF Internet Drafts:

- HTTP-NG Architectural Model Working Draft, 10th July 1998. This document defines a simple model for what an HTTP-ng architecture might look like, along with a set of terms for referring to parts of it.
- HTTP-NG Web Interfaces, Working Draft, 10th July 1998. This draft document describes an initial set of extensible formal object interfaces.
- HTTP-NG Binary Wire Protocol Working Draft, 10th July 1998. This document describes a binary `on-the-wire' protocol to be used when sending HTTP-NG operation invocations or terminations across a network connection.
- WebMUX Protocol Specification Working Draft, 10 July 1998. This document describes an experimental design for a multiplexing transport, intended for, but not restricted to, use with the Web.

While there was interest in the Project, the general feeling was that it was too early to bring it to IETF and that we needed to provide a plan for how to get where are today to where we would like to be. One of the arguments that we got was that people were just getting used to HTTP/1.1 and saw HTTP-NG as a 'warm-hole' into a very different Web infrastructure than what we have today.

Two results came out of this:

Some Fix

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 fo 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.

It is likely that the Web has caused the average packet train length on the Internet to decrease significantly over the last 2-3 years. Results from [13] and [21] indicate that sending fewer big packets is more cost effective than sending more small packets due to less overhead in routers and hosts. By multiplexing multiple lightweight HTTP transactions onto the same underlying transport connection and deploying smart output buffer management, small packets can to a large extend be avoided.

Performance Anomalies

#1: "[…] poor performance due to […] round trips required to open and close each TCP connection"

#2: "[…] does not adequately support simultaneous rendering of inlined objects"

#3: "[…] nor does it provide suitable fairness between protocol flows"

#4: "[…] or allow for graceful abortion of HTTP transactions without closing the TCP connection"

#5: "[…] do not share congestion information across multiple simultaneous connections"

Root Causes

#1: “[…] poor perfo **Handshake latency** *to open and close each TCP connecti*

#2: “[…] does not adequately support simultaneous rendering of inlined objects”

#3: “[…] nor does it provide suitable fairness between protocol flows”

#4: “[…] or allow for graceful abortion of HTTP transactions without closing the TCP connection”

#5: “[…] do not share congestion information across multiple simultaneous connections”

Root Causes

#1: “[…] poor perfo **Handshake latency** *to open and close each TCP connecti*

#2: “[…] does not a **Parallelism** *ing of inlined objects”*

#3: “[…] nor does it provide suitable fairness between protocol flows”

#4: “[…] or allow for graceful abortion of HTTP transactions without closing the TCP connection”

#5: “[…] do not share congestion information across multiple simultaneous connections”

Root Causes

#1: "[…] poor perfo **Handshake latency** *to open and close each TCP connecti*

#2: "[…] does not a **Parallelism** *ring of inlined objects"*

#3: "[…] nor does **Scheduling** *tocol flows"*

#4: "[…] or allow for graceful abortion of HTTP transactions without closing the TCP connection"

#5: "[…] do not share congestion information across multiple simultaneous connections"

Root Causes

#1: "[…] poor perfo | **Handshake latency** | *to open and close each TCP connecti*

#2: "[…] does not a | **Parallelism** | *ring of inlined objects"*

#3: "[…] nor does | **Scheduling** | *tocol flows"*

#4: "[…] or allow fo | **Request cancellation** | *ns without closing the TCP connection"*

#5: "[…] do not share congestion information across multiple simultaneous connections"

Root Causes

#1: “[…] poor perfo **Handshake latency** *to open and close each TCP connecti*

#2: “[…] does not a **Parallelism** *ring of inlined objects”*

#3: “[…] nor does **Scheduling** *tocol flows”*

#4: “[…] or allow f **Request cancellation** *s without closing the TCP connection”*

#5: “[…] do not sha **Many congestion controllers** *ple simultaneous connections”*

Root Causes

#1: “[…] poor perfo… to open and close each TCP connecti…

Handshake latency

The need: “[…] mutiplexing multiple lightweight HTTP transactions onto the same underlying transport connection and deploying smart output buffer management”

Many congestion controllers

connections”

Can we optimize the transport layer to support HTTP better?

The Transport Story

- T/TCP (Transactional Transmission Control Protocol)
  - By Bob Branden in 1994

```
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 (Transactional Transmission Control Protocol)
  - By Bob Branden in 1994

- TCP Session
  - By Venkata N. Padmanabhan in 1997

The Transport Story

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.

This drawback of P-HTTP led us to develop a comprehensive solution, which has two components. The first component, ***TCP session***, 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, thereby combining the flexibility of separate connections with the performance efficiency of a shared connection. This integration decreases download time by up to a factor of ten compared to HTTP/1.0 layered on standard TCP. TCP session does not alter TCP's messaging semantics, so deployment only involves local changes at the sender.

The second component of our solution, ***TCP fast start***, improves bandwidth utilization for short transfers by reusing information about the network conditions cached in the recent past. To avoid adverse effects in case the cached information is stale, TCP fast start exploits priority dropping at routers, and augments TCP's loss recovery mechanisms to quickly detect and abort a failed fast start attempt.

In addition to the two challenges we have discussed, a third challenge arises from the increasing deployment of ***asymmetric access networks***. Although Web browsing has asymmetric bandwidth requirements, bandwidth asymmetry could adversely impact Web download performance due to a disruption in the flow of acknowledgement feedback that is critical to sustaining good TCP throughput. To avoid performance degradation, we have developed end-host and router-based techniques, that both reduce disruption in the feedback and reduce TCP's dependence on such feedback. In certain situations, these techniques help decrease download time by a factor of fifteen.

This thesis includes mathematical and trace-based analysis, simulation, implementation and performance evaluation. In addition to the algorithms that we have designed and software that we have developed, our contributions include a set of paradigms for advancing the state-of-the-art in Internet transport protocols. These paradigms include the use of shared state and/or persistent state, and the exploitation of differentiated services mechanisms in routers.

The Transport Story

- T/TCP (Transactional Transmission Control Protocol)
  - By Bob Branden in 1994
- TCP Session
  - By Venkata N. Padmanabhan in 1997
- Congestion Manager
  - By Hari Balakrishnan in 1998

The Transport Story

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.

"[…] 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 (Transactional Transmission Control Protocol)
  - By Bob Branden in 1994
- TCP Session
  - By Venkata N. Padmanabhan in 1997
- Congestion Manager
  - By Hari Balakrishnan in 1998
- SCTP (Stream Control Transmission Protocol)
  - By IEFT in 2000

The Transport Story

- T/TCP (Transa… …col)
  - By Bob Brande…
- TCP Session
  - By Venkata N.
- Congestion Ma…
  - By Hari Balakri…
- SCTP (Stream…
  - By IEFT in 200…

```
Network Working Group                                    R. Stewart, Ed.
Request for Comments: 4960                                September 2007
Obsoletes: 2960, 3309
Category: Standards Track

                 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.

Abstract

   This document obsoletes RFC 2960 and RFC 3309.  It describes the
   Stream Control Transmission Protocol (SCTP).  SCTP is designed to
   transport Public Switched Telephone Network (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:

   --  acknowledged error-free non-duplicated transfer of user data,

   --  data fragmentation to conform to discovered path MTU size,

   --  sequenced delivery of user messages within multiple streams, with
       an option for order-of-arrival delivery of individual user
       messages,

   --  optional bundling of multiple user messages into a single SCTP
       packet, and
```

The In-Network Challenge

- Network changes in the mid-1990s
  - Network Address Translators (NATs): IP address scarcity
  - Firewalls: Protecting and policy
  - Protocol accelerators: transfer performance accelerations

The In-Network Challenge

- Network changes in the mid-1990s
  - Network Address Translators (NATs): IP address scarcity
  - Firewalls: Protecting and policy
  - Protocol accelerators: transfer performance accelerations

- Network devices started to read/modify end-to-end information
  - NATs: transport port number, checksum
  - Others: most transport header fields, state machine

The Middlebox

- Architectural control points of the Internet
  - By Lixia Zhang (1999)

```
Network Working Group                                      B. Carpenter
Request for Comments: 3234               IBM Zurich Research Laboratory
Category: Informational                                         S. Brim
                                                          February 2002


                   Middleboxes: Taxonomy and Issues

Status of this Memo

   This memo provides information for the Internet community.  It does
   not specify an Internet standard of any kind.  Distribution of this
   memo is unlimited.

Copyright Notice
```

A middlebox is defined as any 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.

The Transport Story Continued

- SST (Structured Stream Transport)
  - By Byran Ford et al. in Sigcomm’07
  - UDP-based
- Minion
  - By Byran Ford et al. in NSDI’12
  - Unordered Delivery Wire-Compatible with TCP and TLS
- TCP Fast Open
  - By Sivasankar Radhakrishnan et al. in CoNEXT’12
  - Accelerate the opening of successive TCP connections
- MPTCP
  - By Damon Wischik et al. in NSDI’11
  - Enable multi-path TCP connections

The HTTP Story Continued

- SPDY
  - By Google in 2009

The Chromium Projects

Search ⌘ K

Home
Chromium
ChromiumOS

Quick links
Report bugs
Discuss

Other sites
Chromium Blog
Google Chrome
Extensions

Privacy

Edit this page

SPDY >

SPDY: An experimental protocol for a faster web

Executive summary

As part of the "Let's make the web faster" initiative, we are experimenting with alternative protocols to help reduce the latency of web pages. One of these experiments is SPDY (pronounced "SPeeDY"), an application-layer protocol for transporting content over the web, designed specifically for minimal latency. In addition to a specification of the protocol, we have developed a SPDY-enabled Google Chrome browser and open-source web server. In lab tests, we have compared the performance of these applications over HTTP and SPDY, and have observed up to 64% reductions in page load times in SPDY. We hope to engage the open source community to contribute ideas, feedback, code, and test results, to make SPDY the next-generation application protocol for a faster web.

Background: web protocols and web latency

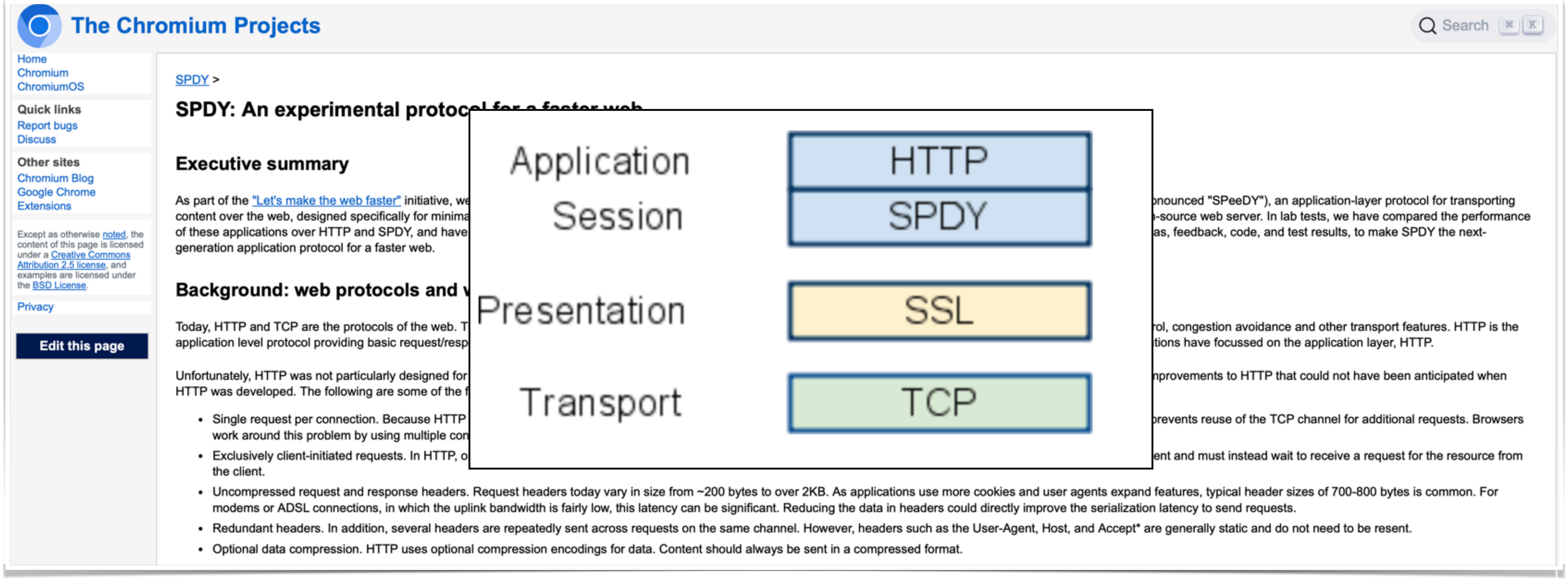
Today, HTTP and TCP are the protocols of the web. TCP is the generic, reliable transport protocol, providing guaranteed delivery, duplicate suppression, in-order delivery, flow control, congestion avoidance and other transport features. HTTP is the application level protocol providing basic request/response semantics. While we believe that there may be opportunities to improve latency at the transport layer, our initial investigations have focussed on the application layer, HTTP.

Unfortunately, HTTP was not particularly designed for latency. Furthermore, the web pages transmitted today are significantly different from web pages 10 years ago and demand improvements to HTTP that could not have been anticipated when HTTP was developed. The following are some of the features of HTTP that inhibit optimal performance:

- Single request per connection. Because HTTP can only fetch one resource at a time (HTTP pipelining helps, but still enforces only a FIFO queue), a server delay of 500 ms prevents reuse of the TCP channel for additional requests. Browsers work around this problem by using multiple connections. Since 2008, most browsers have finally moved from 2 connections per domain to 6.
- Exclusively client-initiated requests. In HTTP, only the client can initiate a request. Even if the server knows the client needs a resource, it has no mechanism to inform the client and must instead wait to receive a request for the resource from the client.
- Uncompressed request and response headers. Request headers today vary in size from ~200 bytes to over 2KB. As applications use more cookies and user agents expand features, typical header sizes of 700-800 bytes is common. For modems or ADSL connections, in which the uplink bandwidth is fairly low, this latency can be significant. Reducing the data in headers could directly improve the serialization latency to send requests.
- Redundant headers. In addition, several headers are repeatedly sent across requests on the same channel. However, headers such as the User-Agent, Host, and Accept* are generally static and do not need to be resent.
- Optional data compression. HTTP uses optional compression encodings for data. Content should always be sent in a compressed format.

The HTTP Story Continued

- SPDY
  - By Google in 2009



The HTTP Story Continued

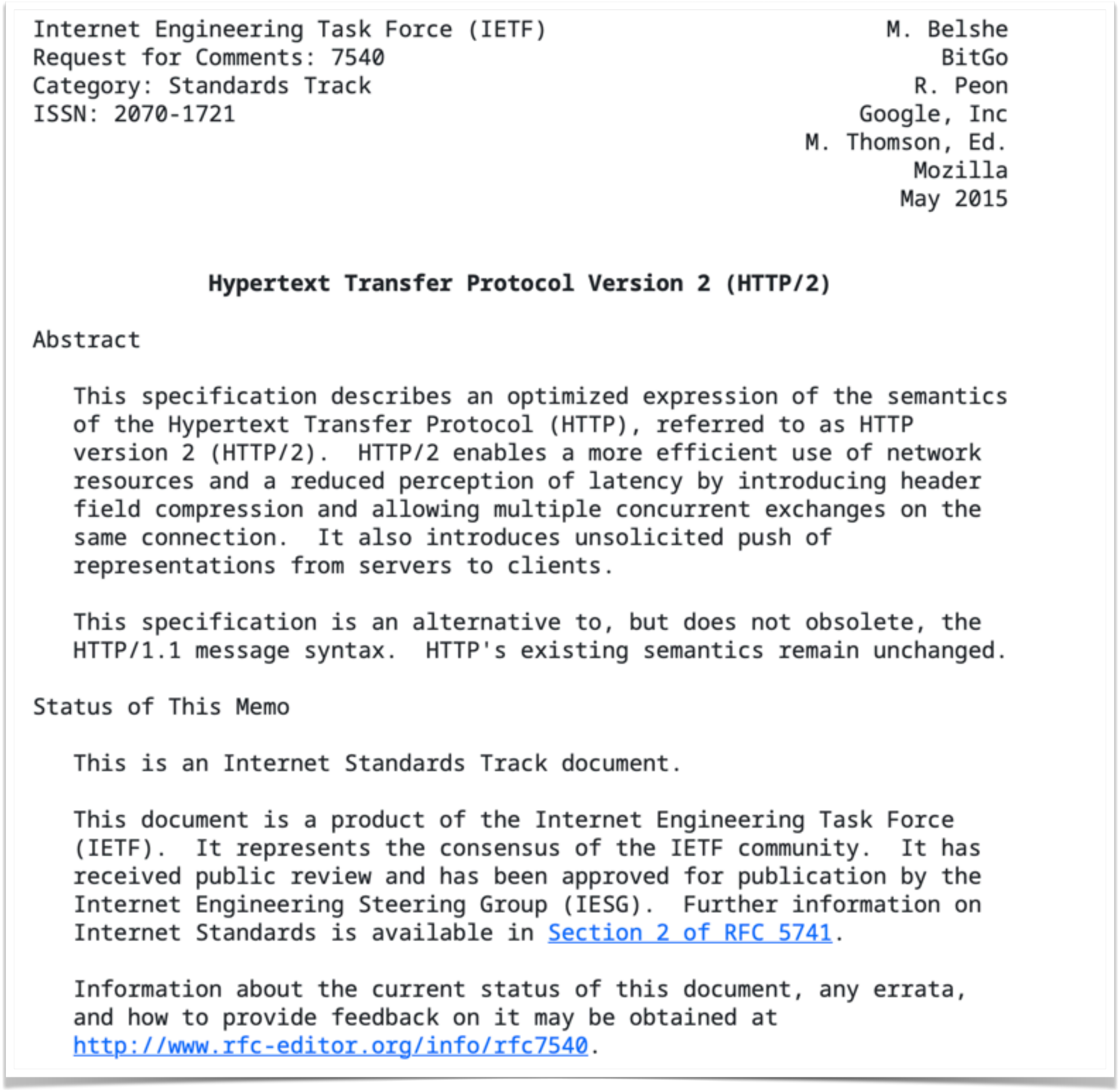
- SPDY
  - By Google in 2009
- SPDY Features
  - Streams
  - Multiplexing
  - Flow control
  - Priorities

The HTTP Story Continued

- SPDY
  - By Google in 2009
- SPDY Features
  - Streams
  - Multiplexing
  - Flow control
  - Priorities
- HTTP/2
  - Derived from SPDY

The HTTP Story Continued

- SPDY
  - By Google in 2009
- SPDY Features
  - Streams
  - Multiplexing
  - Flow control
  - Priorities
- HTTP/2
  - Derived from SPDY



Internet Engineering Task Force (IETF) M. Belshe
Request for Comments: 7540 BitGo
Category: Standards Track R. Peon
ISSN: 2070-1721 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.

Status of This Memo

This is an Internet Standards Track document.

This document is a product of the Internet Engineering Task Force (IETF). It represents the consensus of the IETF community. It has received public review and has been approved for publication by the Internet Engineering Steering Group (IESG). Further information on Internet Standards is available in Section 2 of RFC 5741.

Information about the current status of this document, any errata, and how to provide feedback on it may be obtained at http://www.rfc-editor.org/info/rfc7540.

Evolved HTTPS Stack

HTTP/2

TLS

TCP

IP

Evolved HTTPS Stack

HTTP/2

Good enough?

IP

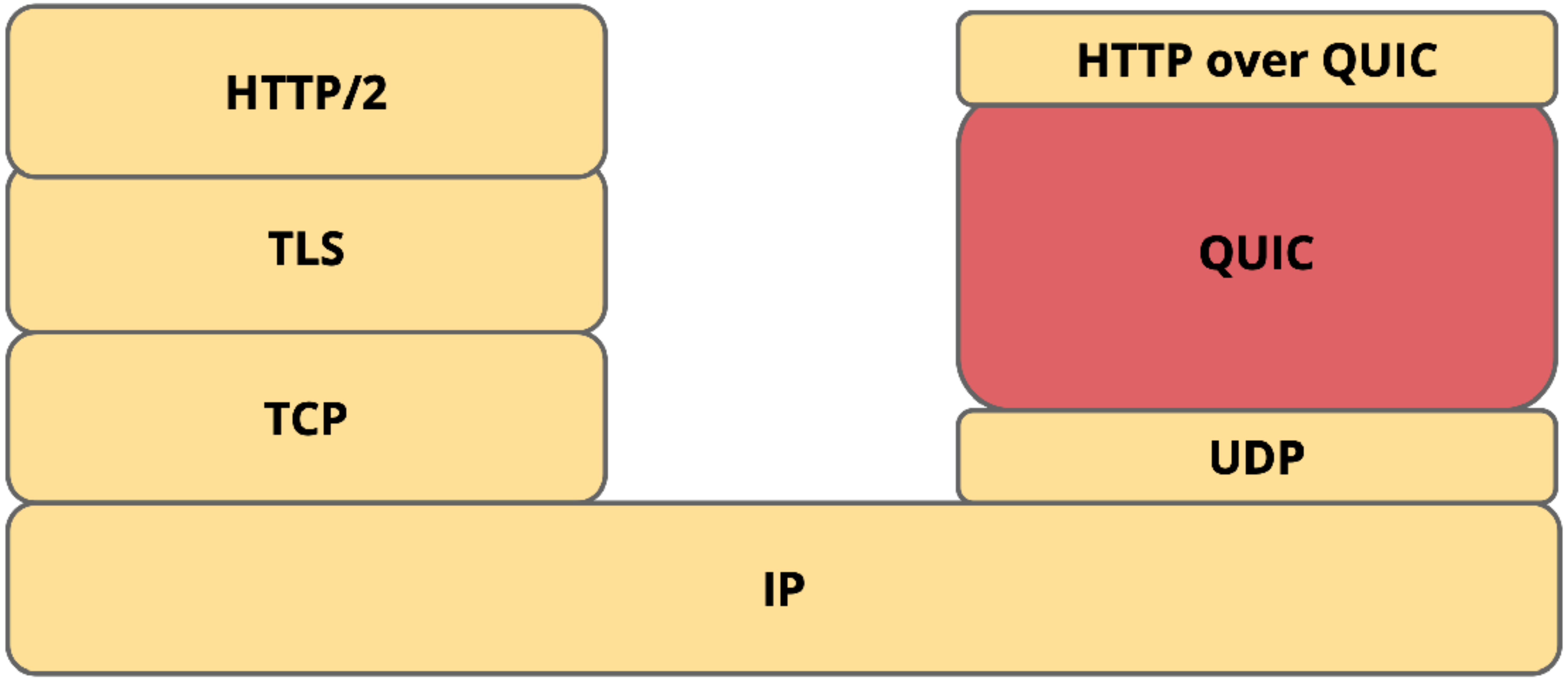
Applications Demands on Web Architecture

- Millisecond-scale latency
  - Search, video streaming, …
- Generality
  - Everything’s going over HTTP, e.g., video, DNS, …
- Scalability
  - Wide API, broad applicability

How can we get rid of inefficiencies in the web stack?

QUIC Overview

- A new transport protocol

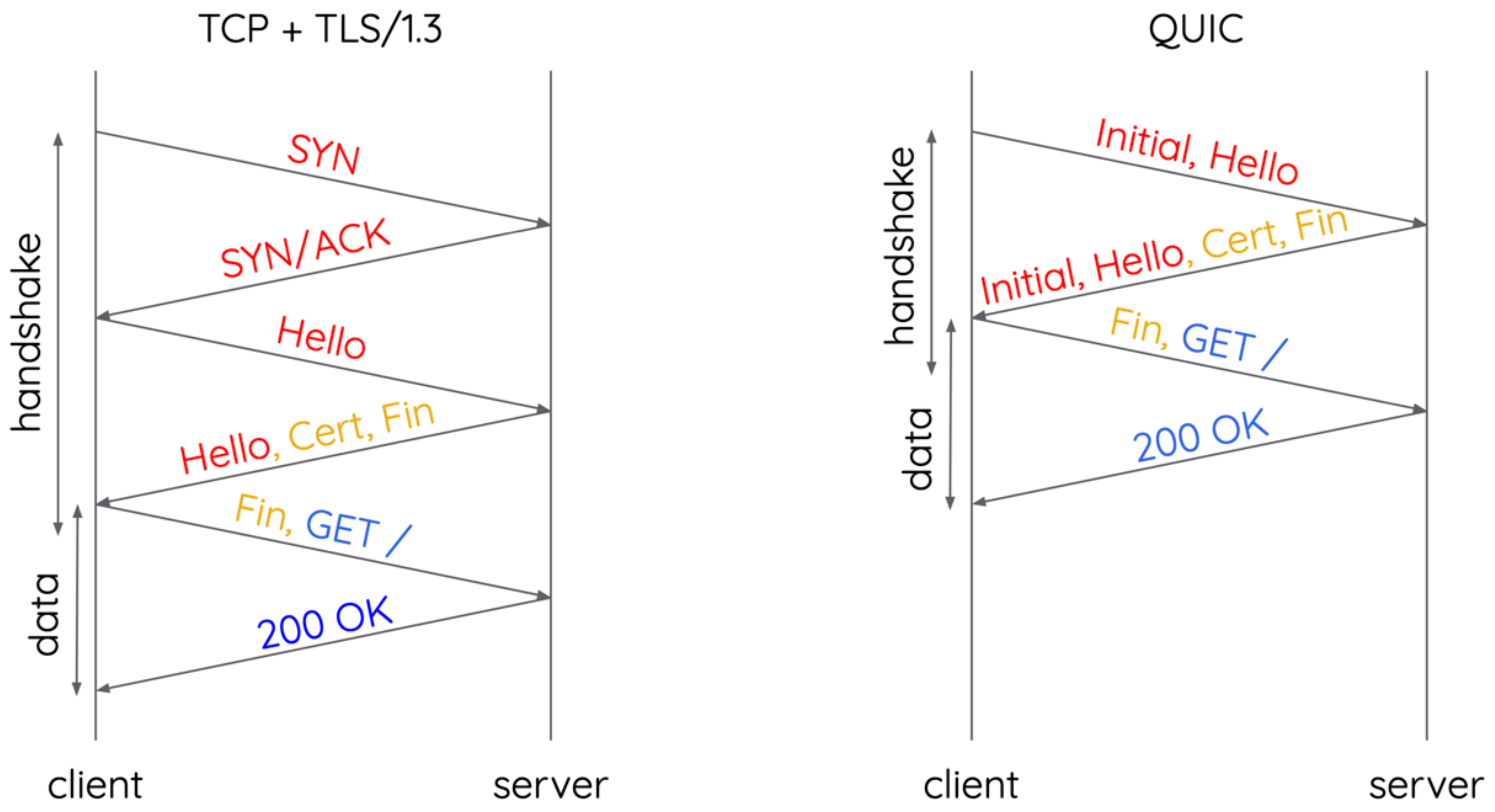


Design Goals

- Performance
  - Reduce page load latency, improve video QoE
- Userspace transport
  - Offer control and agility
  - Enable architecture exploration
- Deployment agility of new features
  - Ossification protection with version, encryption, etc.

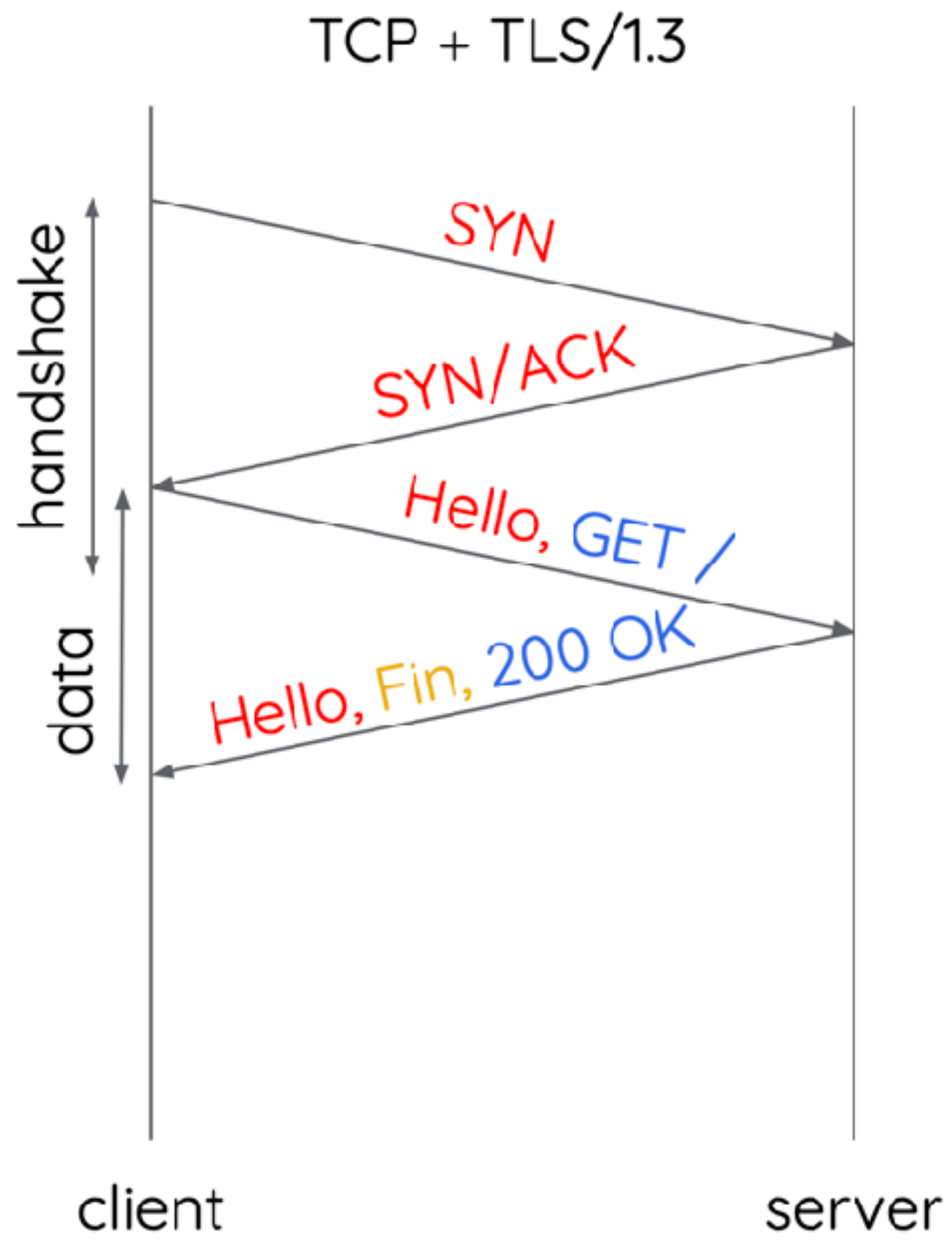
Technique #1: Low-latency Handshake

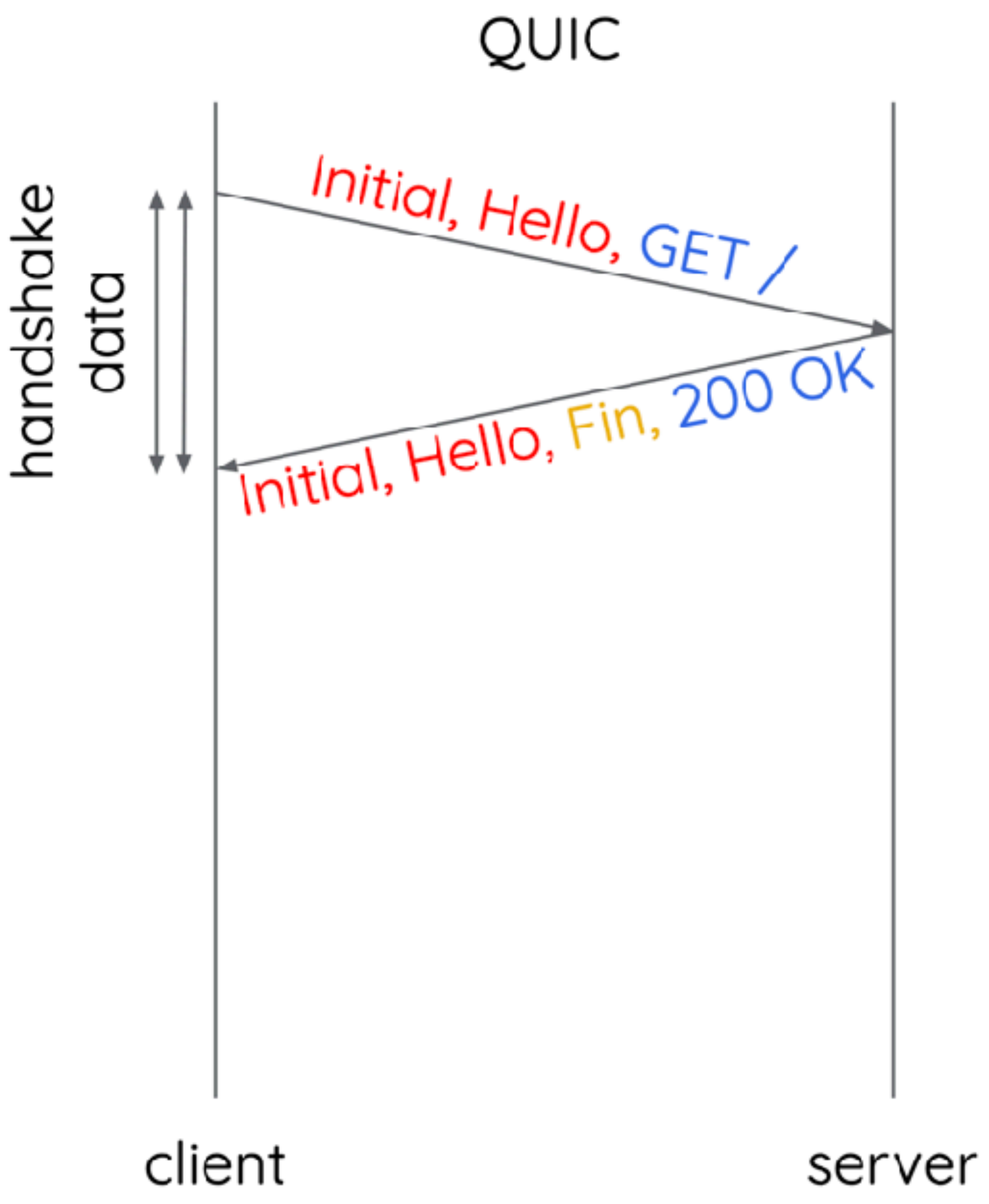
- First connection to server: 2 RTTs



Technique #1: Low-latency Handshake

- First connection to server: 2 RTTs
- Subsequent connection to the same server: 1 RTT





QUIC Handshake Details

- Transport options exchanged in transport parameters
  - Flow control limits, etc
  - Sent as extensions to TLS handshake

- Connection IDs exchanged during handshake
  - Each endpoint chooses CID (and length) to be used towards it

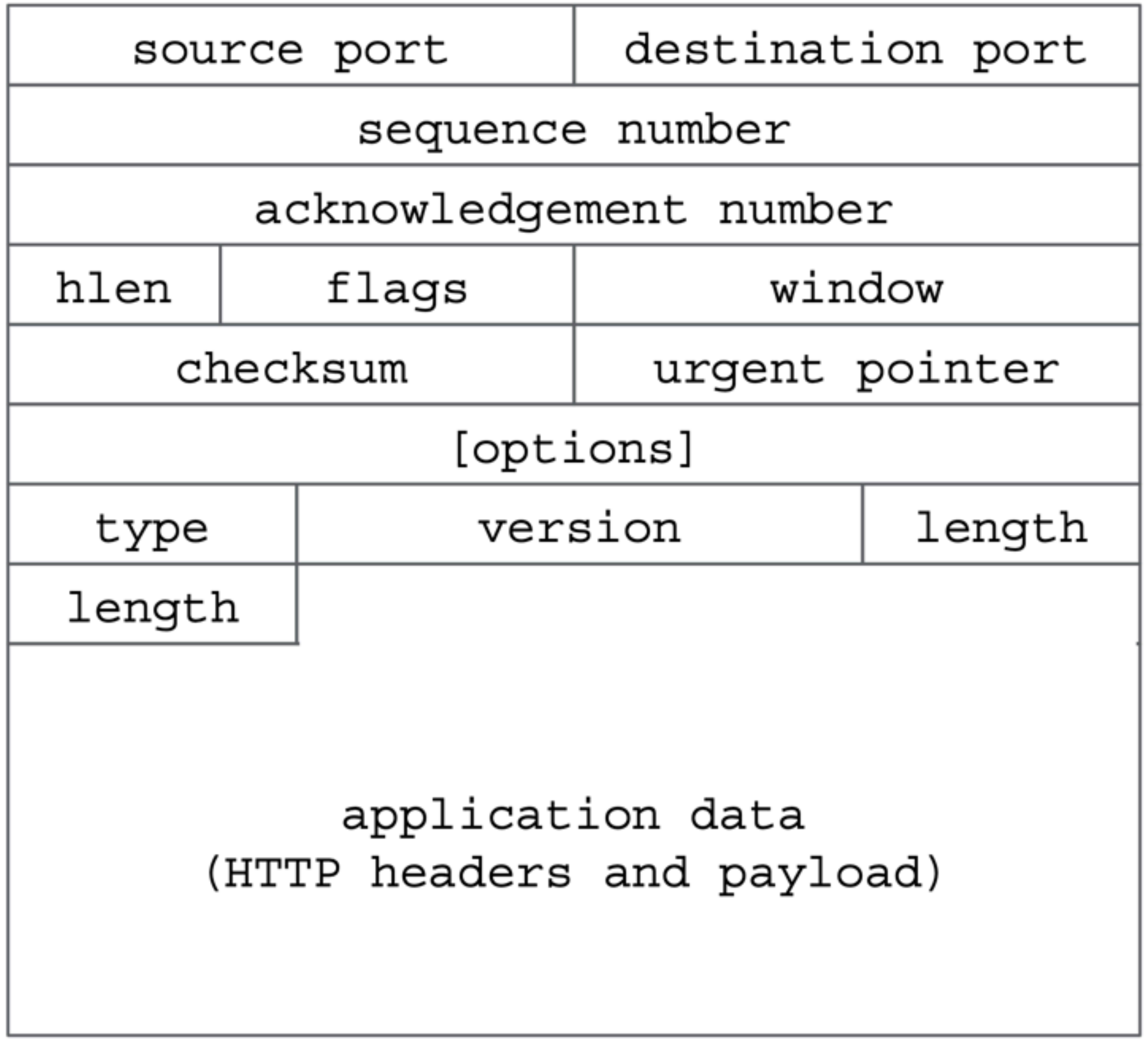
- TLS handshake carried in QUIC packets

- QUIC packets flow on wire
  - Carrying TLS messages, including QUIC options

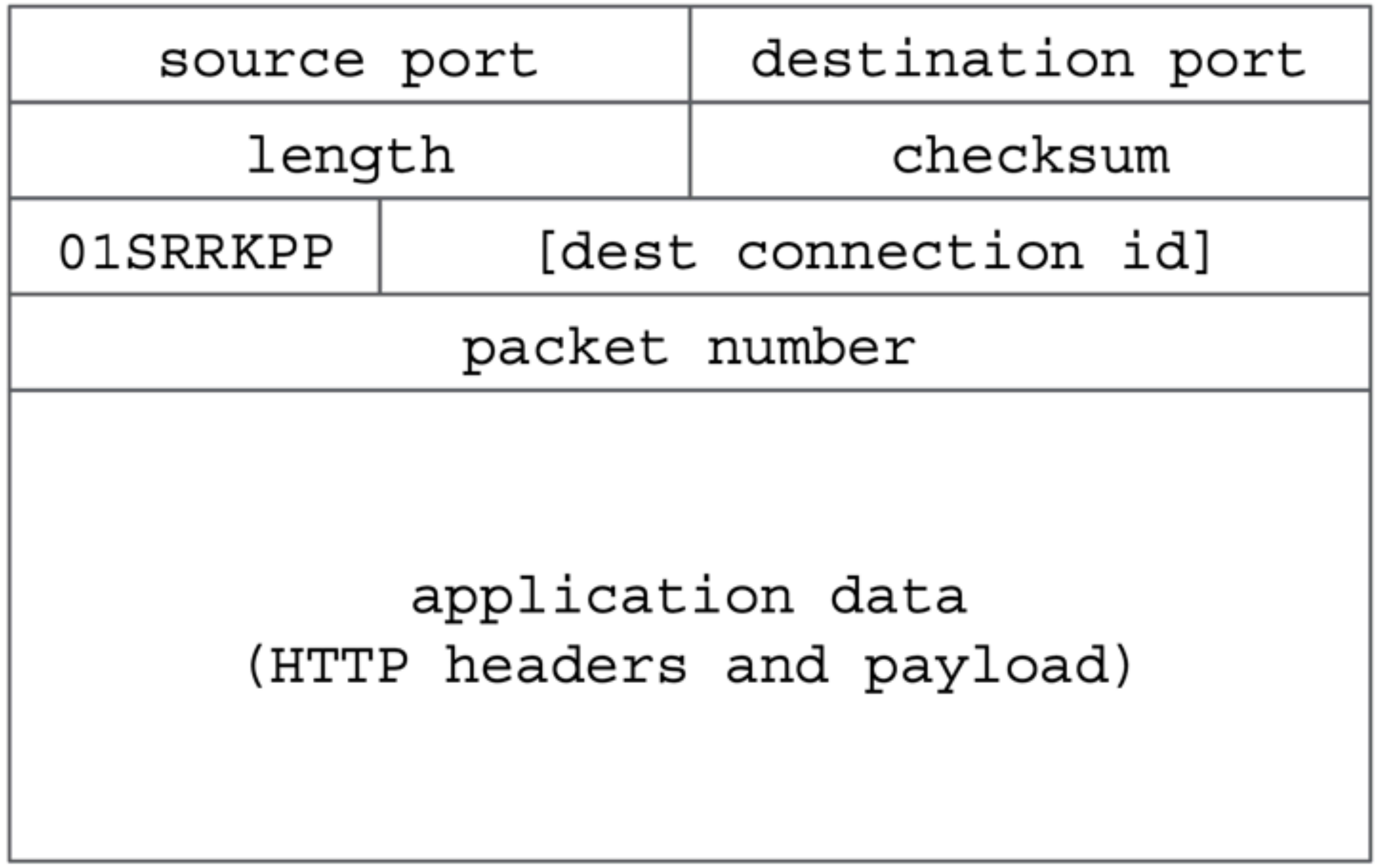
Technique #2: Encrypted Transport

“the ultimate defense of the end to end mode is end to end encryption” — D. Clark, J. Wrocolawski, K. Sollins, and R. Braden

Technique #2: Encrypted Transport

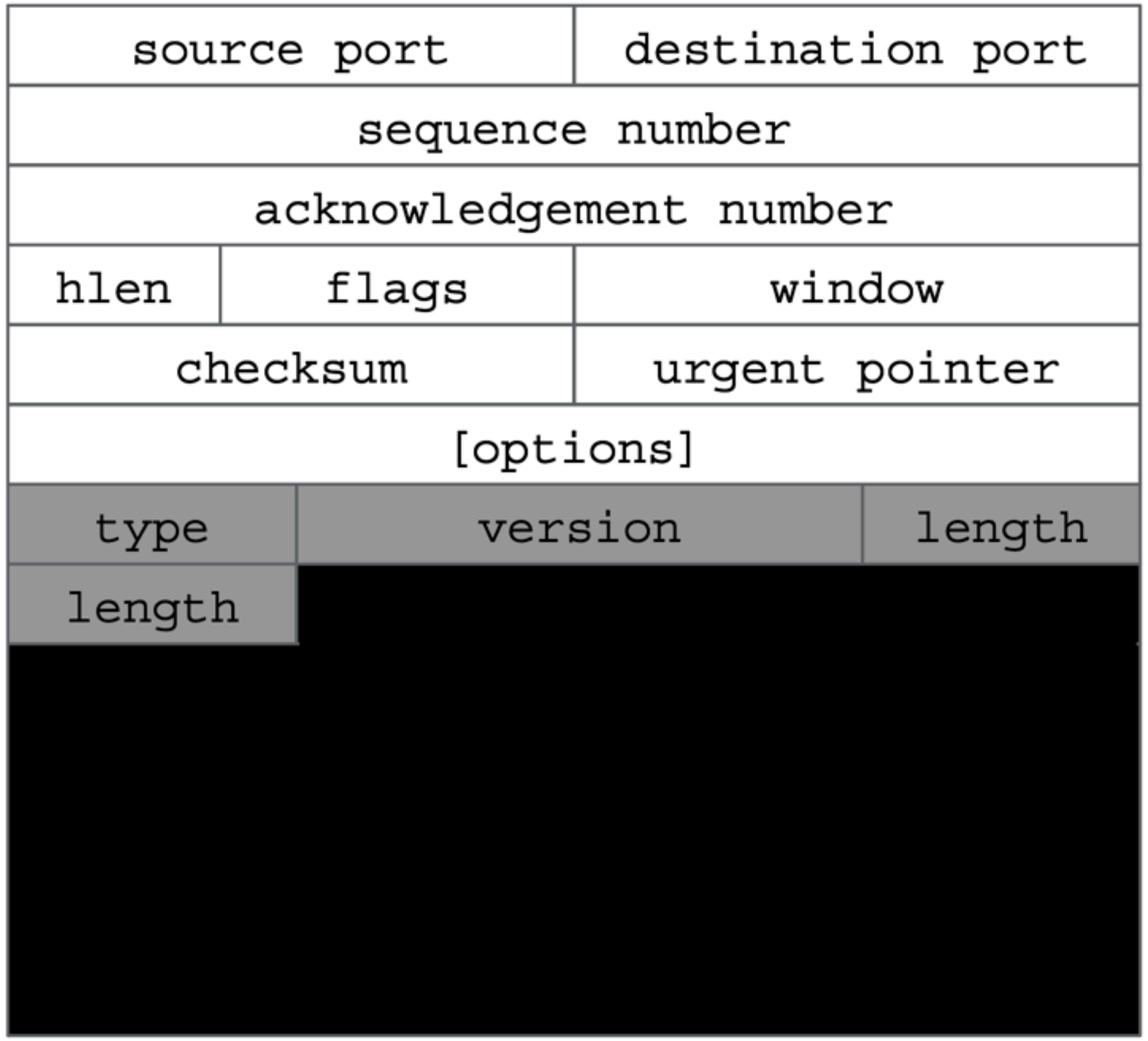


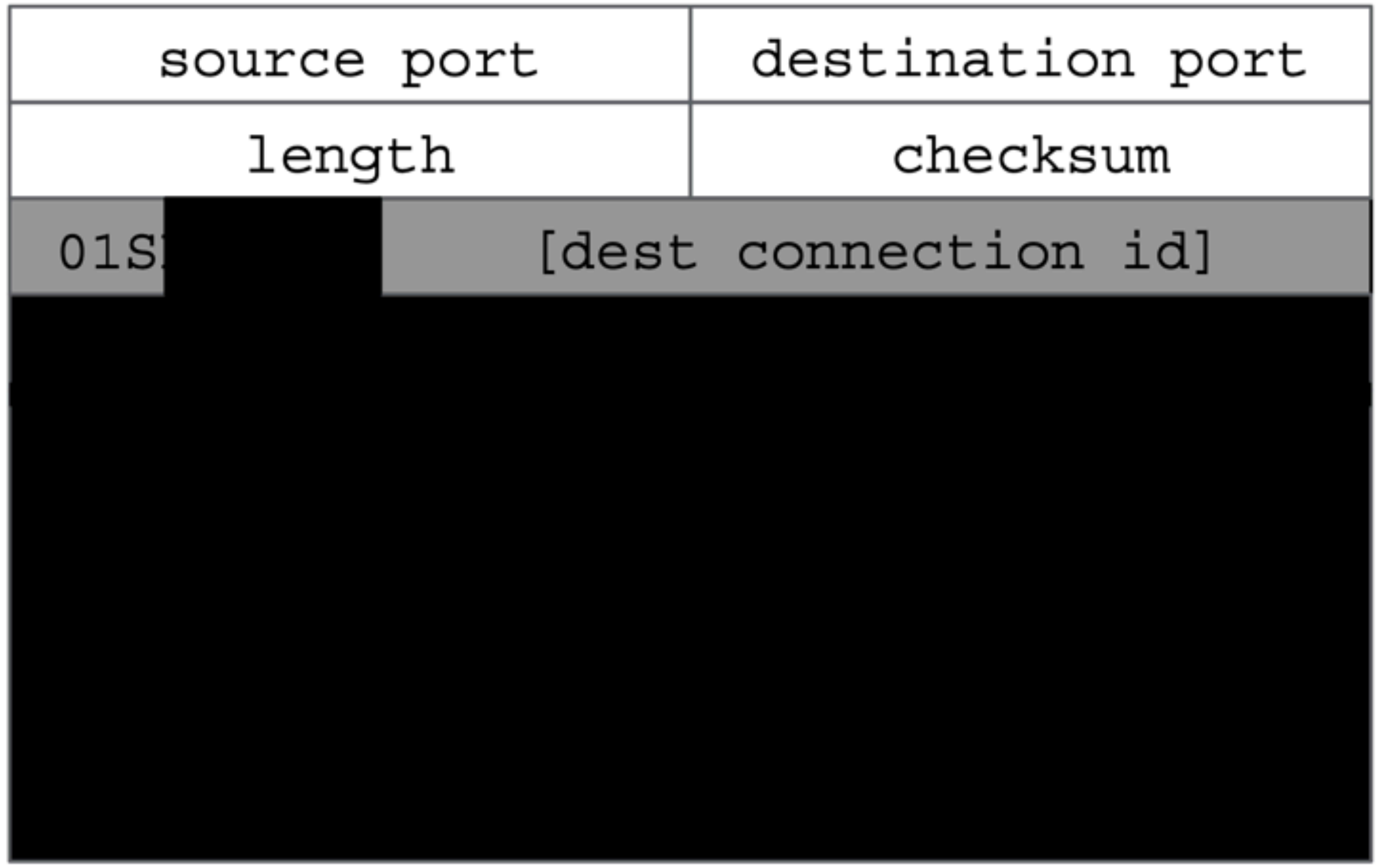
HTTP with TLS/TCP



HTTP with QUIC

Technique #2: Encrypted Transport

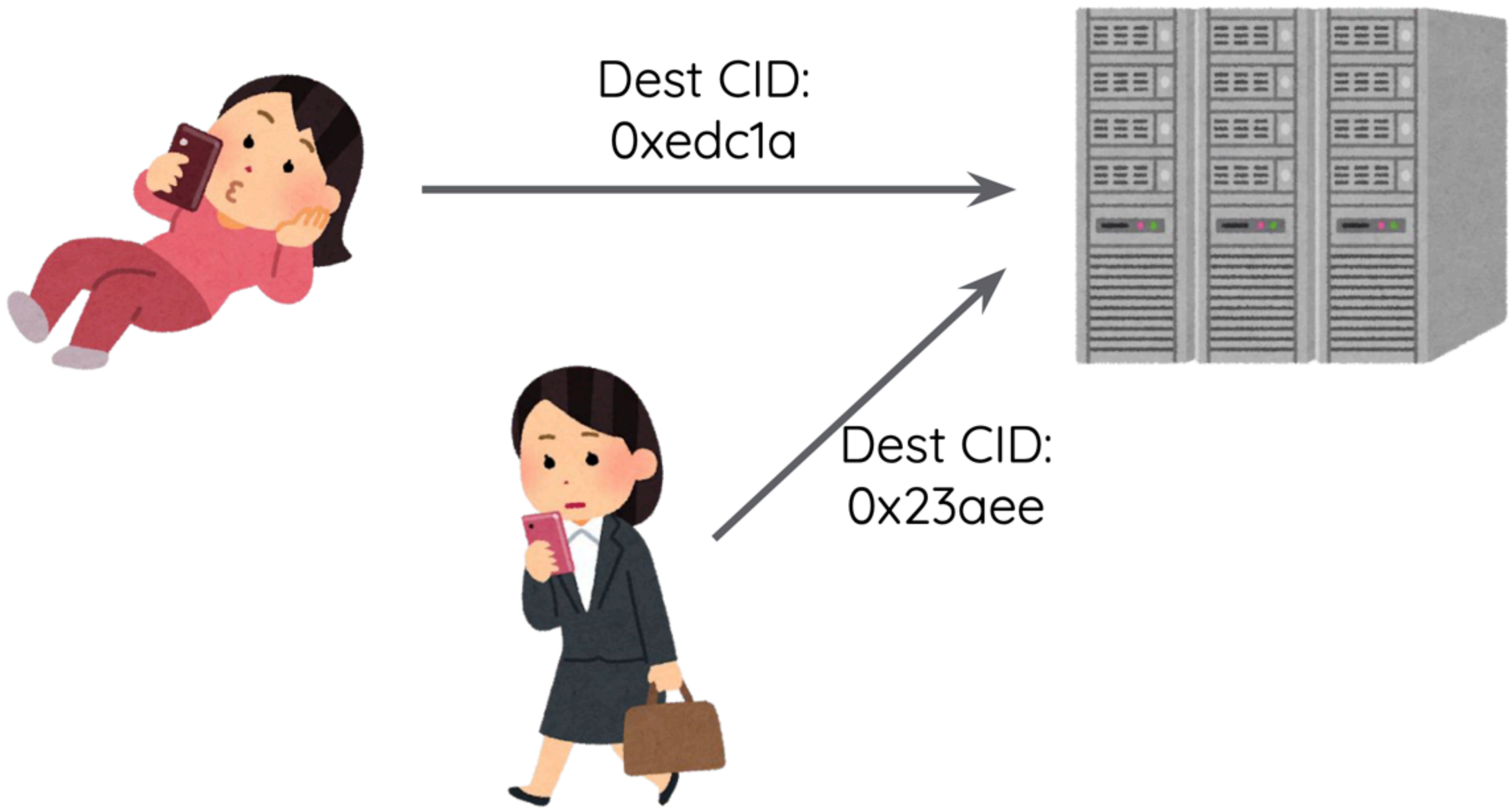




HTTP with TLS/TCP

HTTP with QUIC

Technique #3: Resilient Connections



Packet Number Encryption

- Packet number used as a nonce for packet encryption
  - nonce = used once
  - Receivers need it to decrypt the packet
  - Monotonically increasing, for loss detection and compression

- Visible packet number allows for correlation across networks
  - Any visible bits ossify in the network

- Encrypting packet number would require (another) nonce
  - Encrypted bytes from the packet are random

QUIC Techniques Recap

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

- Encrypted transport
  - Connections protected from tamper and disruption
  - Most of the headers are not even visible to third parties

- Resilient connections
  - Allow connection migration
  - Use 18-byte connection IDs
  - Improved loss recovery, helping connections over “bad” network

QUIC Packet Format

- Use Wireshark for analysis

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 (*)                   ...
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

QUIC Performance

- Application Metric
  - Search latency: user enters search term —> entire page is loaded
  - Video rebuffer rate: rebuffer time / (rebuffer time + video play time)

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

QUIC Performance

- Application Metric
  - Search latency: user enters search term —> entire page is loaded
  - Video rebuffer rate: rebuffer time / (rebuffer time + video play time)

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

QUIC Status

- IETF
  - RFC 9000
- Implementations
  - Apple, Meta, Fastly, Firefox, F5, Google, Microsoft,…
- Server deployments
  - Akamai, Cloudflare, Meta, Fastly, Google,…
- Client-side adoption
  - Chrome, Firefox, Edge, Safari, iOS, MacOS

We are nearly done for this semester.

Networking Research

Building fast, efficient, secure, and reliable networked systems and protocols at different scales

Networking Research

Close to the physical limits of the communication medium

Building fast, efficient, secure, and reliable networked systems and protocols at different scales

Networking Research

Close to the physical limits of the communication medium

Building fast, efficient, secure, and reliable networked systems and protocols at different scales

Energy-efficiency: bps/J
Cost-efficiency: bps/$
Multi-tenancy

Networking Research

Close to the physical limits of the communication medium

Minimize the attack vector
Fast attack detection/prevention

Building fast, efficient, secure, and reliable networked systems and protocols at different scales

Energy-efficiency: bps/J
Cost-efficiency: bps/$
Multi-tenancy

Networking Research

Close to the physical limits of the communication medium

Minimize the attack vector
Fast attack detection/prevention

Building fast, efficient, secure, and reliable networked systems and protocols at different scales

Energy-efficiency: bps/J
Cost-efficiency: bps/$
Multi-tenancy

Maximize MTBF
Close to zero downtime
Prompt troubleshooting

Networking Research

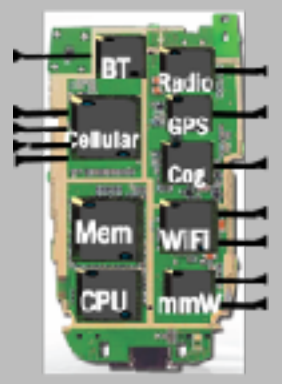
Close to the physical limits of the communication medium

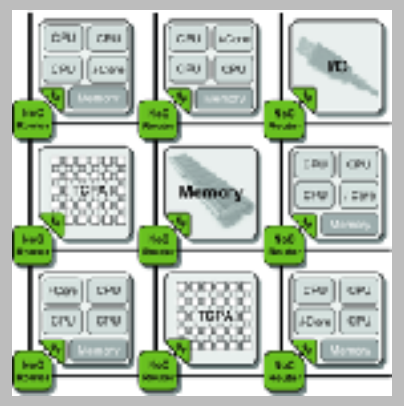
Minimize the attack vector
Fast attack detection/prevention

Building fast, efficient, secure, and reliable networked systems and protocols at different scales

Energy-efficiency: bps/J
Cost-efficiency: bps/$
Multi-tenancy

Maximize MTBF
Close to zero downtime
Prompt troubleshooting





CS 740 studies computer network problems and techniques in the context of data center networks.

CS740 Recap

Data Center Network

CS740 Recap

Data Center Network

Multiple communication paths exist when accessing and traversing data center networks!

Physical Connectivity + Networking Architecture (L1, L2, L3)

CS740 Recap

Data Center Network

The forwarding (destination) address and routing table determine how packets are forwarded!

Addressing and Routing (L4, L5)

Physical Connectivity + Networking Architecture (L1, L2, L3)

CS740 Recap

Data Center Network

Flow scheduling requires knowing the loading status (congestion degree) of path candidates!

Flow Scheduling (L6, L7)

Addressing and Routing (L4, L5)

Physical Connectivity + Networking Architecture (L1, L2, L3)

CS740 Recap

Data Center Network

A performant load-balancer design requires per-packet and per-flow processing at line rate with traffic monitoring.

Load balancing (L8, L9)

Flow Scheduling (L6, L7)

Addressing and Routing (L4, L5)

Physical Connectivity + Networking Architecture (L1, L2, L3)

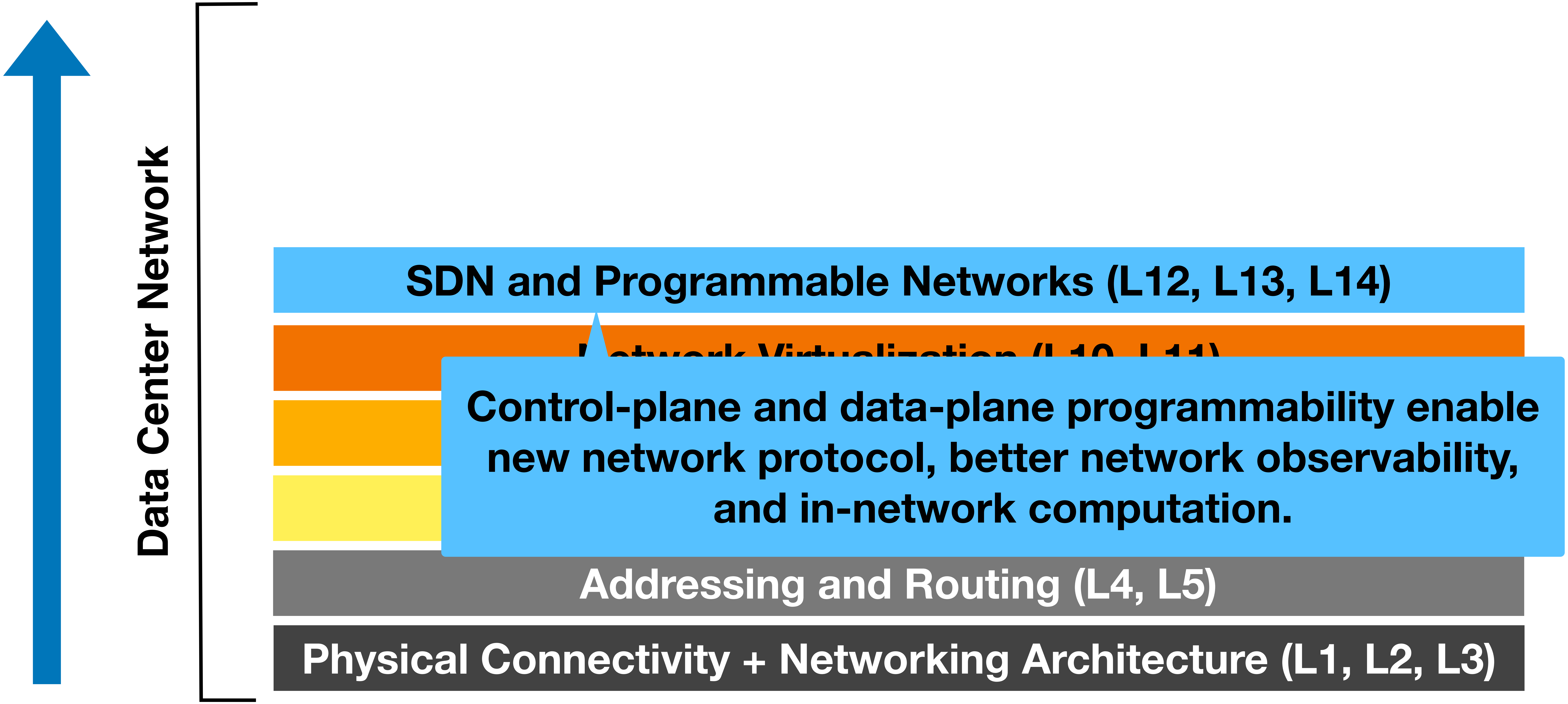
CS740 Recap

Data Center Network

A privileged networking layer stack ensures security isolation and performance isolation.

Network Virtualization (L10, L11)
Load balancing (L8, L9)
Flow Scheduling (L6, L7)
Addressing and Routing (L4, L5)
Physical Connectivity + Networking Architecture (L1, L2, L3)

CS740 Recap



Data Center Network
SDN and Programmable Networks (L12, L13, L14)
Control-plane and data-plane programmability enable new network protocol, better network observability, and in-network computation.
Addressing and Routing (L4, L5)
Physical Connectivity + Networking Architecture (L1, L2, L3)

CS740 Recap

Data Center Network

Transport Layer (L15, L16, L17)

SDN and Programmable Networks (L12, L13, L14)

High throughput, low tail (average) latency, and traffic incast of data center applications motivate in-network and receiver-driven transport design.

Flow Scheduling (L6, L7)

Addressing and Routing (L4, L5)

Physical Connectivity + Networking Architecture (L1, L2, L3)

CS740 Recap

Data Center Network

Endhost Networking Stack (L18, L19)

Transport Layer (L15, L16, L17)

High-bandwidth demands require carefully streamlining data through the compute and memory subsystems at the endhost.

Load balancing (L8, L9)

Flow Scheduling (L6, L7)

Addressing and Routing (L4, L5)

Physical Connectivity + Networking Architecture (L1, L2, L3)

CS740 Recap

Data Center Network

Appliation Layer (L20, L21)

Endhost Networking Stack (L18, L19)

Application end-to-end requirements determine how to design the endhost and in-network mechanisms.

SDN and Programmable Networks (L12, L13, L14)

Network Virtualization (L10, L11)

Load balancing (L8, L9)

Flow Scheduling (L6, L7)

Addressing and Routing (L4, L5)

Physical Connectivity + Networking Architecture (L1, L2, L3)

We are nearly done for this semester.

Servers —> Accelerators (GPUs)

L1

Data Center Networks

- Data center networks connect **servers**.

Servers —> Accelerators (GPUs)

L1

Data Center Networks

- Data center networks connect ~~servers.~~ **GPUs**

Summary

- Today
  - QUIC (SIGCOMM’17)
- Next
  - DCNet for GPUs