Increasing the speed of Internet: HTTP/3 & QUIC

23.01.2020

PyStok

Container Solutions

piotr.szwed@container-solutions.com

Agenda

- Internet Protocols
  - Past: HTTP **1+2** /TCP
  - Future: HTTP **3** /UDP
- Demo
  - Google Chrome
  - Golang + QUIC
  - Deploying QUIC to Kubernetes (nginx? openresty?)
- Summary

Internet Protocols

Past: TCP

TCP (RFC: 761)

Transmission Control Protocol
Introduction

TCP is based on concepts first described by Cerf and Kahn in [1]. The TCP fits into a layered protocol architecture just above a basic Internet Protocol [2] which provides a way for the TCP to send and receive variable-length segments of information enclosed in internet datagram "envelopes". The internet datagram provides a means for addressing source and destination TCPs in different networks. The internet protocol also deals with any fragmentation or reassembly of the TCP segments required to achieve transport and delivery through multiple networks and interconnecting gateways. The internet protocol also carries information on the precedence, security classification and compartmentation of the TCP segments, so this information can be communicated end-to-end across multiple networks.

Protocol Layering

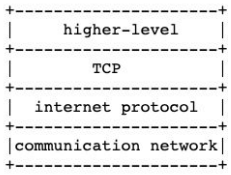


Figure 1

Much of this document is written in the context of TCP implementations which are co-resident with higher level protocols in the host computer. As a practical matter, many computer systems will be connected to networks via front-end computers which house the TCP and internet protocol layers, as well as network specific software. The TCP specification describes an interface to the higher level protocols which appears to be implementable even for the front-end case, as long as a suitable host-to-front end protocol is implemented.

UDP (RFC: 768)

RFC 768

J. Postel
ISI
28 August 1980

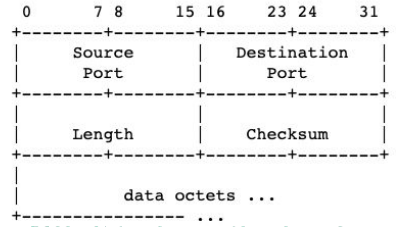
User Datagram Protocol

Introduction

This User Datagram Protocol (UDP) is defined to make available a datagram mode of packet-switched computer communication in the environment of an interconnected set of computer networks. This protocol assumes that the Internet Protocol (IP) [1] is used as the underlying protocol.

This protocol provides a procedure for application programs to send messages to other programs with a minimum of protocol mechanism. The protocol is transaction oriented, and delivery and duplicate protection are not guaranteed. Applications requiring ordered reliable delivery of streams of data should use the Transmission Control Protocol (TCP) [2].

Format



HTTP 0.9 (www.w3.org/Protocols/HTTP)

The Original HTTP as defined in 1991

This document defines the Hypertext Transfer protocol (HTTP) as originally implemented by the World Wide Web initaitive software in the prototype released. This is a subset of the full HTTP protocol, and is known as HTTP 0.9.

No client profile information is transferred with the query. Future HTTP protocols will be back-compatible with this protocol.

This restricted protocol is very simple and may always be used when you do not need the capabilities of the full protocol which is backwards compatible.

The definition of this protocol is in the public domain (see policy).

The protocol uses the normal internet-style telnet protocol style on a TCP-IP link. The following describes how a client acquires a (hypertext) document from an HTTP server, given an HTTP document address .

Connection

The client makes a TCP-IP connection to the host using the domain name or IP number , and the port number given in the address.

If the port number is not specified, 80 is always assumed for HTTP.

The server accepts the connection.

Note: HTTP currently runs over TCP, but could run over any connection-oriented service. The interpretation of the protocol below in the case of a sequenced packet service (such as DECnet(TM) or ISO TP4) is that that the request should be one TPDU, but the response may be many.

Request

The client sends a document request consisting of a line of ASCII characters terminated by a CR LF (carriage return, line feed) pair. A well-behaved server will not require the carriage return character.

This request consists of the word "GET", a space, the document address , omitting the "http:, host and port parts when they are the coordinates just used to make the connection. (If a gateway is being used, then a full document address may be given specifying a different naming scheme).

The document address will consist of a single word (ie no spaces). If any further words are found on the request line, they MUST either be ignored, or else treated according to the full HTTP spec .

The search functionality of the protocol lies in the ability of the addressing syntax to describe a search on a named index .

A search should only be requested by a client when the index document itself has been descibed as an index using the ISINDEX tag .

Response

The response to a simple GET request is a message in hypertext mark-up language (HTML). This is a byte stream of ASCII characters.

Lines shall be delimited by an optional carriage return followed by a mandatory line feed chararcter. The client should not assume that the carriage return will be present. Lines may be of any length. Well-behaved servers should retrict line length to 80 characters excluding the CR LF pair.

The format of the message is HTML - that is, a trimmed SGML document. Note that this format allows for menus and hit lists to be returned as hypertext. It also allows for plain ASCII text to be returned following the PLAINTEXT tag .

The message is terminated by the closing of the connection by the server.

Well-behaved clients will read the entire document as fast as possible. The client shall not wait for user action (output paging for example) before reading the whole of the document. The server may impose a timeout of the order of 15 seconds on inactivity.

Error responses are supplied in human readable text in HTML syntax. There is no way to distinguish an error response from a satisfactory response except for the content of the text.

Disconnection

The TCP-IP connection is broken by the server when the whole document has been transferred.

The client may abort the transfer by breaking the connection before this, in which case the server shall not record any error condition.

Requests are idempotent . The server need not store any information about the request after disconnection.

Tim BL

HTTP/1.0 (RFC: 1945)

Network Working Group
Request for Comments: 1945
Category: Informational

T. Berners-Lee
MIT/LCS
R. Fielding
UC Irvine
H. Frystyk
MIT/LCS
May 1996

Hypertext Transfer Protocol -- HTTP/1.0

Status of This Memo

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

IESG Note:

The IESG has concerns about this protocol, and expects this document to be replaced relatively soon by a standards track document.

Abstract

The Hypertext Transfer Protocol (HTTP) is an application-level protocol with the lightness and speed necessary for distributed, collaborative, hypermedia information systems. It is a generic, stateless, object-oriented protocol which can be used for many tasks, such as name servers and distributed object management systems, through extension of its request methods (commands). A feature of HTTP is the typing of data representation, allowing systems to be built independently of the data being transferred.

HTTP has been in use by the World-Wide Web global information initiative since 1990. This specification reflects common usage of the protocol referred to as "HTTP/1.0".

HTTP/1.1 (RFC: 2616)

June 1999

Hypertext Transfer Protocol -- HTTP/1.1

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

Abstract

The Hypertext Transfer Protocol (HTTP) is an application-level protocol for distributed, collaborative, hypermedia information systems. It is a generic, stateless, protocol which can be used for many tasks beyond its use for hypertext, such as name servers and distributed object management systems, through extension of its request methods, error codes and headers [47]. A feature of HTTP is the typing and negotiation of data representation, allowing systems to be built independently of the data being transferred.

HTTP has been in use by the World-Wide Web global information initiative since 1990. This specification defines the protocol referred to as "HTTP/1.1", and is an update to RFC 2068 [33].

HTTP/2.0 (RFC: 7540)

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.

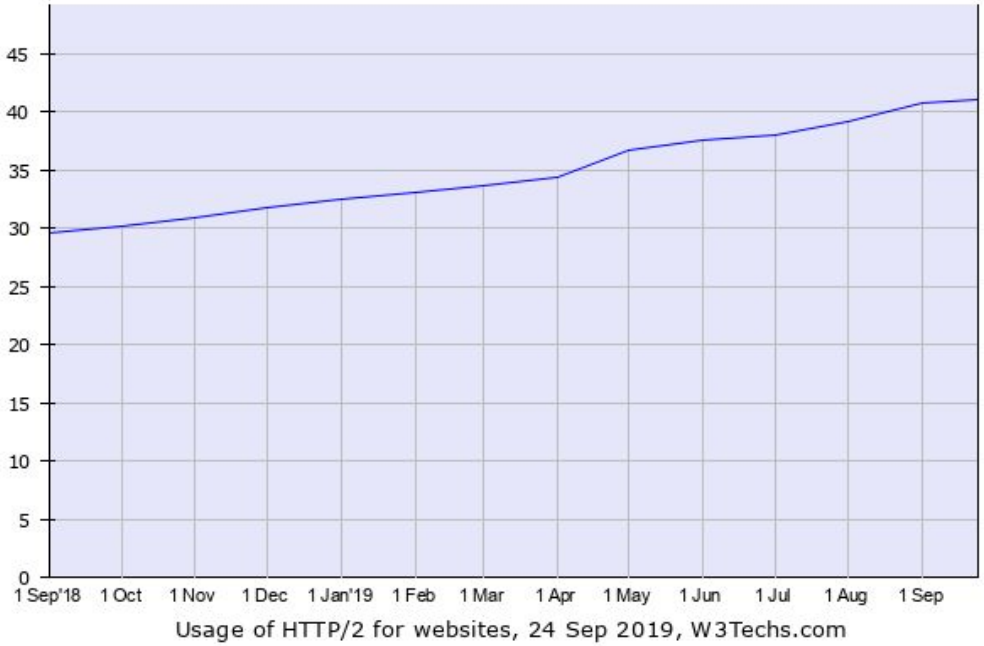
Status of This Memo

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.

HTTP/2.0 is used by 41.0% of all the websites.



Usage of HTTP/2 for websites, 24 Sep 2019, W3Techs.com

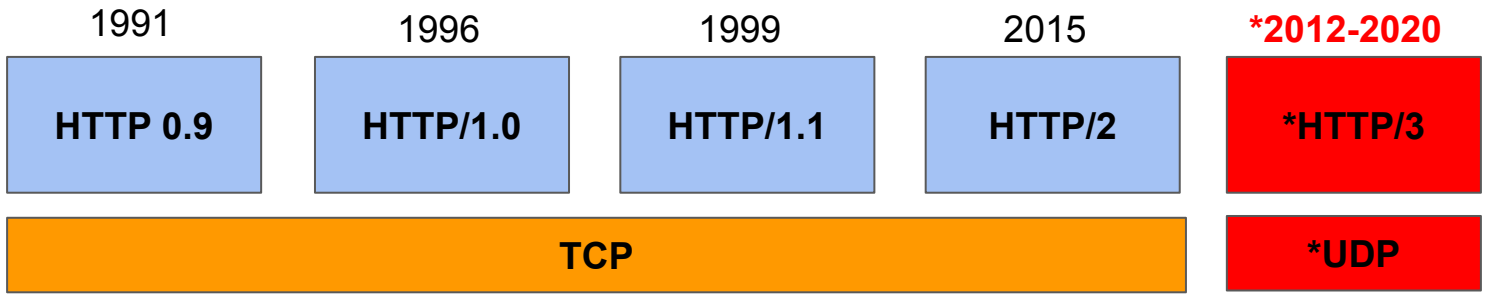
Popular sites using HTTP/2

- Google.com
- Youtube.com
- Tmall.com
- Qq.com
- Sohu.com
- Wikipedia.org
- Taobao.com
- Yahoo.com
- Amazon.com
- Sina.com.cn

Internet Protocols

Future: UDP

Timeline



*__QUIC Internet-Draft__: *A UDP-Based Multiplexed and Secure Transport*

draft-ietf-quic-transport-23:

- released: September 12, 2019
- expires: March 15, 2020

QUIC birth at Google

QUIC

Quick UDP Internet Connections

MULTIPLEXED STREAM TRANSPORT OVER UDP

Jim Roskind <jar@chromium.org>

To paraphrase a famous quote: *I apologize in advance for the length of this document, and if I had more time, I would surely have written less.*

First Draft 2012-04
Revised: 2013-06-24
Minor Edit: 2013-12-2

[Update, 2015-06-5] The QUIC protocol has been deployed at scale across Google for several months now, and is going strong! I'm extremely proud of this work that I initiated at the start of 2012, and the team that I led for over two and a half years. In 2012, I had an idea that we could drastically improve the way that traffic moves on the Internet, and we're now seeing that vision realized. While QUIC will always be near and dear to my heart, I've transitioned to other work inside Google, and won't be updating this document any further. Instead, I'm going to let the document stand as is, to provide historical and background context. Though some of the details in QUIC may have changed, the central design philosophy described herein has not. Readers interested in the QUIC protocol's current status and interoperability should refer to updated specifications found here. - Jim

source: https://tinyurl.com/otssex3

Chromium Blog

News and developments from the open source browser project

Experimenting with QUIC

Thursday, June 27, 2013

At Google, we're always working to make the web faster. The SPDY protocol, which is now the foundation of the upcoming HTTP 2.0 protocol, is a significant step forward. However, despite increasing bandwidth, round trip time (RTT)--which is ultimately bounded by the speed of light--is not decreasing, and will remain high on mobile networks for the foreseeable future. To continue improving network performance we need to decrease the number of round trips, something that is difficult with protocols that currently rely on the Transmission Control Protocol (TCP).

source: https://tinyurl.com/yanmt2tg

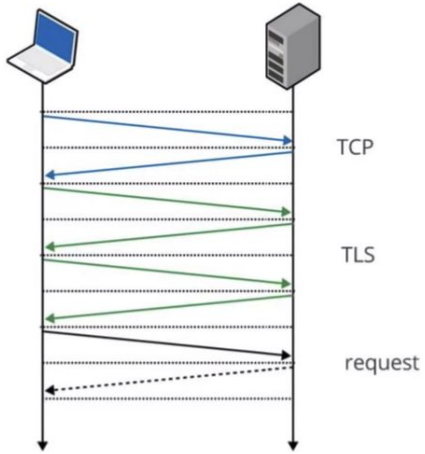
Why milliseconds count?

Delay	Perception
0 – 100 ms	Instant
100 – 300 ms	Small perceptible delay
300 – 1000 ms	Machine is working
1+ s	Mental context switch
10+ s	Task abandoned

Performance goal:
load time **< 1s**

Ilya Grigorik, "High performance browser networking" (2013)

TCP is relatively slow

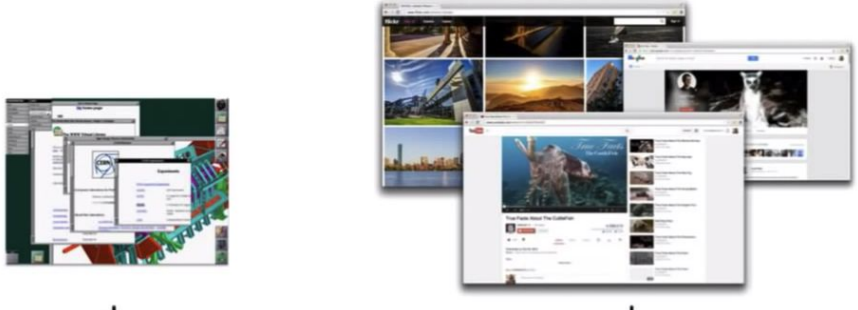


TCP + TLS: 3 round trips

image from High Performance Browser Networking (Grigorik 2013)

source: http://bit.ly/1gqR7WN

HTTP traffic grows fast



1990

single, static page
one resource
one domain

2014

1,200 KB
80 resources
30 domains

HTTP/3.0 (QUIC)

Google

QUIC is an IETF Working Group that is chartered to deliver the next transport protocol for the Internet.

September 12, 2019

QUIC: A UDP-Based Multiplexed and Secure Transport
draft-ietf-quic-transport-23

Abstract

This document defines the core of the QUIC transport protocol. Accompanying documents describe QUIC's loss detection and congestion control and the use of TLS for key negotiation.

Note to Readers

Discussion of this draft takes place on the QUIC working group mailing list (quic@ietf.org), which is archived at <https://mailarchive.ietf.org/arch/search/?email_list=quic>.

Working Group information can be found at <https://github.com/quicwg>; source code and issues list for this draft can be found at <https://github.com/quicwg/base-drafts/labels/-transport>.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at https://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on March 15, 2020.

HTTP/3.0 is used by 3.0% of all the websites.

Usage of QUIC for websites, 25 Sep 2019, W3Techs.com

Popular sites using QUIC

- Google.com
- Youtube.com
- Blogspot.com
- Google.co.in
- Google.com.hk
- Google.co.jp
- Google.com.br
- Google.ru
- Google.de
- Tumblr.com

QUIC birth at Google

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 Tenneti, 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 improve-

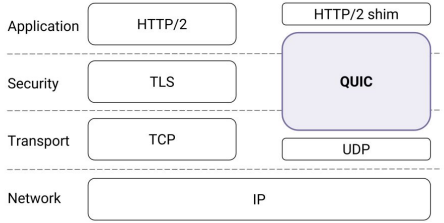
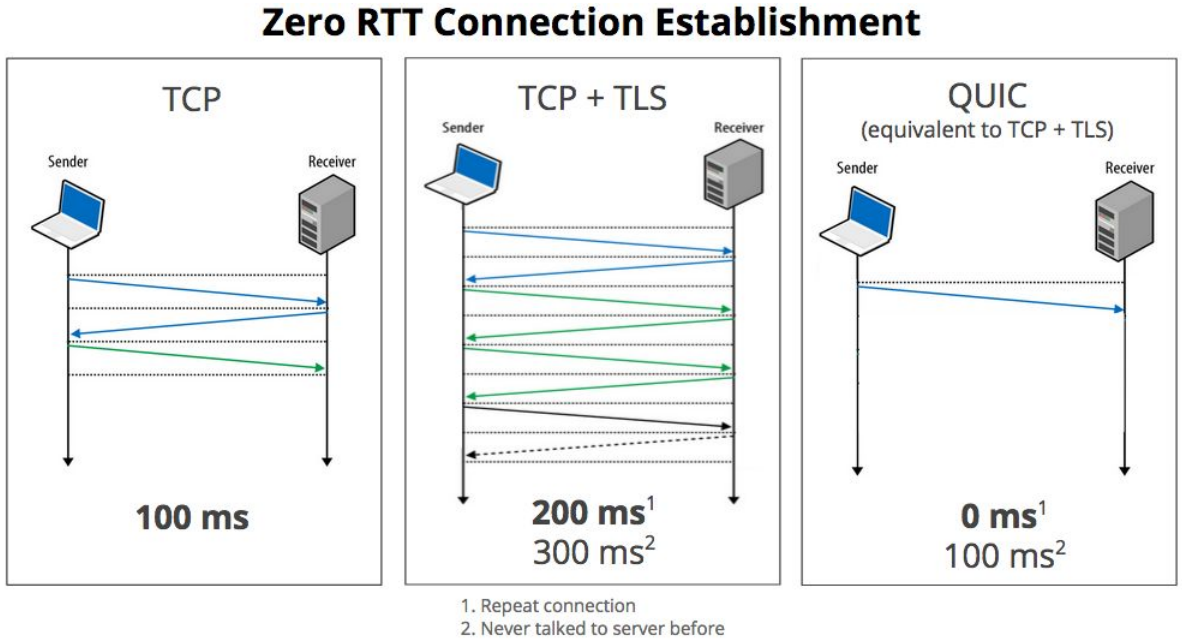


Figure 1: QUIC in the traditional HTTPS stack.

source: http://tiny.cc/r7rddz

What is QUIC?



source: https://tinyurl.com/oymkqzf

What is QUIC?

QUIC (**Q**uick **U**DP **I**nternet **C**onnections) is a new transport protocol for the internet, developed by Google.

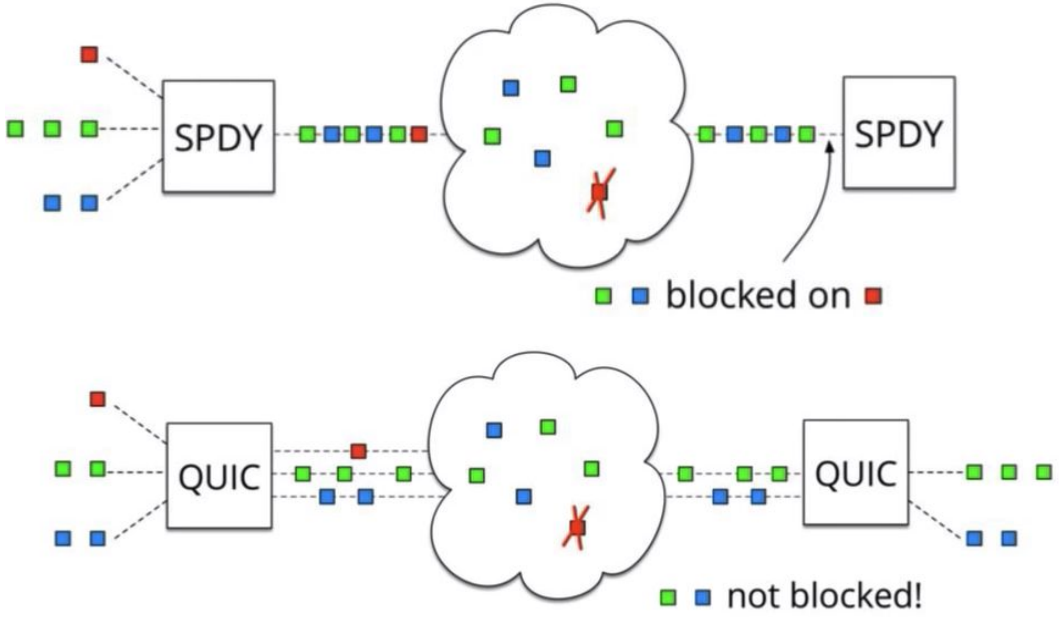
QUIC solves a number of transport-layer and application-layer problems experienced by modern web applications, while requiring little or no change from application writers. QUIC is very similar to TCP+TLS+HTTP2, but implemented on top of UDP. Having QUIC as a self-contained protocol allows innovations which aren't possible with existing protocols as they are hampered by legacy clients and middleboxes.

Key advantages of QUIC over TCP+TLS+HTTP2 include:

- Connection establishment latency
- Improved congestion control
- Multiplexing without head-of-line blocking
- Forward error correction
- Connection migration

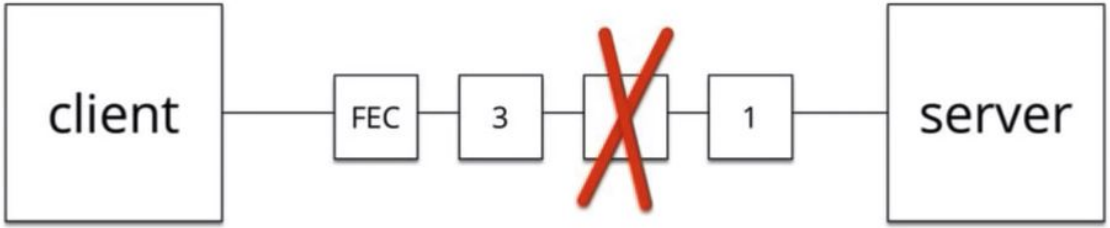
source: https://tinyurl.com/oymkqzf

No head-of-line blocking in QUIC



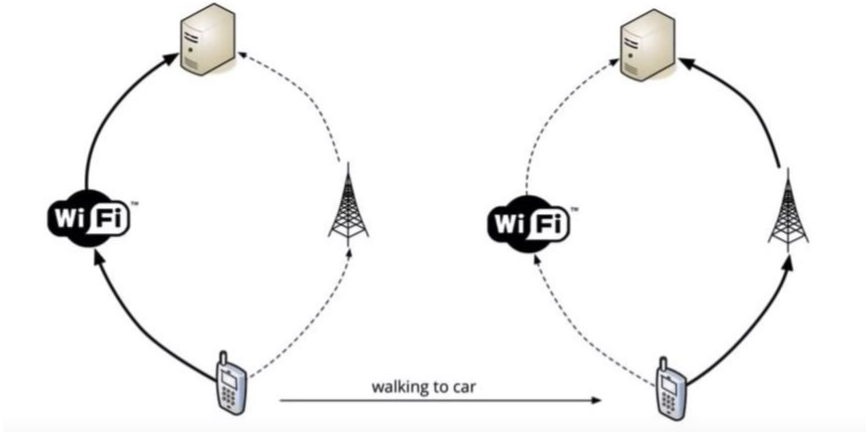
source: http://bit.ly/1gqR7WN

Forward Error Correction



source: http://bit.ly/1gqR7WN

Parking Iot Problem



- TCP Identification
  - Source IP
  - Destination IP
  - Source Port
  - Destination Port
- QUICK Identification
  - Unique ID (64 bits)

source: http://bit.ly/1gqR7WN

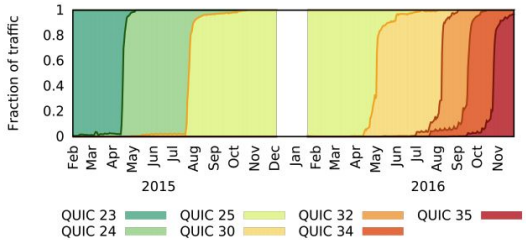
Parking lot Problem: Connection Migration Demo

https://drive.google.com/file/d/1DlMI_3MOxnWarvEVfzKxFqmD7c-u1cYG/view

Why not to improve TCP?

- “Middleboxes” - hardware dependency (custom TCP implementations)
- OS kernel dependency (kernel space vs user space)
- Testing new versions takes years
- Global deployment in decades…
- Example: Python 2 > 3 transition (over 10 years..)

While QUICK allows to deploy fast:



New versions are being deployed on a monthly basis:

- Q019: Adds session/connection level flow control
- Q020: Allow endpoints to set different stream/session flow control windows
- Q021: Crypto and headers streams are flow controlled (at stream level)
- Q023: Ack frames include packet timestamps
- Q024: HTTP/2-style header compression
- Q025: HTTP/2-style header keys. Removal of error_details from the RST_STREAM frame.
- Q026: Token binding, adds expected leaf cert (XLCT) tag to client hello
- Q027: Adds a nonce to the server hello
- Q029: Server and client honor QUIC_STREAM_NO_ERROR on early response

Middleboxes: RFC 3234

RFC 3234 Middleboxes: Taxonomy and Issues February 2002

Table of Contents

Example TCP improvements being slowed down:

- Stream Control Transmission Protocol (SCTP)
- Congestion Manager (CM)
- TCP Fast Open (TFO)
- Structured Stream Transport (SST)

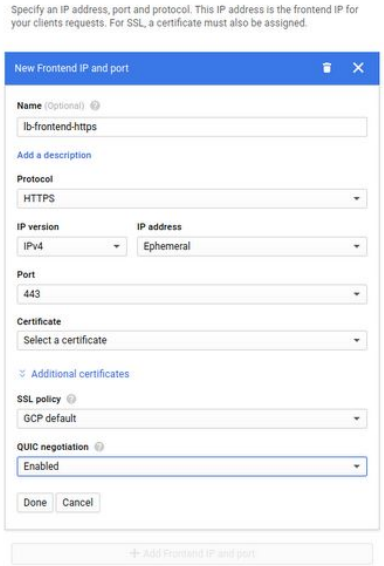
QUICK versions

```
// Version 40 was an attempt to convert QUIC to IETF frame format; it was
//            never shipped due to a bug.
// Version 41 was a bugfix for version 40.  The working group changed the wire
//            format before it shipped, which caused it to be never shipped
//            and all the changes from it to be reverted.  No changes from v40
//            or v41 are present in subsequent versions.
// Version 42 allowed receiving overlapping stream data.

QUIC_VERSION_43 = 43,  // PRIORITY frames are sent by client and accepted by
                       // server.
// Version 44 used IETF header format from draft-ietf-quic-invariants-05.

// Version 45 added MESSAGE frame.

QUIC_VERSION_46 = 46,  // Use IETF draft-17 header format with demultiplexing
                       // bit.
QUIC_VERSION_47 = 47,  // Allow variable-length QUIC connection IDs.
QUIC_VERSION_48 = 48,  // Use CRYPTO frames for the handshake.
QUIC_VERSION_49 = 49,  // Client connection IDs, long header lengths, IETF
                       // header format from draft-ietf-quic-invariants-06.
QUIC_VERSION_99 = 99,  // Dumping ground for IETF QUIC changes which are not
                       // yet ready for production.
```

Note: Different versions require different implementation effort and tests so it is not easy to predict the next major release date.

https://cs.chromium.org/chromium/src/net/third_party/quiche/src/quic/core/quic_versions.h?q=quic_versions.h

Easy to deploy in GCP



HTTP/3 issues

- UDP is blocked in some organizations
- Unoptimized UDP stacks (including hardware)
- Potential DDoS/DoS attacks
- Fall-back algorithms have to be implemented
- CPU consumption increase
- TLS layer implementation needs a significant effort
- QUIC is user-space implemented
- Lack of tooling
- [https:// :443] old urls support? (Alt-Svc: response header)

Example benefits

- “5% faster on average”
- “1 second faster for web search at 99th-percentile”
- “30% fewer rebuffers (video pauses)”
- “Over 50% of the latency improvement (at median and 95th-percentile)”
- “Over 10x fewer timeout based retransmissions improve tail latency and YouTube video rebuffer rates”

Depends on network quality, more features and optimizations on the way.

It is expected QUIC will be highly customizable (parameters, enable/disable flags, etc..)

Other protocols: QUIC & MQTT

Table 4
Comparison of IoT application layer protocols.

Protocol	CoAP	MQTT	MQTT-SN	XMPP	REST	AMQP	MQTTw/QUIC
UDP Compatible	√	X	√	X	X	√	√
TCP Compatible	X	√	√	√	√	√	X
Multiplexing Capability	X	X	X	X	X	X	√
0-RTT Capable	X	X	X	X	X	X	√
Fixing Head-of-Line Blocking	N/A	X	X	X	X	X	√
Fixing TCP Half-Open Problem (Adaptability to Lossy Networks)	N/A	X	X	X	X	X	√
Supporting Connection Migration	X	X	X	X	X	X	√

"The results confirmed that MQTTw/QUIC reduces connection establishment overhead, lowers delivery latency, reduces processor and memory utilization, and shortens the level and duration of throughput degradation during connection migration significantly compared to MQTTw/TCP."

https://www.researchgate.net/publication/328380610_Implementation_and_Analysis_of_QUIC_for_MQTT

QUIC Implementations

- Aioquic - https://github.com/aiortc/aioquic (Python, died 2 months ago?)
- Facebook: https://github.com/facebookincubator/mvfst
- Caddy: https://github.com/lucas-clemente/quic-go
- H2O: https://github.com/h2o/quicly
- LiteSpeed: https://github.com/litespeedtech/lsquic
- Cloudflare: https://github.com/cloudflare/quiche
- Microsoft: closed (winquic.dll / winquic.sys)
- Google:
  - https://quiche.googlesource.com/quiche
  - https://cs.chromium.org/chromium/src/net/third_party

Full list: https://github.com/quicwg/base-drafts/wiki/Implementations

QUIC Tools: quick-trace, QUICvis, Wireshark

QUIC trace utilities

This repository contains a format for recording a trace of QUIC connection, together with the tools for analyzing the resulting traces. This format of traces is currently partially supported by Chromium, and we hope for more implementations to adopt it in the future.

The primary focus of this format is debugging congestion-control related issues, but other transport aspects (e.g. flow control) and transport-related aspects of the application (e.g. annotations for the types of data carried by the streams) are also within the scope.

How to record this format

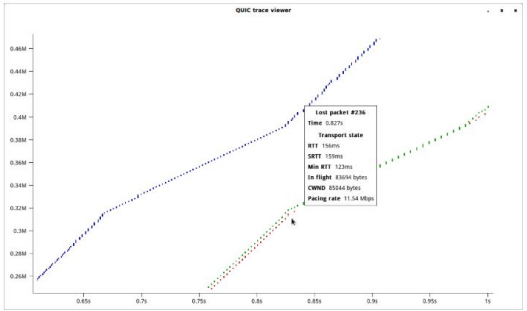
The traces are represented as a Protocol Buffer, which is completely described in `lib/quic_trace.proto`. Projects that use Bazel can embed this repository directly and use the provided Bazel rules.

OpenGL-based renderer

This repository contains two different tools to visualize the trace:

1. A simple gnuplot-based trace renderer.
2. A fully featured OpenGL-based renderer.

The OpenGL-based is the current preferred way of rendering traces. It supports Linux and macOS, though Windows support is coming soon. OpenGL 3.3 or later is required. Most of its dependencies are shipped using Bazel build files, but for Linux, system SDL2 is used.



QUICvis

> A set of tools to analyze the IETF QUIC protocol.

Build Setup

```
# install dependencies for API server to load files
cd ./apiserver
npm install
cd ../

# install dependencies for the website
cd ./visualisation-tool
npm install
npm install typescript
```

Run website

```
# First open a terminal for the API server and run
cd ./apiserver
npm run start

# Then open a second terminal for the website and run
cd ./visualisation-tool
npm run serve
```

Networking/Computing Tips/Tricks

Using Wireshark to Analyze QUIC/GQUIC Traffic

Created: Wednesday, 07 March 2018 14:54
Hits: 8177

As explained in our prior article on QUIC/GQUIC, you may be seeing a lot of GQUIC traffic in your packet captures. Assuming you have read that article, and understand that all GQUIC traffic is encrypted, you know the only way to see some of the details is using Chrome itself. That does not mean that you can't use Wireshark as well.

First, we have created a GQUIC profile for Wireshark. It is in our Profiles repository - you can find that repository here. You will need to download and install that profile in your Wireshark personal configuration.

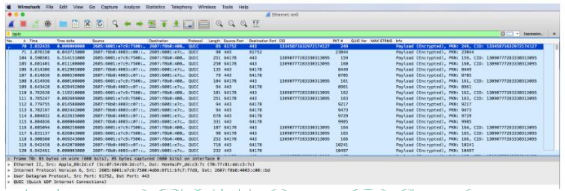
As you dive deeper into GQUIC, you may want to refer to the detailed RFC drafts. Surprisingly there are approximately 30 internet draft specifications dealine with QUIC. You can find those drafts here.

Next, you will want to perform a simple capture on your computer by:

1. Start Wireshark on your working Internet interface, making sure the GQUIC profile is selected
2. Open Chrome, and then going to Google's home page (www.google.com), and then click on the "I'm Feeling Lucky" button.
3. Stop the capture.

Using the profile, click on the Display Filter bookmark, and select the GQUIC packets only filter:

You should now only see GQUIC packets. Let's discuss some of the columns I created (note here – I took the screenshots below before the protocol name changed to GQUIC – so while I updated my profile at the repository, I do not show the updates below – but it is easy to resolve – anywhere it says "quic" – simply change in your mind or on your latest version of Wireshark to "gquic" and you are good):



IETF QUIC vs Google QUIC

QUIC Prototype Protocol Discussion group ›

What's difference between IETF quic and google quic?

3 posts by 2 authors

James Mike

Hi there.
I am wonder what's difference between IETF quic and gquic? Is there any document for that?
In the future, is it possible for Google to completely abandon gquic and switch to ietf quic?
Thanks a lot!

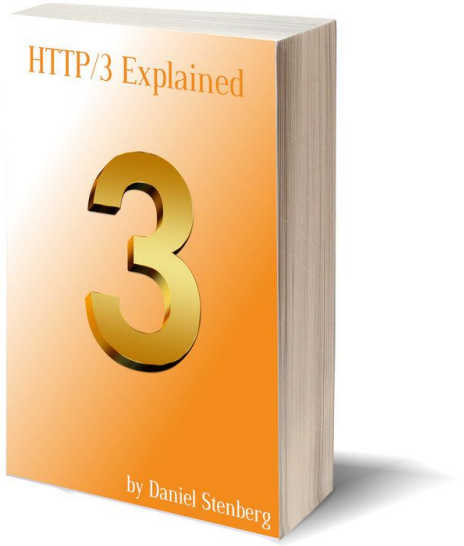
Ian Swett

There are so many differences, and there is not a document for it. I'd recommend reading the IETF specs to find out more about IETF QUIC.

Yes, Google intends to switch to IETF QUIC ASAP, but there's no specific timeframe.

https://groups.google.com/a/chromium.org/forum/#!topic/proto-quic/vX5VV8pLxAQ

Free book: HTTP/3 explained



HTTP/3 explained

HTTP/3 is the to-become next generation of the HTTP protocol family.

QUIC is a new reliable transport protocol that could be viewed as a sort

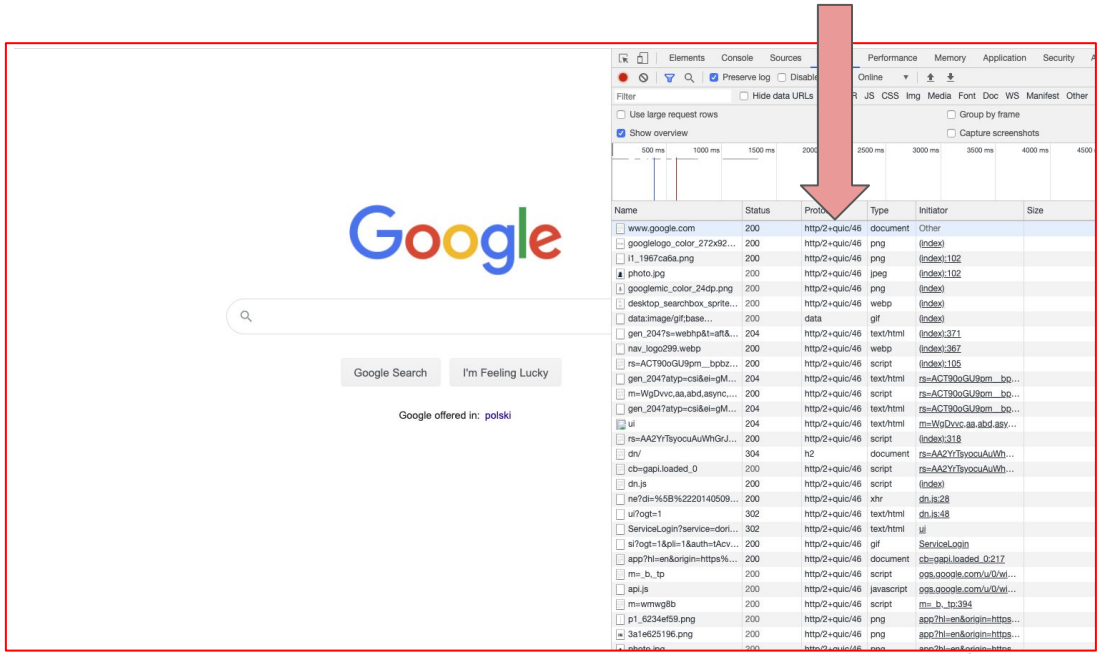
HTTP/3 explained is a free and open booklet describing the HTTP/3 an

Deutsche	Web	PDF	Mobi	ePub
English	Web	PDF	Mobi	ePub
Français	Web	PDF	Mobi	ePub
Italiano	Web	PDF	Mobi	ePub
Română	Web	PDF	Mobi	ePub
日本語	Web	PDF	Mobi	ePub
简体中文	Web	PDF	Mobi	ePub
한국어	Web	PDF	Mobi	ePub

https://daniel.haxx.se/http3-explained

(Daniel Stenberg, the author and the maintainer of the cURL)

Example: Chrome + http/2 + quic/46



Laboratory test: Vegeta + netem performance testing

Vegeta build passing | fuzzit passing | go report A+ | godoc reference | gitter join chat | donate bitcoin

Vegeta is a versatile HTTP load testing tool built out of a need to drill HTTP services with a constant request rate. It can be used both as a command line utility and a library.

netem

netem provides Network Emulation functionality for testing protocols by emulating the properties of wide area networks. The current version emulates variable delay, loss, duplication and re-ordering.

If you run a current 2.6 distribution, (Fedora, OpenSuse, Gentoo, Debian, Mandriva, Ubuntu), then netem is already enabled in the kernel and a current version of iproute2 is included. The netem kernel component is enabled under:

```
Networking -->
  Networking Options -->
    QoS and/or fair queuing -->
       Network emulator
```

Netem is controlled by the command line tool 'tc' which is part of the iproute2 package of tools. The tc command uses shared libraries and data files in the /usr/lib/tc directory.

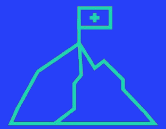
Added QUIC support:
https://github.com/Ottovsky/vegeta/commit/f3ebe35dd3b9916161de5e288da1bb427da647ff


QUESTIONS?

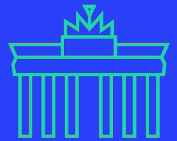
Container Solutions

Amsterdam

London

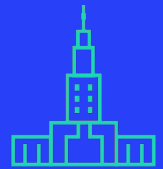


Zurich



Berlin

Montreal



Warsaw