QUIC
Context, Design, Deployment
Jana Iyengar
Fastly
The QUIC Transport Protocol: Design and Internet-Scale Deployment

Adam Langley, Alistair Riddoch, Alyssa Wilk, Antonio Vicente, Charles Krasic, Dan Zhang, Fan Yang, Fedor Kouranov, Ian Swett, Janardhan Iyengar, Jeff Bailey, Jeremy Dorfman, Jim Roskind, Joanna Kulik, Patrik Westin, Raman Tenenti, Robbie Shade, Ryan Hamilton, Victor Vasiliev, Wan-Teh Chang, Zhongyi Shi *
Google
quic-sigcomm@google.com

ABSTRACT
We present our experience with QUIC, an encrypted, multiplexed, and low-latency transport protocol designed from the ground up to improve transport performance for HTTPS traffic and to enable rapid deployment and continued evolution of transport mechanisms. QUIC has been globally deployed at Google on thousands of servers and is used to serve traffic to a range of clients including a widely-used web browser (Chrome) and a popular mobile video streaming app (YouTube). We estimate that 7% of Internet traffic is now QUIC. We describe our motivations for developing a new transport, the principles that guided our design, the Internet-scale process that we used to perform iterative experiments on QUIC, performance improvements seen by our various services, and our experience deploying QUIC globally. We also share lessons about transport design and the Internet ecosystem that we learned from our deployment.

TCP (Figure 1). We developed QUIC as a user-space transport with UDP as a substrate. Building QUIC in user-space facilitated its deployment as part of various applications and enabled iterative changes to occur at ammilation undate timescales. The use of UDP
A QUIC history

Protocol for HTTPS transport, deployed at Google starting 2014
Between Google services and Chrome / mobile apps
A QUIC history

Protocol for HTTPS transport, deployed at Google starting 2014
Between Google services and Chrome / mobile apps

Improved application performance
YouTune Video Rebuffers: 15 - 18%
Google Search Latency: 3.6 - 8%
A QUIC history

Protocol for HTTPS transport, deployed at Google starting 2014
Between Google services and Chrome / mobile apps

Improved application performance
YouTube Video Rebuffers: 15 - 18%
Google Search Latency: 3.6 - 8%

35% of Google’s egress traffic (7% of Internet)
A QUIC history

Protocol for HTTPS transport, deployed at Google starting 2014
  Between Google services and Chrome / mobile apps

Improved application performance
  YouTube Video Rebuffers: 15 - 18%
  Google Search Latency: 3.6 - 8%

35% of Google’s egress traffic (7% of Internet)

IETF QUIC working group formed in Oct 2016
  Modularize and standardize QUIC
Google’s QUIC deployment
Google’s QUIC deployment
Google's QUIC deployment
"The life of the dead is placed in the memory of the living"
(Cicero)

QUIC comes in a long line of work on web transport
What are we talking about?

- HTTP
- TLS
- TCP
- IP
The HTTP Story

HTTP/1.0

: independent file transfers
  (open, write, close)
HTTP/1.0: independent file transfers (open, write, close)

The Case for Persistent-Connection HTTP

Jeffrey C. Mogul

May, 1995

Abstract

The success of the World-Wide Web is largely due to the simplicity, hence ease of implementation, of the Hypertext Transfer Protocol (HTTP). HTTP, however, makes inefficient use of network and server resources, and adds unnecessary latencies, by creating a new TCP connection for each request. Modifications to HTTP have been proposed that would transport multiple requests over each TCP connection. These modifications have led to debate over their actual impact on users, on servers, and on the network. This paper reports the results of log-driven simulations of several variants of the proposed modifications, which demonstrate the value of persistent connections.

This Research Report is an expanded version of a paper to appear in the Proceedings of the SIGCOMM ’95 Conference on Communications Architectures and Protocols.

Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the ACM copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Association for Computing Machinery. To copy otherwise, or to republish, requires a fee and/or specific permission.
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.

1. Introduction
2. Role of W3C
3. Current Situation
4. Contacts

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
“Modularity is an important kind of simplicity, and HTTP/1.x isn’t very modular.”
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.
“[...] these connections have poor performance due to TCP slow start as well as the round trips required to open and close each TCP connection”
“[...] these connections have poor performance due to TCP slow start as well as the round trips required to open and close each TCP connection”

“[...] the serialized behavior of HTTP/1.1 pipelining does not adequately support simultaneous rendering of inlined objects”
“[...] these connections have poor performance due to TCP slow start as well as the round trips required to open and close each TCP connection”

“[...] the serialized behavior of HTTP/1.1 pipelining does not adequately support simultaneous rendering of inlined objects”

“[...] nor does it provide suitable fairness between protocol flows”
“[...] these connections have poor performance due to TCP slow start as well as the round trips required to open and close each TCP connection”

“[...] the serialized behavior of HTTP/1.1 pipelining 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”
“[...] these connections have poor performance due to TCP slow start as well as the round trips required to open and close each TCP connection”

“[...] the serialized behavior of HTTP/1.1 pipelining 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”

“[...] TCP implementations do not share congestion information across multiple simultaneous connections between two peers”
“[...] 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

T/TCP -- TCP Extensions for Transactions
Functional Specification

July 1994

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 ............................................................. 2
The Transport Story

T/TCP

<table>
<thead>
<tr>
<th>Function</th>
<th>Kernel</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>initiating</td>
<td></td>
<td>responding</td>
</tr>
<tr>
<td>socket</td>
<td>sleep</td>
<td>socket</td>
</tr>
<tr>
<td>sendto</td>
<td>SYN,FIN data(request)</td>
<td>bind</td>
</tr>
<tr>
<td>read</td>
<td></td>
<td>listen</td>
</tr>
<tr>
<td>return(data)</td>
<td>wakeup</td>
<td>accept</td>
</tr>
<tr>
<td>read(EOF)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- | 
- | 
- |
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.
“[...] 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
TCP Session
Congestion Manager

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 adequate 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
TCP Session
Congestion Manager

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

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

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

T/TCP
TCP Session
Congestion Manager

The Congestion Manager

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 (2001). All Rights Reserved.

Abstract

This document describes the Congestion Manager (CM), an end-system module that:

(i) Enables an ensemble of multiple concurrent streams from a sender destined to the same receiver and sharing the same congestion properties to perform proper congestion avoidance and control, and

(ii) Allows applications to easily adapt to network congestion.
The Transport Story

T/TCP
TCP Session
Congestion Manager

Network Working Group
Request for Comments: 3124
Category: Standards Track

The Congestion Manager

Status of this Memo

This document specifies an Internet standards track protocol for the Internet community. It requests implementation and deployment of Internet technologies, as described in the Internet Standards track normative documents. Please refer to the current version of the "Internet Protocol Standards" document for the latest specification of the protocol. Implementation and deployment and status of this protocol is subject to change and any implementation of it must be done in a manner consistent with any related specifications. It is not a normative document and can be considered as a work in progress. It is informative.

Copyright Notice

Copyright (C) The Internet Society (IETF), June 2001. All Rights Reserved.

Abstract

This document describes the congestion manager (CM), an end-system module that:

(i) Enables an ensemble of multiple concurrent streams from a sender destined to the same receiver and sharing the same congestion properties to perform proper congestion avoidance and control, and

(ii) Allows applications to easily adapt to network congestion.
IETF Detour!
The Transport Story

T/TCP
TCP Session
Congestion Manager

The Congestion Manager

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 RFC 2026 and the "Internet Style Guide" for more information.

Copyright Notice

Copyright (C) The Internet Society

Abstract

This document describes the congestion manager (CM), an end-system module that:

(i) Enables an ensemble of multiple concurrent streams from a sender destined to the same receiver and sharing the same congestion properties to perform proper congestion avoidance and control, and

(ii) Allows applications to easily adapt to network congestion.
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:
The Transport Story

T/TCP
TCP Session
Congestion Manager
SCTP

...
The Transport Story

T/TCP
TCP Session
Congestion Manager
SCTP
...
...
SST (UDP-based)
Minion (TCP and TLS based)
The Transport Story

T/TCP
TCP Session
Congestion Manager
SCTP
...
...
SST (UDP-based)
Minion (TCP and TLS based)
TCP Fast Open
The Transport Story

T/TCP
TCP Session
Congestion Manager
SCTP
...
...
SST (UDP-based)
Minion (TCP and TLS based)
TCP Fast Open
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

Home routers (NATs)
Firewalls
Application load balancers (HTTP)
Protocol accelerators (PEPs)
Middleboxes

Stages of grief
The HTTP Story (contd.)

HTTP/1.0
HTTP/1.1
HTTP/1.1
HTTP/1.1
HTTP/1.1
The HTTP Story

HTTP/1.0
HTTP/1.1
HTTPng (?)
...
...
...
...
...
The HTTP Story

HTTP/1.0
HTTP/1.1
HTTPng (?)

SPDY

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 another request. Consequently, adding thousands of parallel connections can significantly increase latency (e.g., 5000 requests).

SPDY
HTTP/1.0
HTTP/1.1
HTTPng (?)

SPDY

The HTTP Story

SPDY: An experimental protocol for a faster web

Application
Session
Presentation
Transport

HTTP
SPDY
SSL
TCP

Except as otherwise noted, the content of this page is licensed under a Creative Commons Attribution 2.5 license, and examples are licensed under the BSD License.

While we and other organizations have

Exchanged 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
http://www.chromium.org/spdy/spdy-whitepaper

The Chromium Projects

HTTPng (?)

The Chromium Projects
The HTTP Story

HTTP/1.0
HTTP/1.1
HTTPng (?)

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

HTTP/2
streams
multiplexing
flow control
priorities

---

The HTTP Story

**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.
What are we talking about?

- HTTP/2
- TLS
- TCP
- IP
Google’s QUIC Experiment

HTTP/2
TLS
TCP
IP

HTTP over gQUIC
gQUIC
QUIC Crypto
UDP
The QUIC Standard

HTTP/2

TLS

TCP

UDP

HTTP over QUIC

QUIC

TLS 1.3

IP
The QUIC Standard

- HTTP/2
- TLS
- TCP
- UDP
- draft-ietf-quic-http
- draft-ietf-quic-transport
- draft-ietf-quic-recovery
- draft-ietf-quic-tls
- TLS 1.3
- IP
Deployability and evolvability in userspace, atop UDP encrypted and authenticated headers
Sometime, a while ago

First byte of gQUIC packet was *flags*
Sometime, a while ago

First byte of gQUIC packet was flags : unencrypted, and had been 0x07 for a while
Sometime, a while ago

First byte of gQUIC packet was *flags* : unencrypted, and had been 0x07 for a while

We flipped a bit.
Sometime, a while ago

First byte of gQUIC packet was flags: unencrypted, and had been 0x07 for a while

We flipped a bit.

“users cannot reach any Google property over Chrome!”
What had happened

Firewall
  : allowed first packet in both directions
  : blackholed all subsequent packets
What had happened

Firewall
  : allowed first packet in both directions
  : blackholed all subsequent packets

“in wireshark, noticed that first byte was always the same”
What had happened

if udp_payload[0] == 7: QUIC
“the ultimate defense of the end to end mode is end to end encryption”

QUIC Design Goals (1 of 2)

Deployability and evolvability
  in userspace, atop UDP
  encrypted and authenticated headers

Low-latency secure connection establishment
  mostly 0-RTT, sometimes 1-RTT
  (similar to TCP Fast Open + TLS 1.3)
QUIC Design Goals (1 of 2)

Deployability and evolvability
in userspace, atop UDP
encrypted and authenticated headers

Low-latency secure connection establishment
mostly 0-RTT, sometimes 1-RTT
(similar to TCP Fast Open + TLS 1.3)

Streams and multiplexing
lightweight abstraction within a connection
avoids head-of-line blocking in TCP
Better loss recovery and flexible congestion control
unique packet number, receiver timestamp
QUIC Design Goals (2 of 2)

Better loss recovery and flexible congestion control
unique packet number, receiver timestamp

Connection migration
18-byte Connection ID
also, resilience to NAT rebinding
## QUIC Performance

<table>
<thead>
<tr>
<th>Country</th>
<th>Mean Min RTT (ms)</th>
<th>Mean TCP Rtx %</th>
<th>% Reduction in Search Latency</th>
</tr>
</thead>
<tbody>
<tr>
<td>South Korea</td>
<td>38</td>
<td>1</td>
<td>1.3</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1.1</td>
</tr>
</tbody>
</table>
# QUIC Performance

<table>
<thead>
<tr>
<th>Country</th>
<th>Mean Min RTT (ms)</th>
<th>Mean TCP Rtx %</th>
<th>% Reduction in Search Latency</th>
</tr>
</thead>
<tbody>
<tr>
<td>South Korea</td>
<td>38</td>
<td>1</td>
<td>1.3</td>
</tr>
<tr>
<td>USA</td>
<td>50</td>
<td>2</td>
<td>3.4</td>
</tr>
</tbody>
</table>
## QUIC Performance

<table>
<thead>
<tr>
<th>Country</th>
<th>Mean Min RTT (ms)</th>
<th>Mean TCP Rtx %</th>
<th>% Reduction in Search Latency</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>Desktop</td>
</tr>
<tr>
<td>South Korea</td>
<td>38</td>
<td>1</td>
<td>1.3</td>
</tr>
<tr>
<td>USA</td>
<td>50</td>
<td>2</td>
<td>3.4</td>
</tr>
<tr>
<td>India</td>
<td>188</td>
<td>8</td>
<td>13.2</td>
</tr>
</tbody>
</table>
## QUIC Performance

<table>
<thead>
<tr>
<th>Country</th>
<th>Mean Min RTT (ms)</th>
<th>Mean TCP Rtx %</th>
<th>% Reduction in Rebuffer Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>South Korea</td>
<td>38</td>
<td>1</td>
<td>0.0</td>
</tr>
<tr>
<td>USA</td>
<td>50</td>
<td>2</td>
<td>4.1</td>
</tr>
<tr>
<td>India</td>
<td>188</td>
<td>8</td>
<td>22.1</td>
</tr>
</tbody>
</table>
Current Status

Work at IETF for past 2 years

Strong focus on security and privacy
  Network operator woes

Strong focus on avoiding ossification
  Encryption
  GREASEing

Several implementation efforts
  Google, Facebook, Fastly, Firefox, Microsoft, Apple, F5