Towards the Real-Time Web: SPDY, HTTP/2, WebSocket QUIC and WebRTC

Prof. Wing C. Lau
IERG5090
Spring 2017

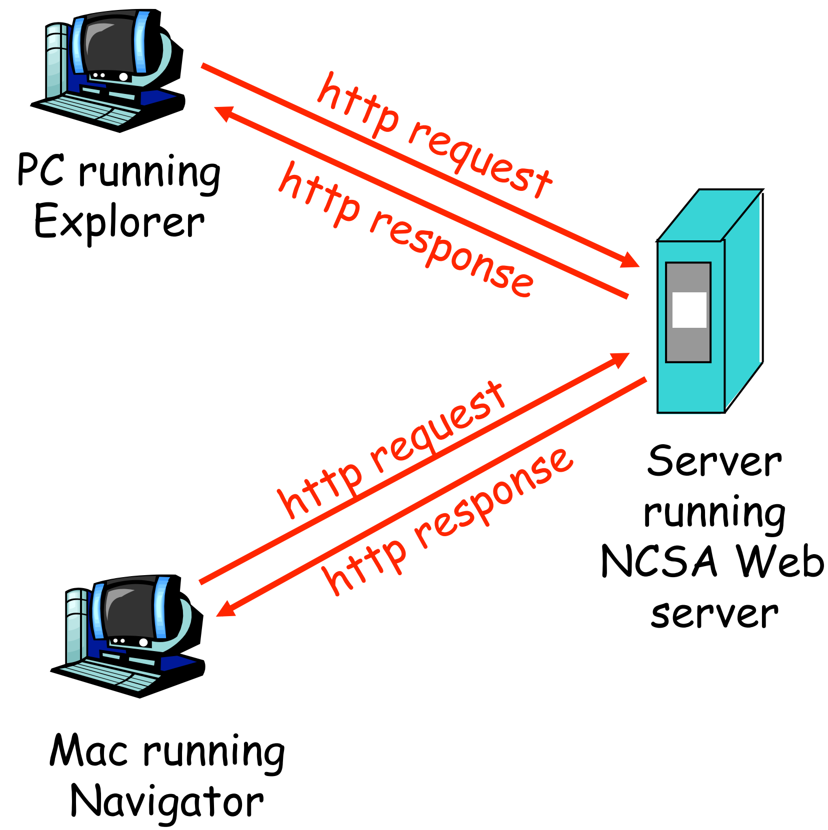
Acknowledgements

The following Lecture Slides are adapted from various sources including those shown below. The copyright of the materials belongs to the original authors:

- http://www.html5rocks.com/en/tutorials/websockets/basics/
- http://www.codeproject.com/Articles/531698/Introduction-to-HTML5-WebSocket
- http://www.slideshare.net/mobile/MarceloJabali/html5-websocket-introduction
- http://www.slideshare.net/peterlubbers/html5-realtime-and-connectivity
- http://www.infoq.com/presentations/Real-time-Web-WebSocket-SPDY
- http://html5videoguide.net/presentations/WebDirCode2012
- http://www.chromium.org/spdy/
- http://en.wikipedia.org/wiki/SPDY
- IERG3090 Lecture Notes of Prof. Jack Lee
- Presentation slides of Lien Gao and Tujia Chen of CMSC5709
- Ilya Grigorik, "HTTP/2 (RFC7540) is here, let's optimize" O'Reilly Velocity Conference, May 2015
- Jana Iyengar, "QUIC – Redefining the Internet," IETF93 BarBOF.

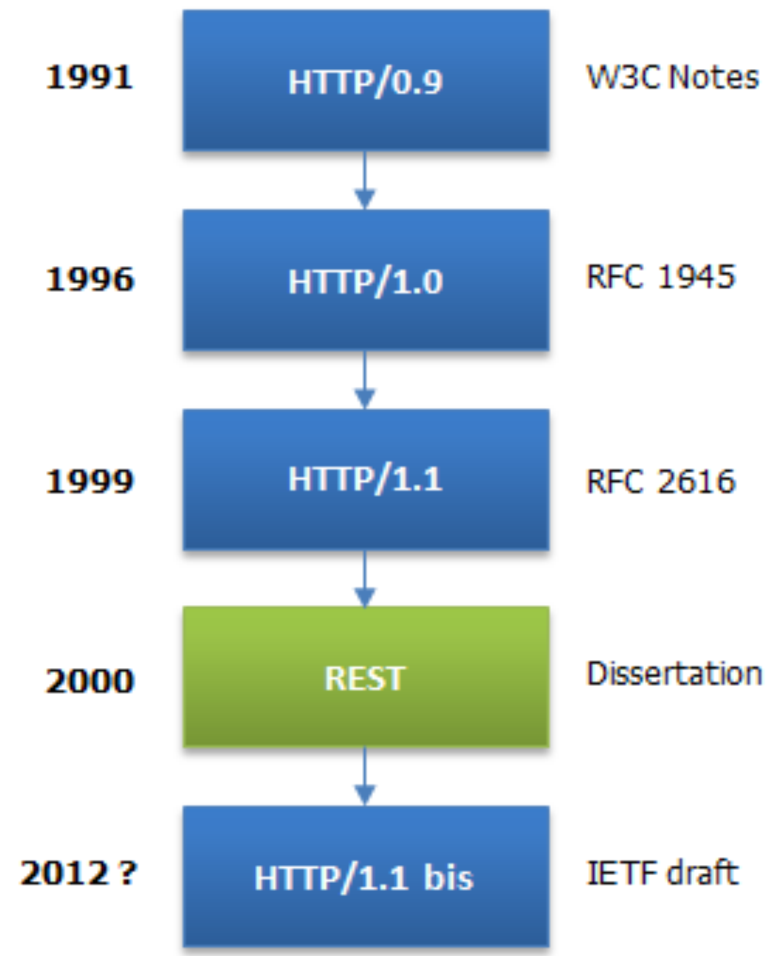
HTTP: hypertext transfer protocol

- WWW's application layer protocol
- client/server model
  - *client:* browser that requests, receives, "displays" WWW objects
  - *server:* WWW server sends objects in response to requests
  - Stateless
- Protocol Encoding: text-based in readable English



Evolution of Hyper-Text Transfer Protocol (HTTP)

- 1991 – Version 0.9 (first specification as W3C Notes written by Tim Berners-Lee and his team)
  - http://www.w3.org/Protocols/HTTP/AsImplemented.html
- 1996 – Version 1.0 (RFC1945)
  - http://tools.ietf.org/html/rfc1945
- 1999 – Version 1.1 (RFC2616)
  - and the formalization of REST-style architecture of the Web by W3C, with major contributions made by Roy T. Fielding.
  - Current standard used by most web servers/browsers

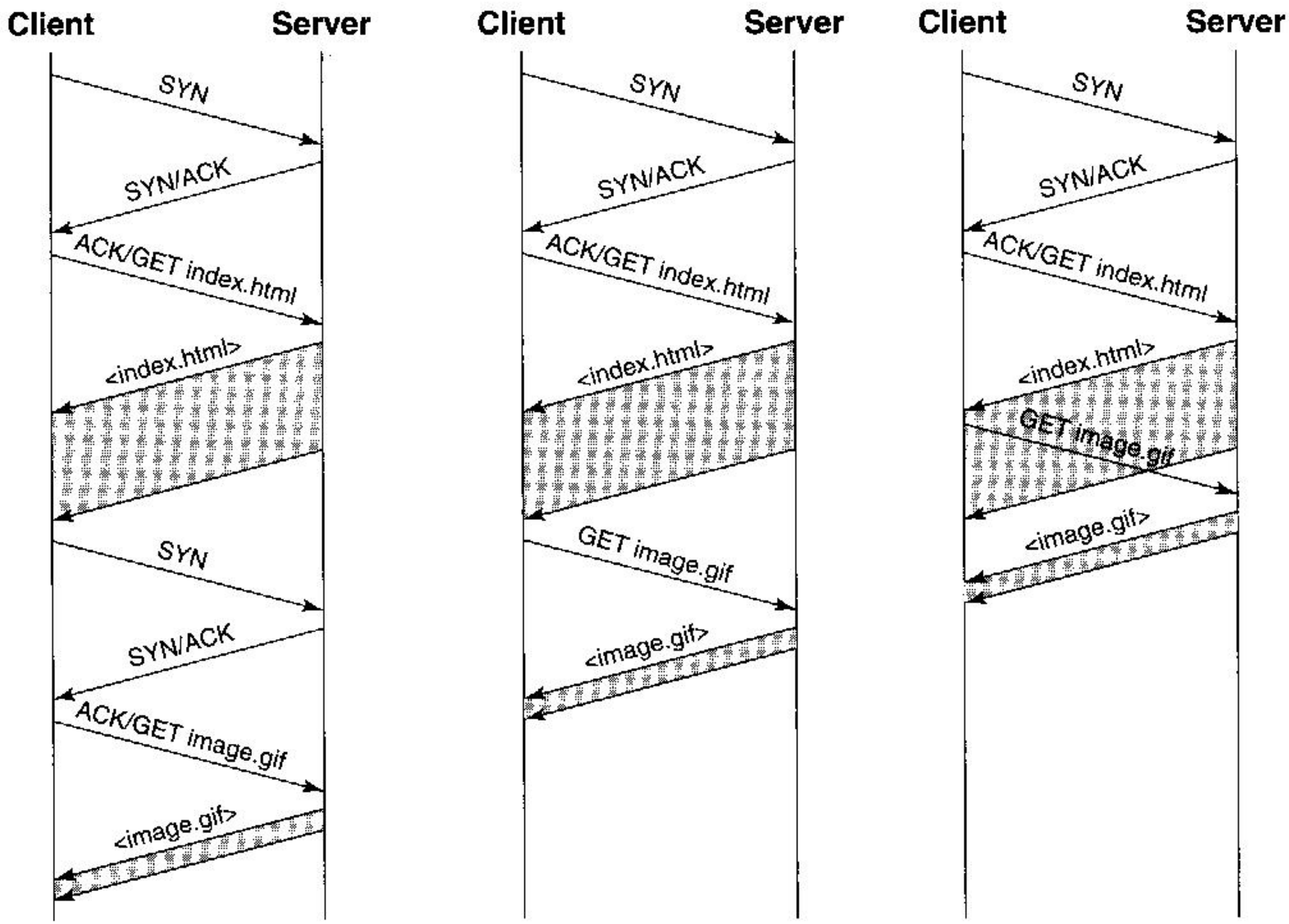


REST = REpresentational State Transfer

Shortcomings of HTTP/1.0

- non-persistent connection: New connection for each request puts burden on server:
  - Each TCP connection must be established and managed
  - Each TCP connection allocates send and receive buffers and maintains state variables
- Each object suffers 2 round-trip times of delay
  - partially alleviated by using multiple parallel connections
- Each object suffers TCP slow-start delay
  - also partially alleviated by using multiple parallel connections
- Limited cache control

Non-persistent Vs. Persistent Vs. Pipelined



(*a*) One HTTP interaction per TCP connection

(*b*) Persistent TCP connections

(*c*) Persistent TCP connections with pipelining

Improvements in HTTP/1.1

- Persistent connections: allows connections to remain open over several requests
- Request pipelining (default for HTTP/1.1)
- Introduces a variety of directives to control caching on proxies and in clients
- new protocol tracing feature for debugging proxy chains

Persistent HTTP

Nonpersistent HTTP issues:

- requires 2 RTTs per object
- OS must work and allocate host resources for each TCP connection
- but browsers often open parallel TCP connections to fetch referenced objects

Persistent HTTP

- server leaves connection open after sending response
- subsequent HTTP messages between same client/server are sent over connection

Persistent without pipelining:

- client issues new request only when previous response has been received
- one RTT for each referenced object

Persistent with pipelining:

- default in HTTP/1.1
- client sends requests as soon as it encounters a referenced object
- as little as one RTT for all the referenced objects

Challenges of Pipelined HTTP

- HTTP is supposed to be stateless but some web sites implement stateful web sessions anyway using techniques such as cookies or URL rewrite.
- If a web session is stateful then the sequence of requests generation and execution may become inter-dependent (i.e., non-idempotent).
- How to determine if a web session is stateful, and if it is safe to send the subsequent requests before prior request is completed?
- Not all servers/proxies implement pipelining correctly.
- Head-of-line blocking
  - A request loading a large object (e.g., large image) may block the delivery of subsequent objects.
  - All subsequent pipelined requests will be blocked by the head-of-line request as requests are processed in FIFO manner.
  - This can be circumvented using HTTP Range request.

Concurrent HTTP Sessions

- Implemented by most browsers
- After the initial HTTP connection which retrieves the HTML body, initiate multiple (4~6) HTTP sessions (per domain) to retrieve multiple objects (e.g., images) in parallel.
- Purposes
  - Effectively multiplies the congestion window size by the number of HTTP connections
  - Potentially overlaps the server-side processing time of multiple HTTP requests

What is SPDY ?

- SPDY (pronounced speedy) was an experimental networking protocol developed primarily at Google for transporting web content.
- Although not a standard protocol, the group developing SPDY submitted it to IETF as the initial basis of HTTP/2 standardization.
- SPDY had reference implementations early on in both Google Chrome and Mozilla Firefox.
- SPDY is similar to HTTP, with particular goals to reduce web page load latency and improve web security.
- SPDY achieves reduced latency through compression, multiplexing, and prioritization.
- In lab tests, SPDY was shown to achieve up to 64% reductions in page load times compared to HTTP.

Source: Wikipedia and SPDY's official whitepaper and protocol specification, available at http://www.chromium.org/spdy/

Design Goals for SPDY

- To target a 50% reduction in page load time.
- To minimize deployment complexity. SPDY uses TCP as the underlying transport layer, so requires no changes to existing networking infrastructure.
- To avoid the need for any changes to content by website authors. The only changes required to support SPDY are in the client user agent and web server applications.
- To bring together like-minded parties interested in exploring protocols as a way of solving the latency problem. The SPDY team hopes to develop this new protocol in partnership with the open-source community and industry specialists.

Recap: Limitations of HTTP over TCP = SPDY's Design Focus

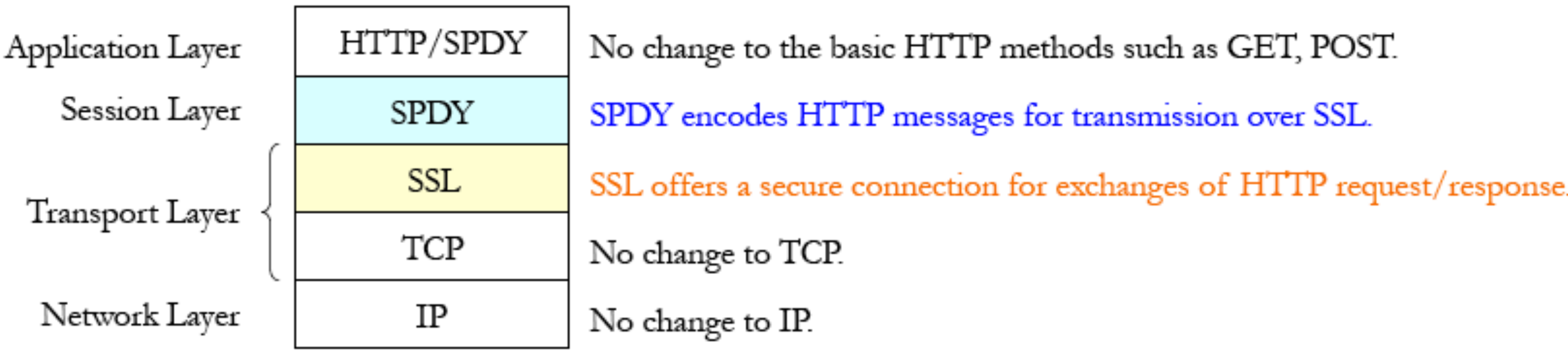
- Single HTTP request per TCP connection. Even Pipelined HTTP is FIFO only.
- Allow only client-initiated request. Server cannot push an data object to the client.
- No compression of HTTP request and response headers (various from hundreds to several KBs, depending on cookies and user agent strings).
- Redundant HTTP header fields across multiple requests on the same session (e.g., User-Agent seldom changes for the same client).
- Content compression is optional rather than mandatory.

Specific Technical Goals for SPDY

- To allow many concurrent HTTP requests to run across a single TCP session.
- To reduce the bandwidth currently used by HTTP by compressing headers and eliminating unnecessary headers.
- To define a protocol that is easy to implement and server-efficient. The SPDY team hopes to reduce the complexity of HTTP by cutting down on edge cases and defining easily parsed message formats.
- To enable the server to initiate communications with the client and push data to the client whenever possible.
- To make SSL the underlying transport protocol, for better security and compatibility with existing network infrastructure.
    - Mandatory Use of SSL by SPDY has been a quite Controversial Decision !
    - Although SSL does introduce a latency penalty, the SPDY team believes that the long-term future of the web depends on a secure network connection.
    - The use of SSL is necessary to ensure that communication across existing proxies is not broken.

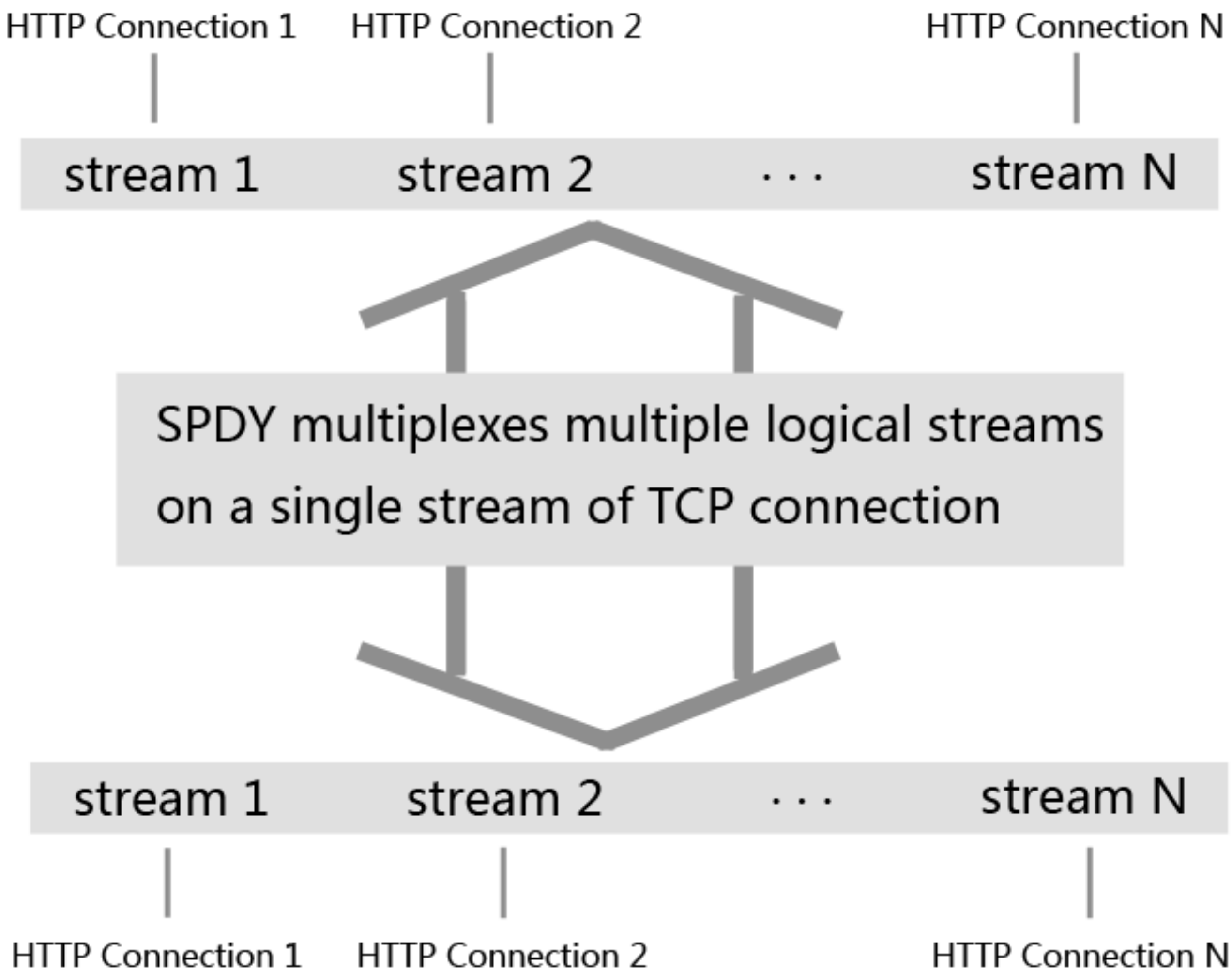
Architecture of SPDY

- SPDY acts as a session layer between HTTP and SSL/TCP



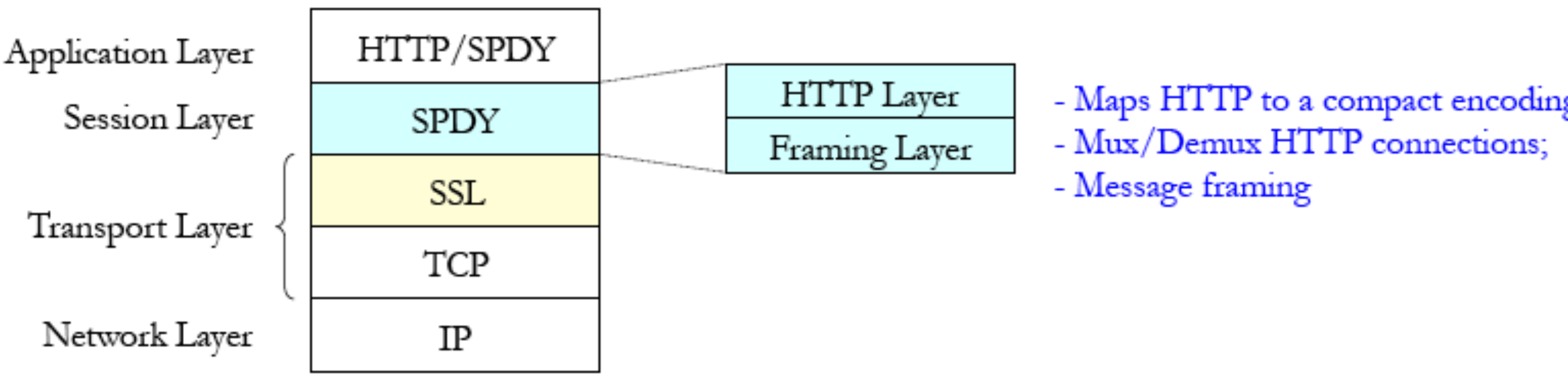
- SPDY sessions are bi-directional and can be initiated by both the client and the server.

Multiplexing HTTP Streams over SPDY



Architecture of SPDY (cont'd)

- The SPDY Specification is split into two parts:
  - A Framing layer, which multiplexes independent, length-prefixed frames into a SSL/TCP connection, and
  - an HTTP layer, which specifies the mechanism for overlaying HTTP request/response pairs on top of the framing layer.



HTTP Layering over SPDY

- The features of HTTP are *mostly* unchanged.
- All of the application request and response header semantics are preserved, although the syntax of conveying those semantics has changed.
- The rules from the HTTP/1.1 specification in RFC2616 apply with some changes.
  - Connection Management
  - HTTP Request/Response
  - Server Push Transactions

Key Features of SPDY vs. HTTP

- Multiplexed requests
  - There is no limit to the number of requests that can be issued concurrently over a single SPDY connection.
- Prioritized requests
  - Clients can request certain resources to be delivered first. This avoids the problem of congesting the network channel with non-critical resources when a high-priority request is pending.
- Compressed headers
  - Clients today send a significant amount of redundant data in the form of HTTP headers. Because a single web page may require 50 or 100 sub-requests, this data is significant.
- Server pushed streams
  - Server Push enables content to be pushed from servers to clients without a request.

Performance of SPDY vs. HTTP/1.x

	DSL 2 Mbps downlink, 375 kbps uplink		Cable 4 Mbps downlink, 1 Mbps uplink	
	Average ms	**Speedup**	**Average ms**	**Speedup**
HTTP	3111.916		2348.188	
SPDY basic multi-domain* connection / TCP	2242.756	27.93%	1325.46	43.55%
SPDY basic single-domain* connection / TCP	1695.72	45.51%	933.836	60.23%
SPDY single-domain + server push / TCP	1671.28	46.29%	950.764	59.51%
SPDY single-domain + server hint / TCP	1608.928	48.30%	856.356	63.53%
SPDY basic single-domain / SSL	1899.744	38.95%	1099.444	53.18
SPDY single-domain + client prefetch / SSL	1781.864	42.74%	1047.308	55.40%

Average page load times for top 25 websites

Support and Usage of SPDY

- Browsers supporting SPDY:
  - Google Chrome/Chromium,
  - Firefox (version 11+, below 13 disabled by default)
    - It can be turned on through the network.http.spdy.enabled preference in about:config.
  - Opera browser (version 12.10+)
  - Amazon's Silk browser for the Kindle Fire uses the SPDY protocol to communicate with their EC2 service for Web page rendering.
- Services support SPDY
  - Many Google services (e.g. Google search, Gmail, Chrome-sync, Google-Ad-servers and other SSL-enabled services) use SPDY when available.
  - Twitter, Facebook, Jetty Web Server, F5 Networks, NGINX, Wordpress.com

Ever wonder how come Chrome is faster accessing certain web sites?

From SPDY to HTTP/2

- So should we all praise Google and switch to SPDY ? **Not Really !**
- Real-world performance gain of SPDY vs. https or http may not be as impressive as the lab-tests indicated:
  - http://www.guypo.com/technical/not-as-spdy-as-you-thought/
- SPDY will hit server and client CPUs much harder than traditional HTTP.
- Making SSL mandatory is a strange move.
  - Some argues that it would pave the way for more man-in-the-middle attacks.
- 1st Draft of HTTP/2 was published by the IETF httpbis working group on November 28, 2012, which is a direct copy of SPDY *bis = Encore (in Latin)
  - Changes in the protocol were made during the subsequent IETF standardization process which introduced various differences between HTTP/2 and SPDY.
- In Feb 2015, Google announced plans to remove support for SPDY in Chrome in favor of support for HTTP/2
- RFC7540 (HTTP/2) and RFC7541 (HPACK), both IETF proposed standards, were published in May 2015.

- HTTP/2 had already surpassed SPDY in adoption by May 2015.

"9% of all Firefox (M36) HTTP transactions are happening over HTTP/2. There are actually more HTTP/2 connections made than SPDY ones. This is well exercised technology."

Feb 18, 2015 - Patrick McManus, Mozilla

New TLS + NPN/ALPN connections in Chrome:

~27% negotiate HTTP/1
~28% negotiate SPDY/3.1
~45% negotiate HTTP/2

May 26, 2015 - Chrome telemetry

Can I use SPDY ? Settings

2 results found

Global 79.65%

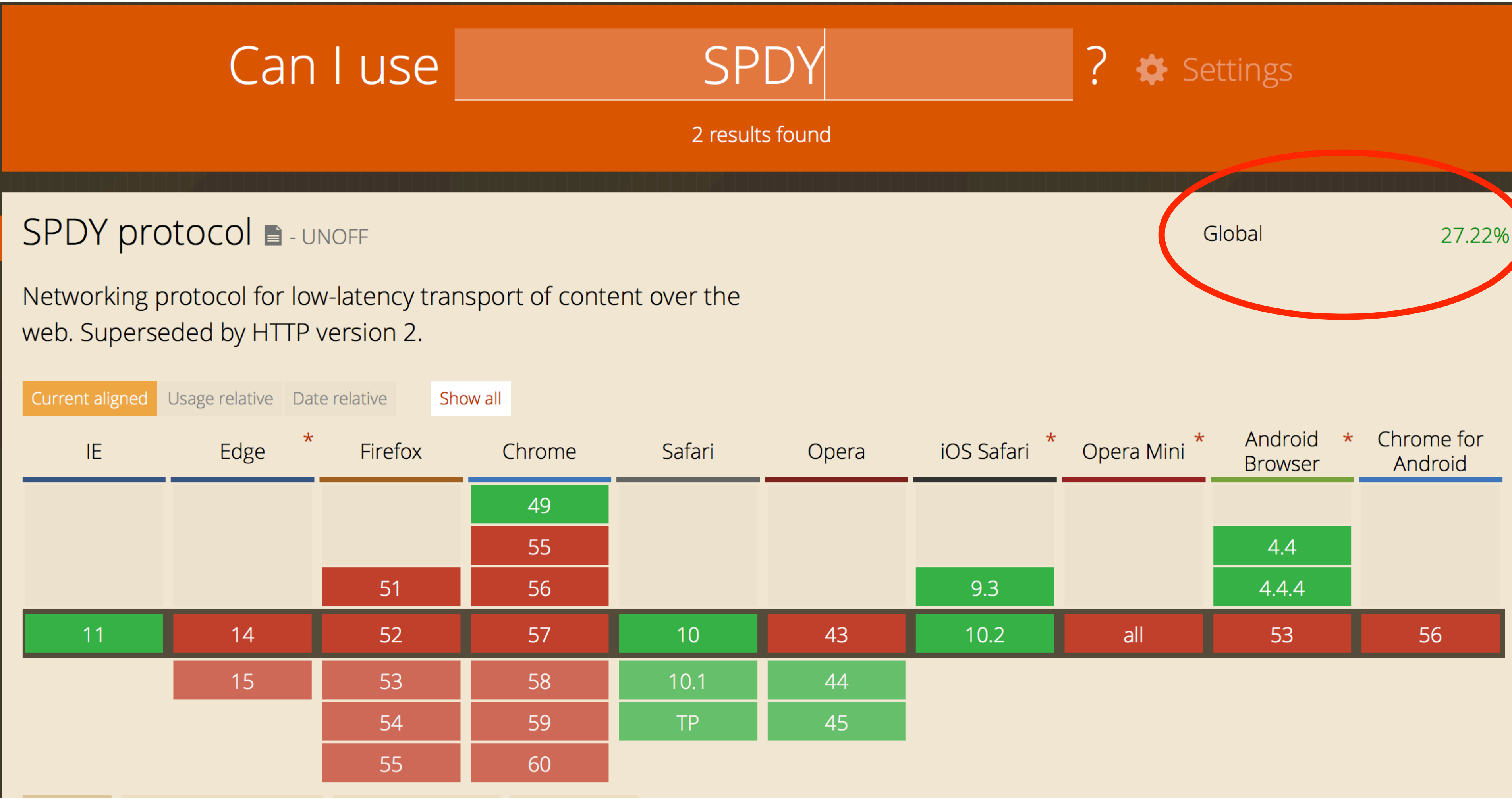
SPDY protocol - UNOFF

Networking protocol for low-latency transport of content over the web. Superseded by HTTP version 2.

Current aligned | Usage relative | Show all

IE	Firefox	Chrome	Safari	Opera	iOS Safari *	Opera Mini *	Android Browser *	Chrome for Android
		31						
		36						
		37						
		39					4.1	
8	31	40					4.3	
9	36	41					4.4	
10	37	42	7	28	7.1		4.4.4	
11	38	43	8	29	8.3	8	40	42
Edge	39	44		30				
	40	45		31				
	41	46						

http://caniuse.com/#feat=spdy



http://caniuse.com/#feat=spdy

Can I use HTTP2 ? Settings

1 result found

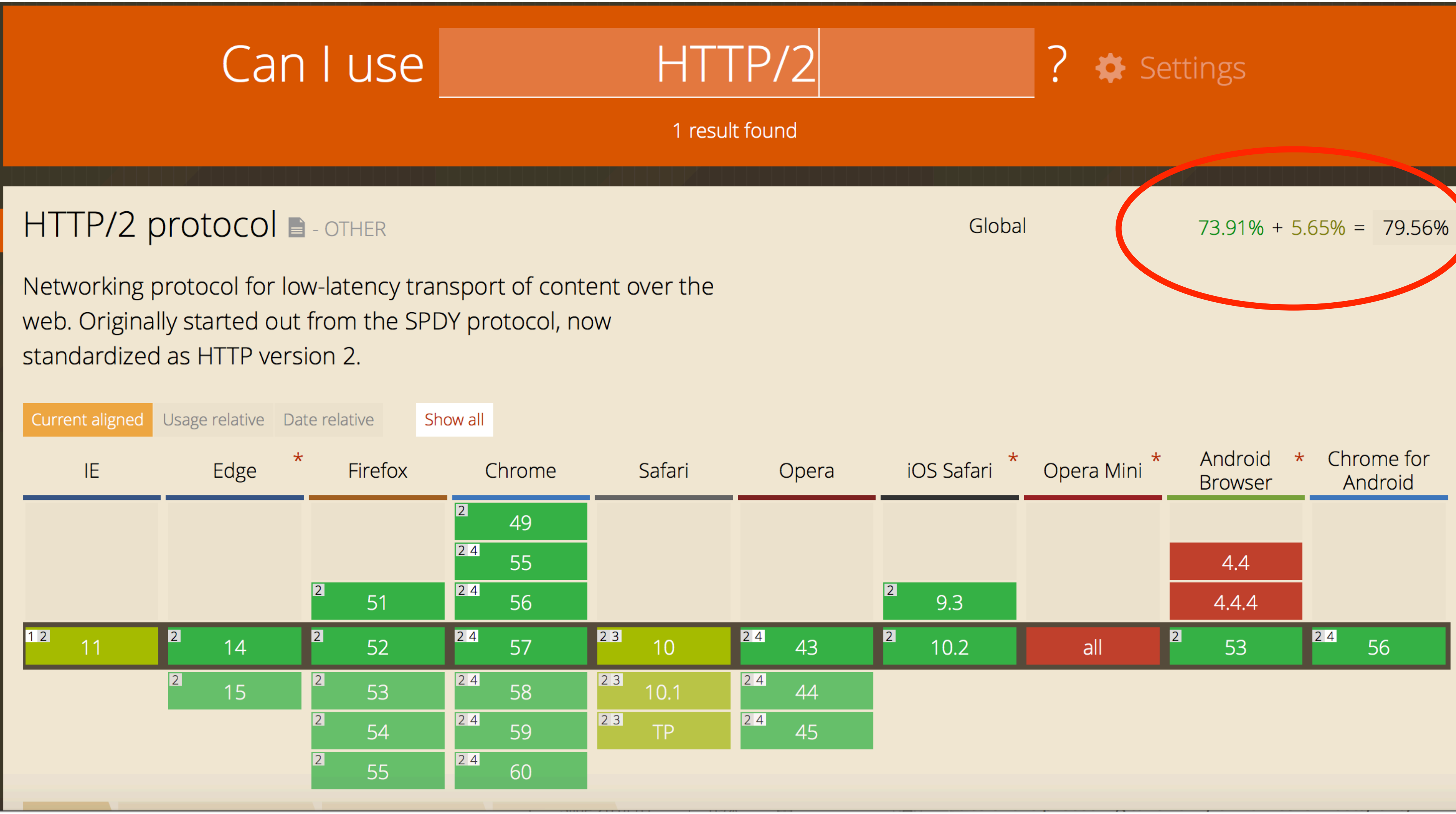
HTTP/2 protocol - OTHER

Global 48.12% + 7.58% = 55.69%

Networking protocol for low-latency transport of content over the web. Originally started out from the SPDY protocol, now standardized as HTTP version 2.

Current aligned | Usage relative | Show all

IE	Firefox	Chrome	Safari	Opera	iOS Safari *	Opera Mini *	Android Browser *	Chrome for Android
		31						
		36						
		37						
		39					4.1	
8	31	40					4.3	
9	36	41					4.4	
10	37	42	7	28	7.1		4.4.4	
[1] 11	38	43	8	29	8.3	8	40	42
Edge	39	44		30				
	40	45		31				
	41	46						



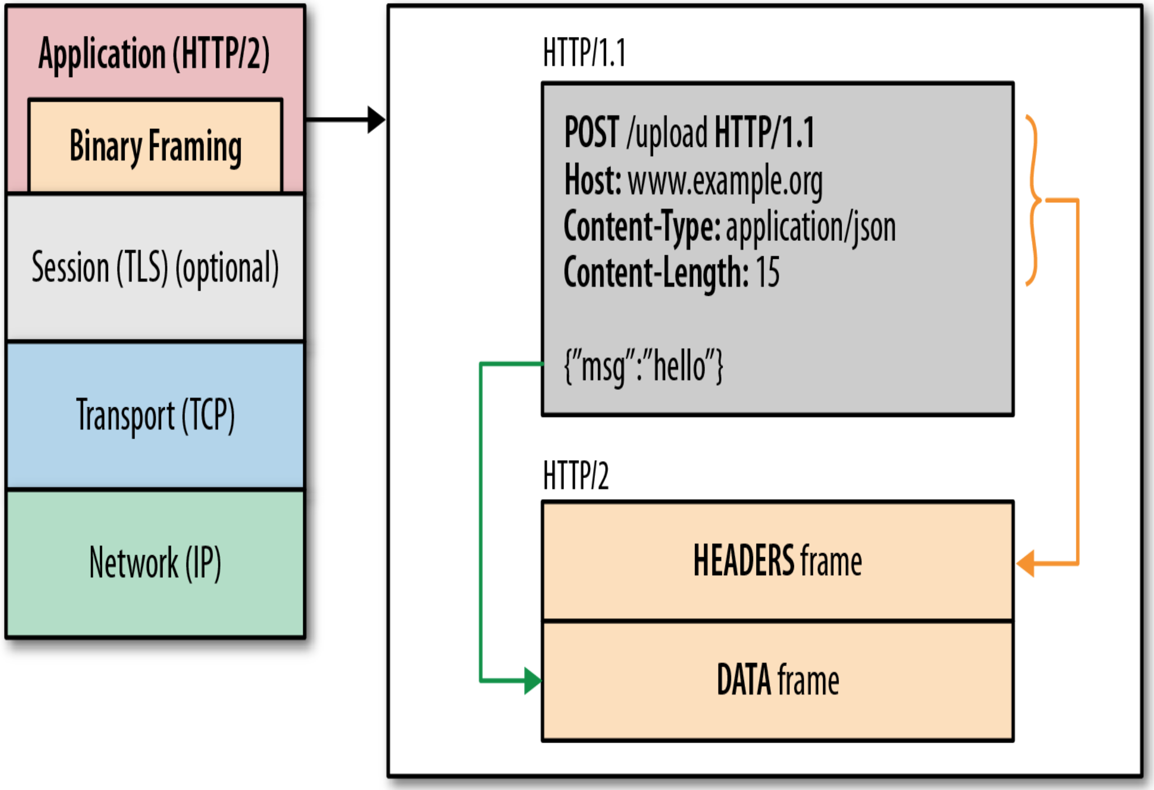
http://caniuse.com/#feat=HTTP%2F2

Differences b/w SPDY and HTTP/2

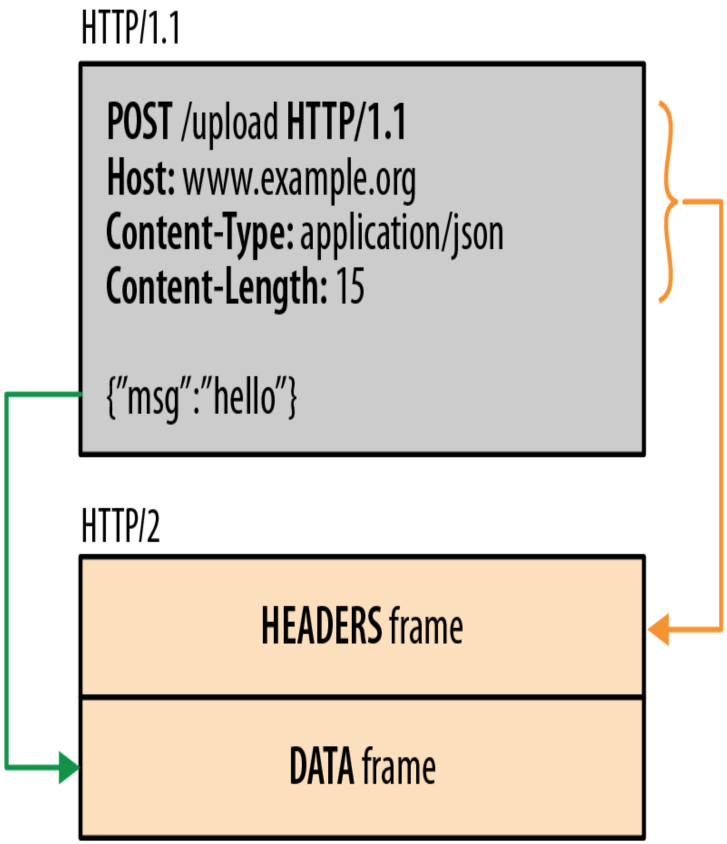
SPDY	HTTP/2
SSL Required. In order to use the protocol and get the speed benefits, connections must be encrypted.	**SSL Not Required.** *However* – even though the IETF doesn't require SSL for HTTP/2 to work, many popular browsers do require it. And because most Internet data is accessed through popular browsers (Chrome and Firefox), what they require matters most.
Fast Encrypted Connections. Does not use the ALPN extension that HTTP/2 uses.	**Faster Encrypted Connections.** The new ALPN extension lets browsers and servers determine which application protocol to use during the initial connection instead of after.
Single-Host Multiplexing. Multiplexing happens on one host at a time.	**Multi-Host Multiplexing.** Multiplexing happens on different hosts at the same time.
Compression. SPDY leaves a small space for vulnerabilities in its current compression methods.	**Faster, More Secure Compression.** HTTP/2 introduces HPACK, a compression format designed specifically for shortening headers and preventing vulnerabilities.
Prioritization. While prioritization is available with SPDY, HTTP/2's implementation is more flexible and friendlier to proxies.	**Improved Prioritization.** Lets web browsers determine how and when to download a web page's content more efficiently.

HTTP/2 Architecture Overview

1. **One TCP connection**
2. **Request → Stream**
   - Streams are multiplexed
   - Streams are prioritized
3. **Binary framing layer**
   - Prioritization
   - Flow control
   - Server push
4. **Header compression (HPACK)**

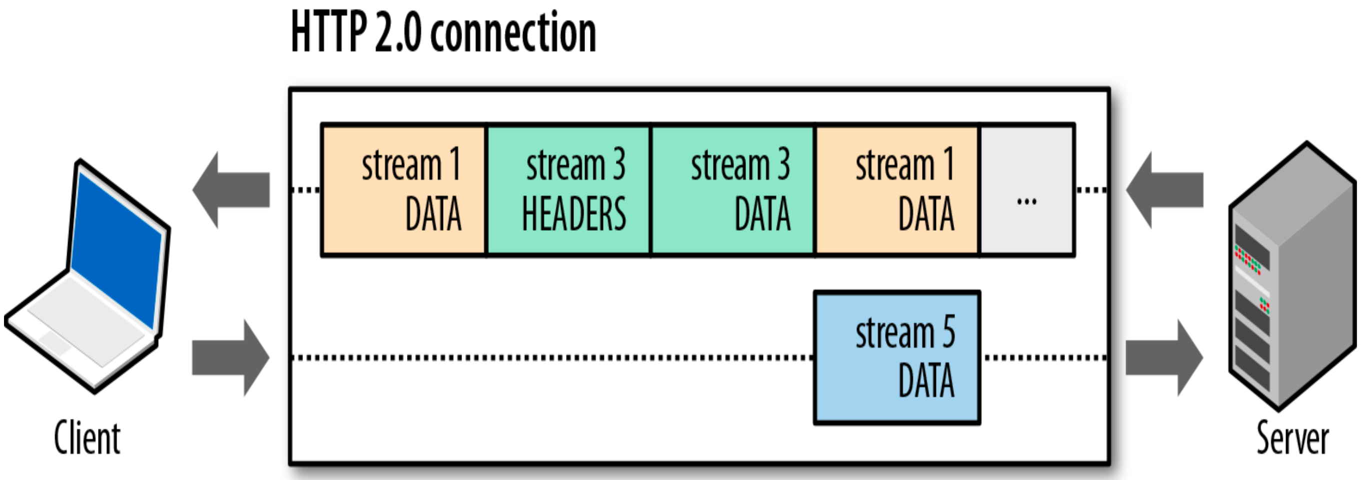


HTTP/2 binary framing 101



- **HTTP messages are decomposed into one or more frames**
  - HEADERS for meta-data
  - DATA for payload
  - RST_STREAM to cancel
  - ...
- **Each frame has a common header**
  - 9-byte, length prefixed
  - Easy and efficient to parse

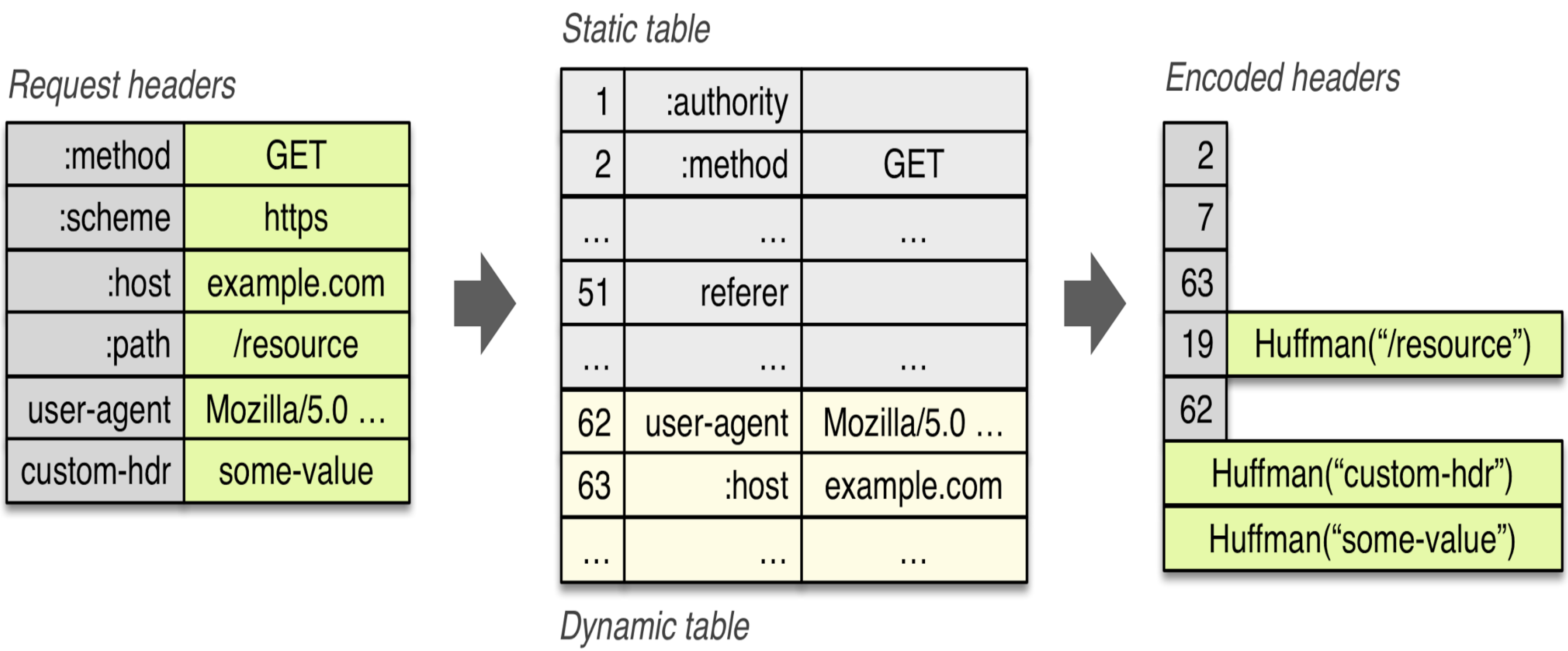
Basic data flow in HTTP/2



Streams are multiplexed because frames can be interleaved

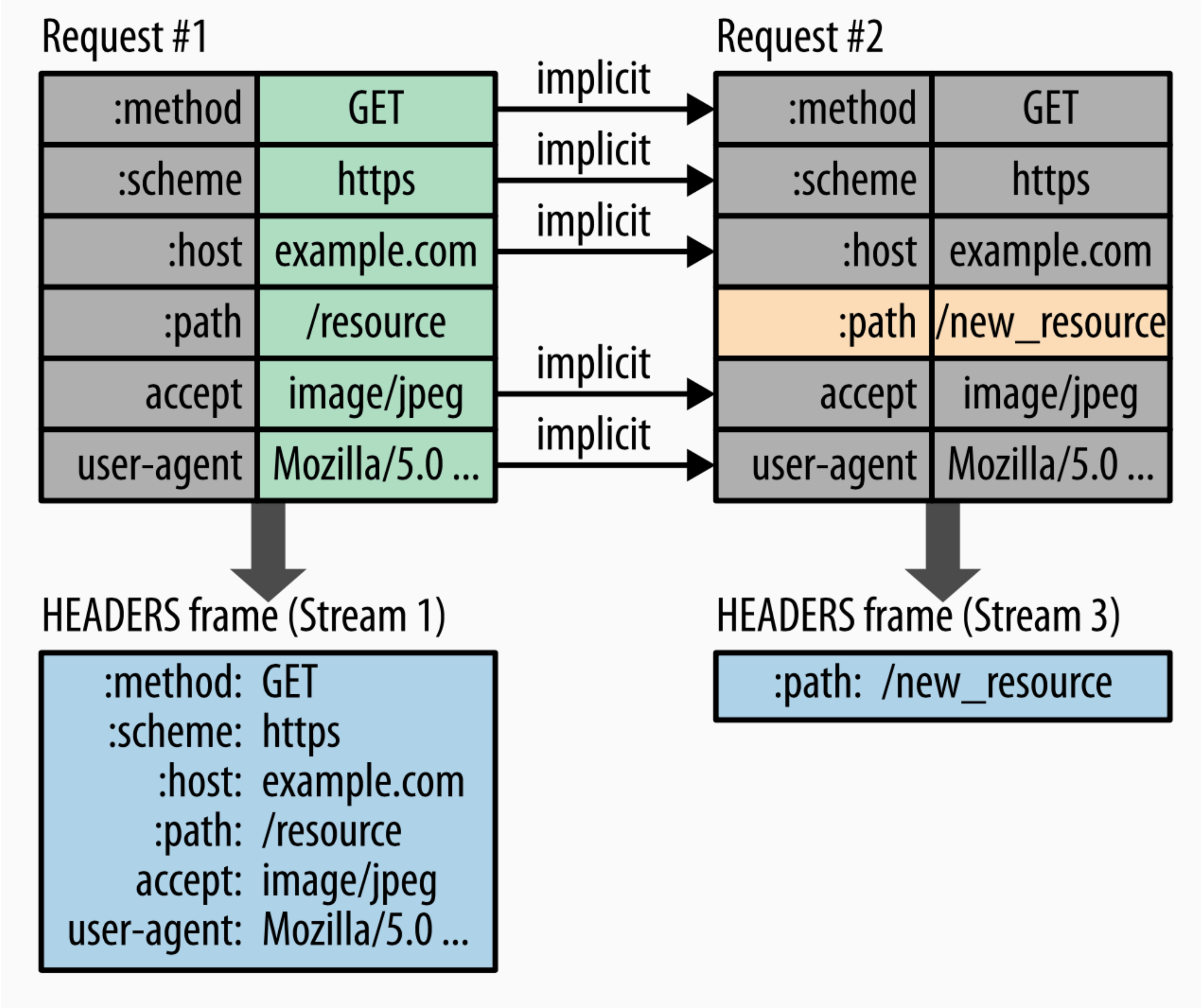
- All frames (e.g. HEADERS, DATA, etc) are sent over single TCP connection
- Frame delivery is prioritized based on stream dependencies and weights
- DATA frames are subject to per-stream and connection flow control

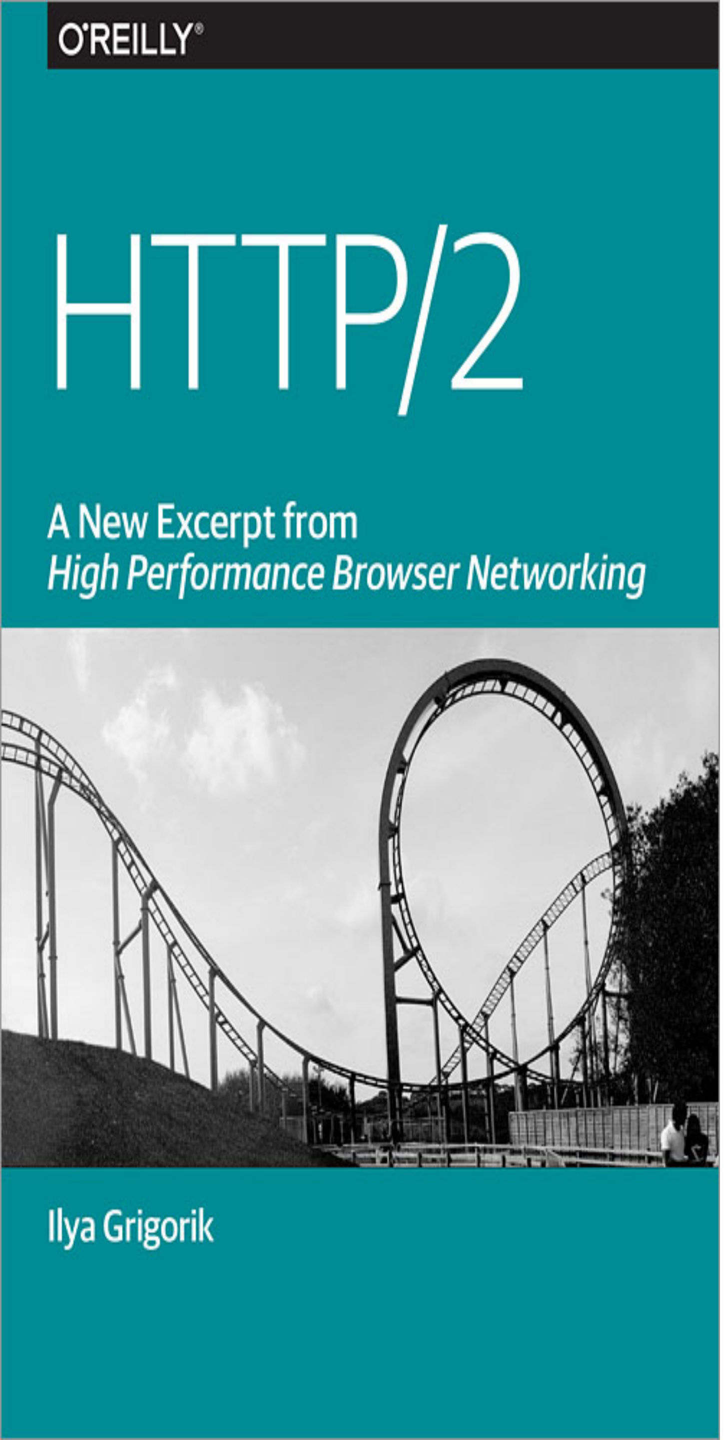
HPACK header compression



- *Literal values are (optionally) encoded with a static Huffman code*
- *Previously sent values are (optionally) indexed*
  - *e.g. "2" in above example expands to "method: GET"*

HPACK header compression (more)





For a deep(er) dive on HTTP/2 protocol, grab the free book at the O'Reilly booth, or…

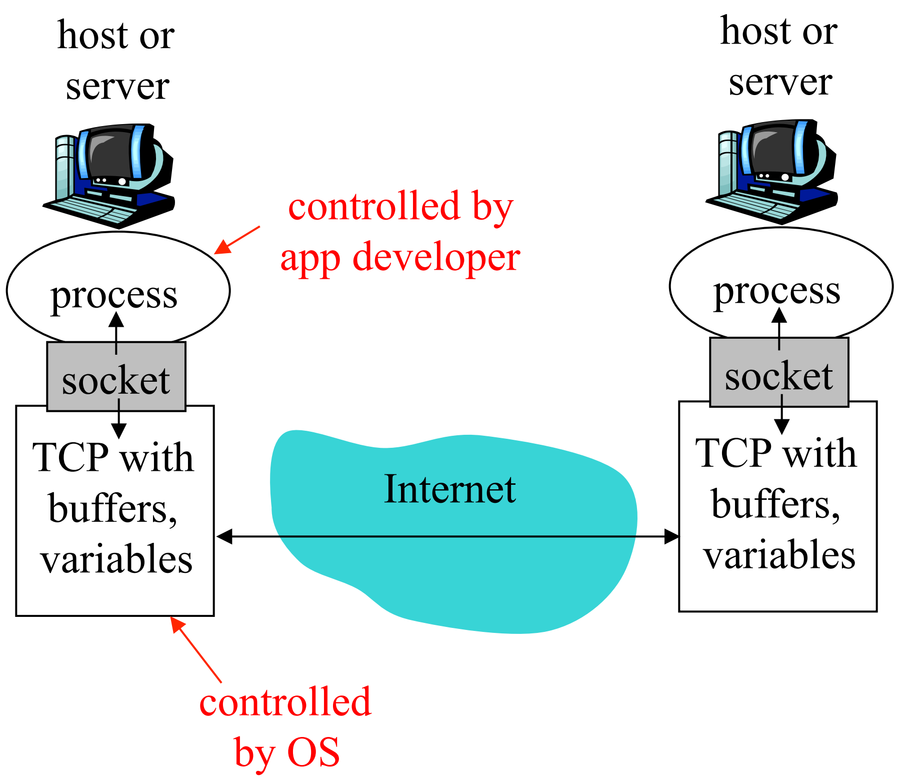
Read it online (free):
hpbn.co/http2

WebSocket

Recalling the original Socket

- process sends/receives messages to/from its socket
- socket analogous to door
  - sending process shoves message out door
  - sending process relies on transport infrastructure on other side of door which brings message to socket at receiving process
- Support both blocking and non-blocking calls

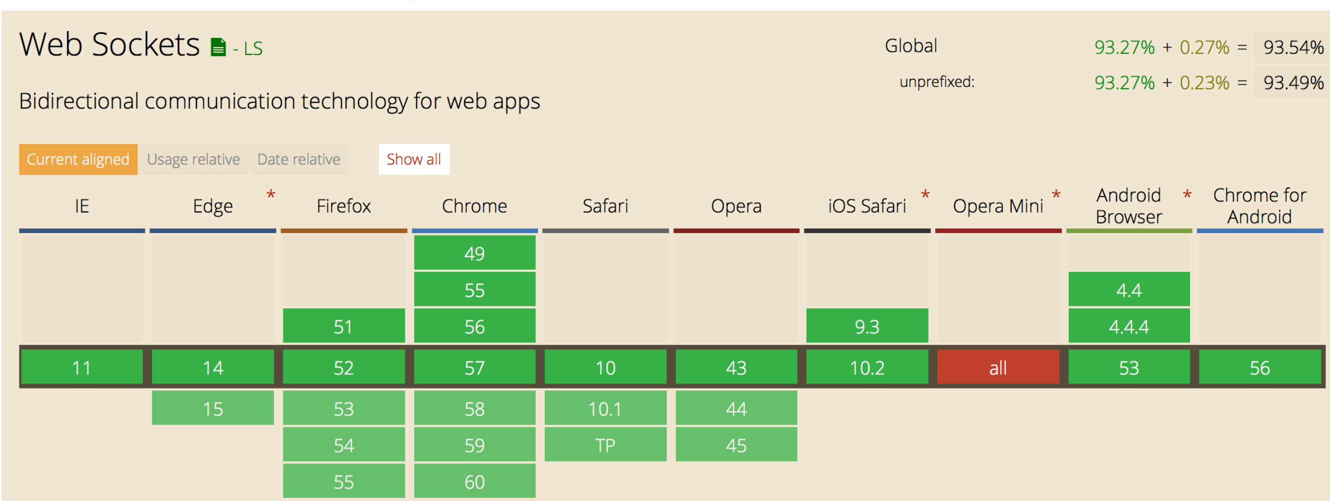
=> Support both synchronous and Asynchronous mode of operations



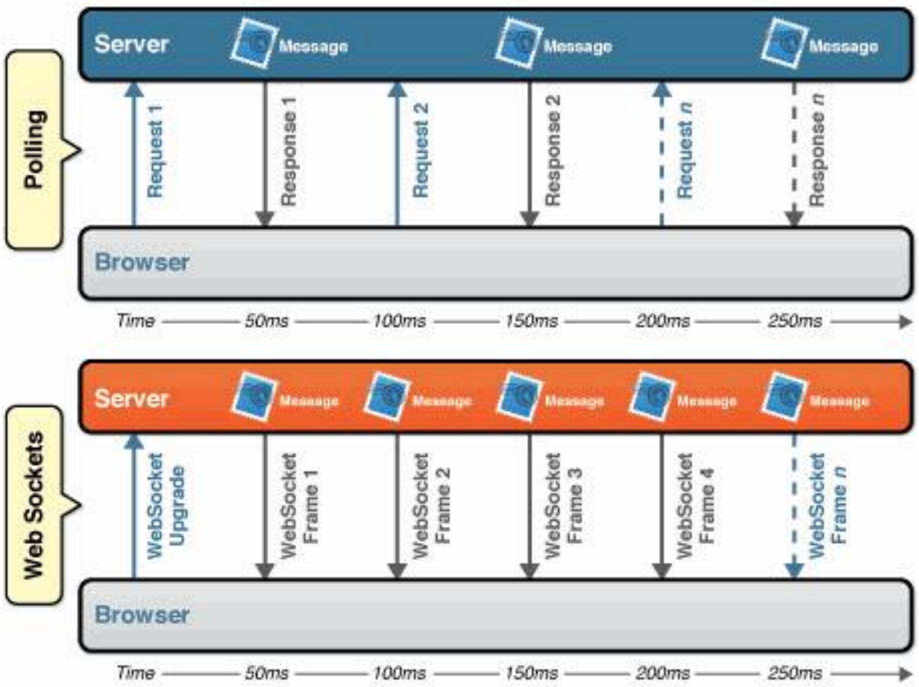
Can we provide similar abstraction of Network Service to a Web Application directly ?

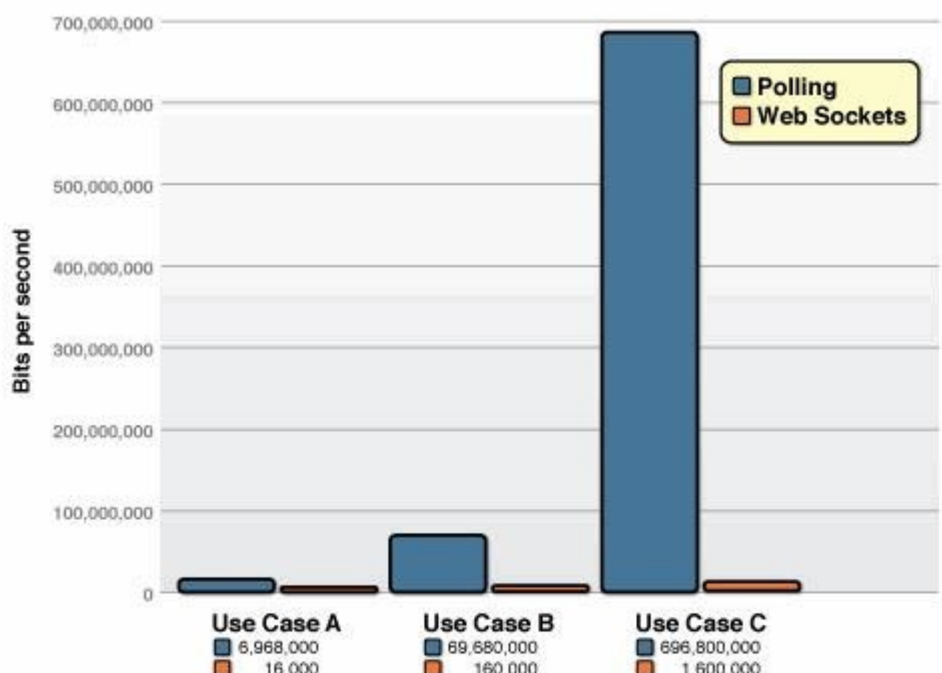
WebSocket (ws:// or wss://)

- Part of the original HTML5 effort to enhance REAL-TIME, asynchronous, bi-directional communications between the browser and the web-server
- Provide full-duplex communications channels over a single TCP connection by carrying sub-protocols, e.g. SOAP, XMPP, JSON-RPC
- Over-the-wire protocol standardized by the IETF as RFC 6455
- WebSocket APIs available for Javascripts & other programming languages
  - Some Server-side Implementations:
    - Node.js – Socket.IO, WebSocket.Node, ws
    - Java – jetty
    - Python – pywebsocket, Tornado
    - C++ - libwebsockets
    - .NET - SuperWebSocket
  - Browser-side Implementation:



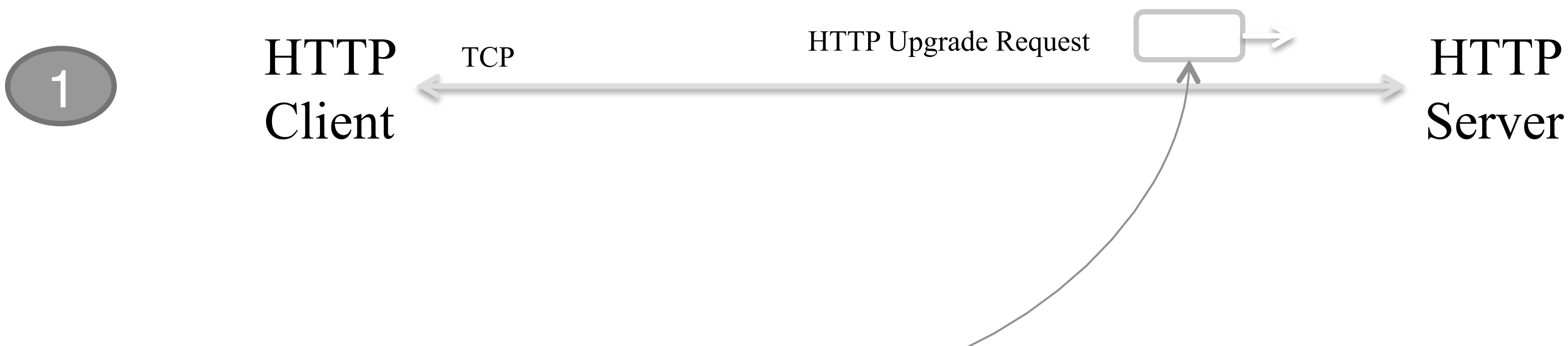
Overhead/Latency Comparison: AJAX long-polling vs. WebSocket





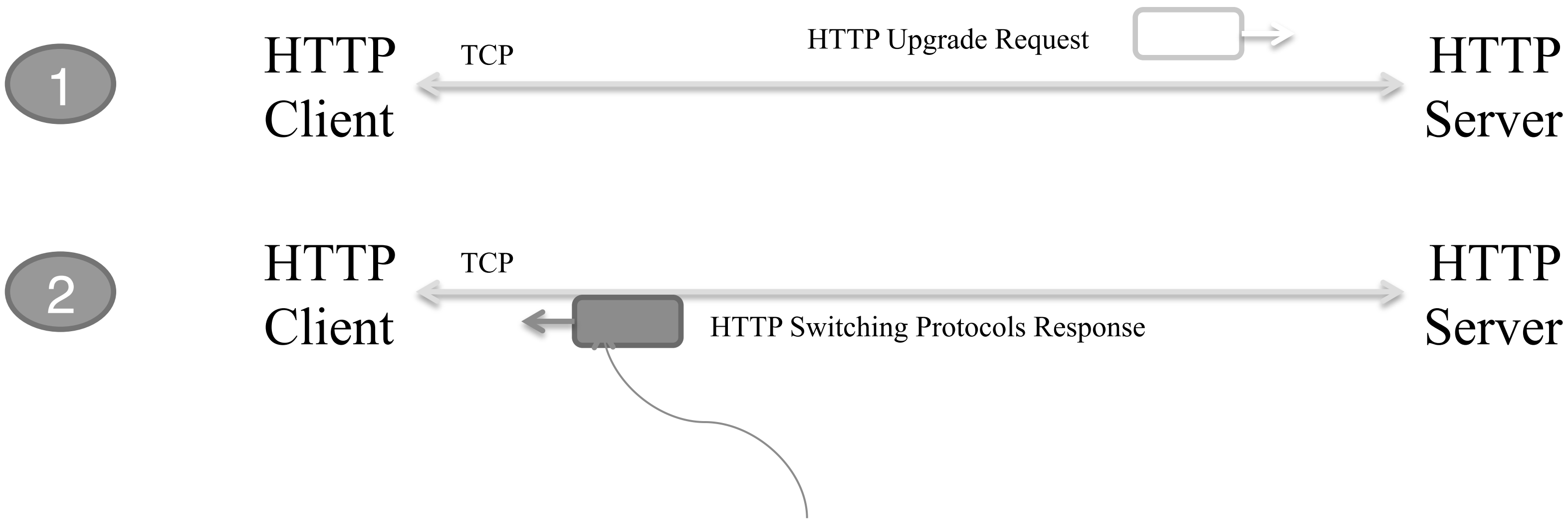
Source: http://www.codeproject.com/Articles/531698/Introduction-to-HTML5-WebSocket

WebSocket is triggered using the HTTP-Upgrade Mechanism during Opening handshake



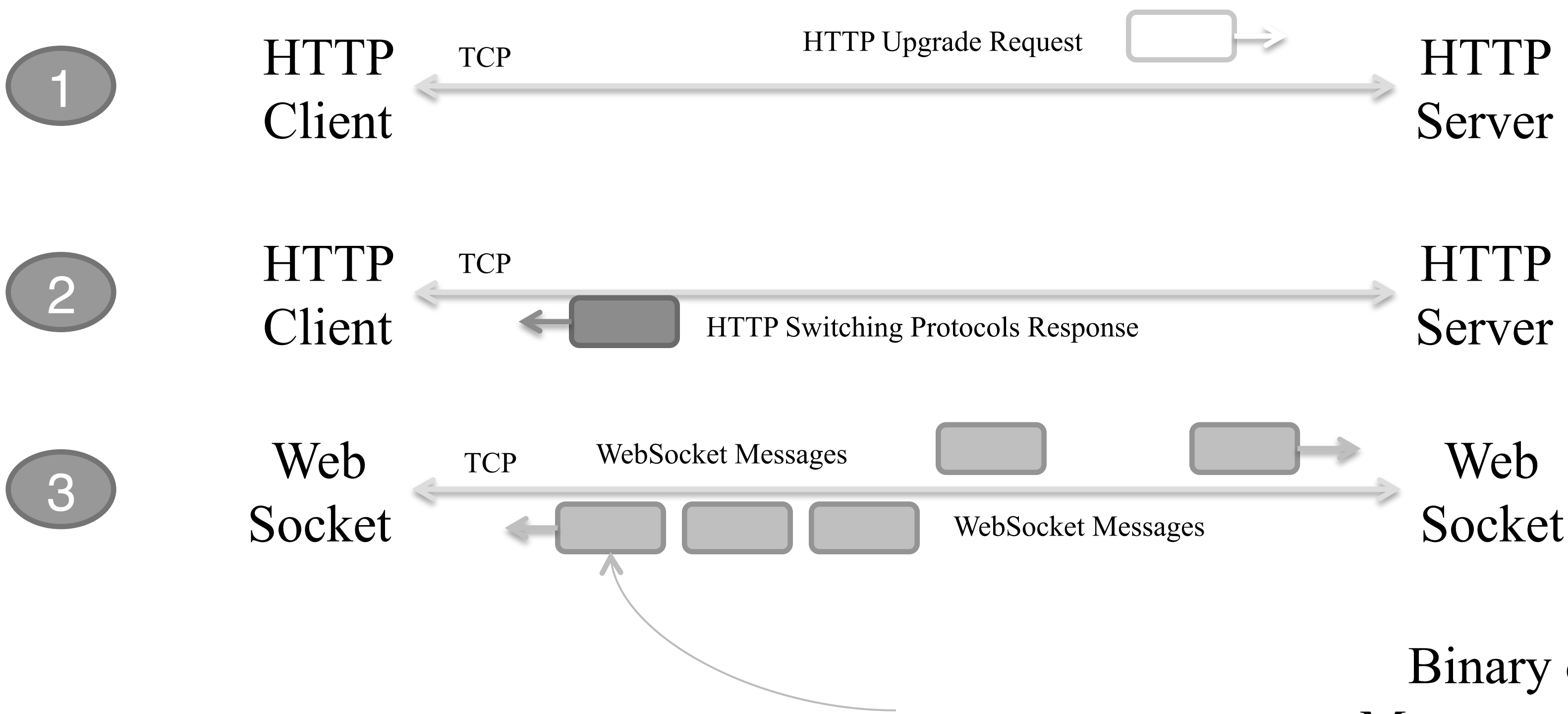

```
GET /chat HTTP/1.1
Host: example.com
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Key: dGhlIHNhbXBsZSBub25jZQ==
Sec-WebSocket-Origin: http://example.com
Sec-WebSocket-Version: 6
```

Opening handshake



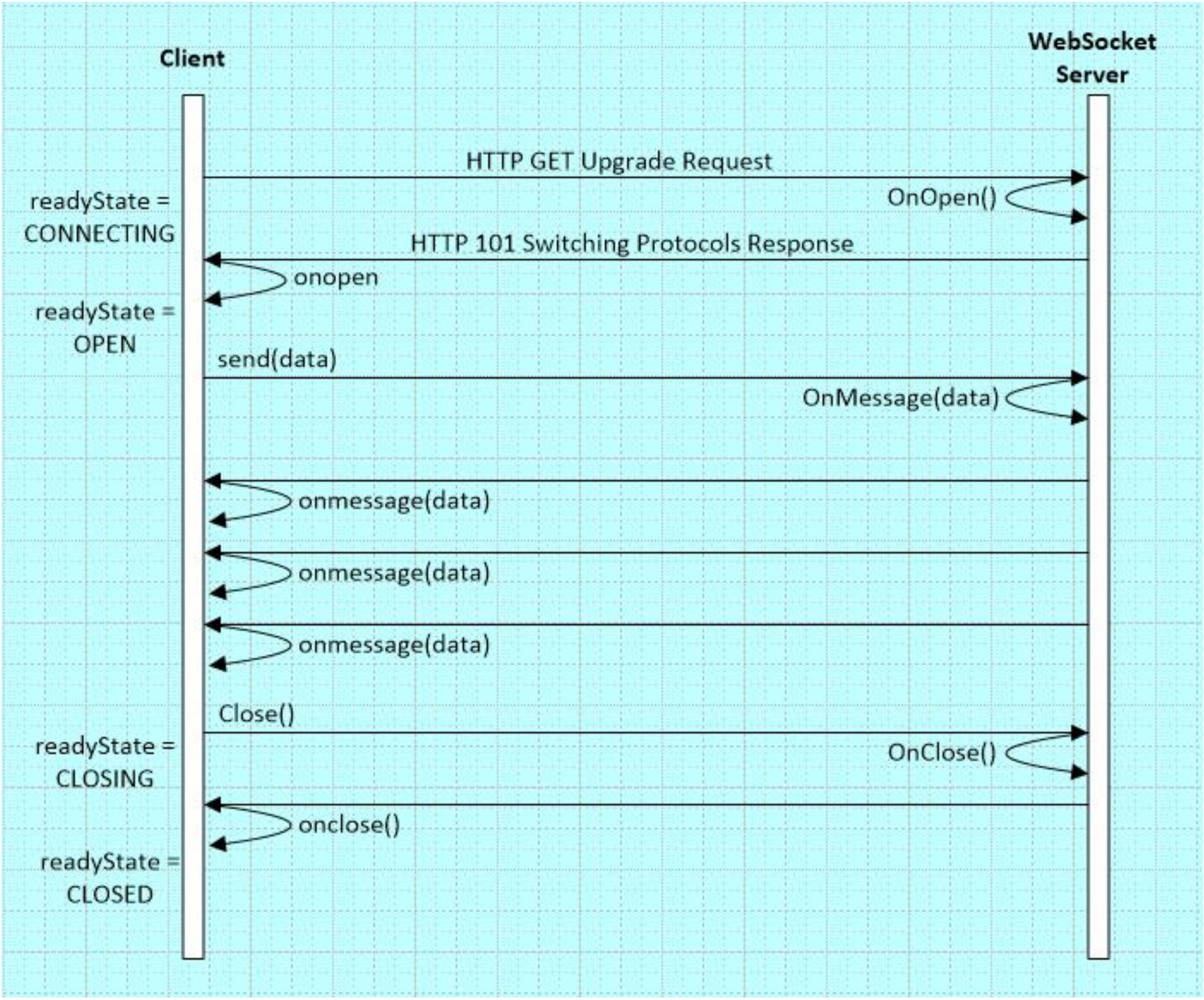

```
HTTP/1.1 101 Switching Protocols
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Accept: s3pPLMBiTxaQ9kYGzzhZRbK+xOo=
```

Opening handshake



Binary or UTF8
Messages or streams

WebSocket Client-Server Communication Pattern



Similarities b/w SPDY and WebSocket

- Support Asynchronous mode of communications
  - eliminates the overhead of “polling” generally used to simulate “real time” updates
- Use only a single TCP connection
  - reduces overhead on servers (and infrastructure) which can translate into better performance for the end-user.
- Make use of compression
  - reduces size of data transferred, better performance, particularly over more constrained mobile networks.

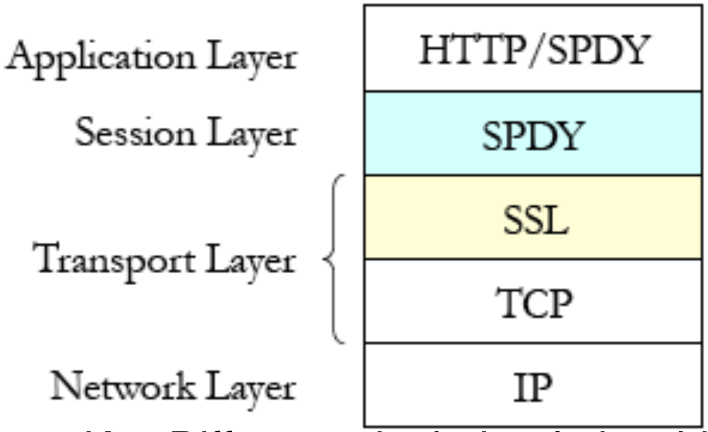
Data Framing in WebSocket

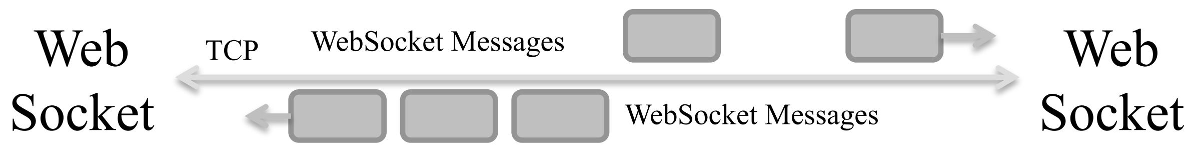
- Messages are segmented as frames.
- Why frames?
  - No need to wait until the whole message is completed
  - Multiplexing, better share the output channel

Origin-based security Model for WebSocket

- Verify the “Origin” field. If the origin indicated is unacceptable to the server, reject.
  - Recall: Same Origin Policy (SOP) in Javascript
- Restrict which web pages can contact a WebSocket server.
- Don’t work when the connection is initiated by **Non-Browser Clients**
- Assume trusted origin is always secure
  - May not be a good assumption
  - Actually, some early versions of WebSocket has been disabled by some browser by default due to Security concern !

Differences b/w SPDY and WebSocket





- Key Difference in their relationship with HTTP
  - SPDY: does not replace HTTP message/header ; HTTP simply nested within SPDY
  - WebSocket: almost independent, without HTTP header
    - Less overhead
    - Lack of HTTP header can blind the infrastructure. IDS, IPS, Load-balancer, Accelerator, Firewalls, anti-virus scanners – any service which relies upon HTTP headers to determine specific content type or location (URI) of the object being requested – is unable to inspect or validate requests due to its lack of HTTP headers.
- There is even a serious draft specification for running WebSocket over SPDY !
  - https://docs.google.com/document/d/1zUEFzz7NCls3Yms8hXxY4wGXJ3EEvoZc3GihrqPQcM0/edit
- When to use what (SPDY or WebSocket) ? Some advice from:

  https://blogs.akamai.com/2012/07/spdy-and-websocket-support-at-akamai.html

  https://www.infoq.com/news/2012/06/spdy-websockets

QUIC

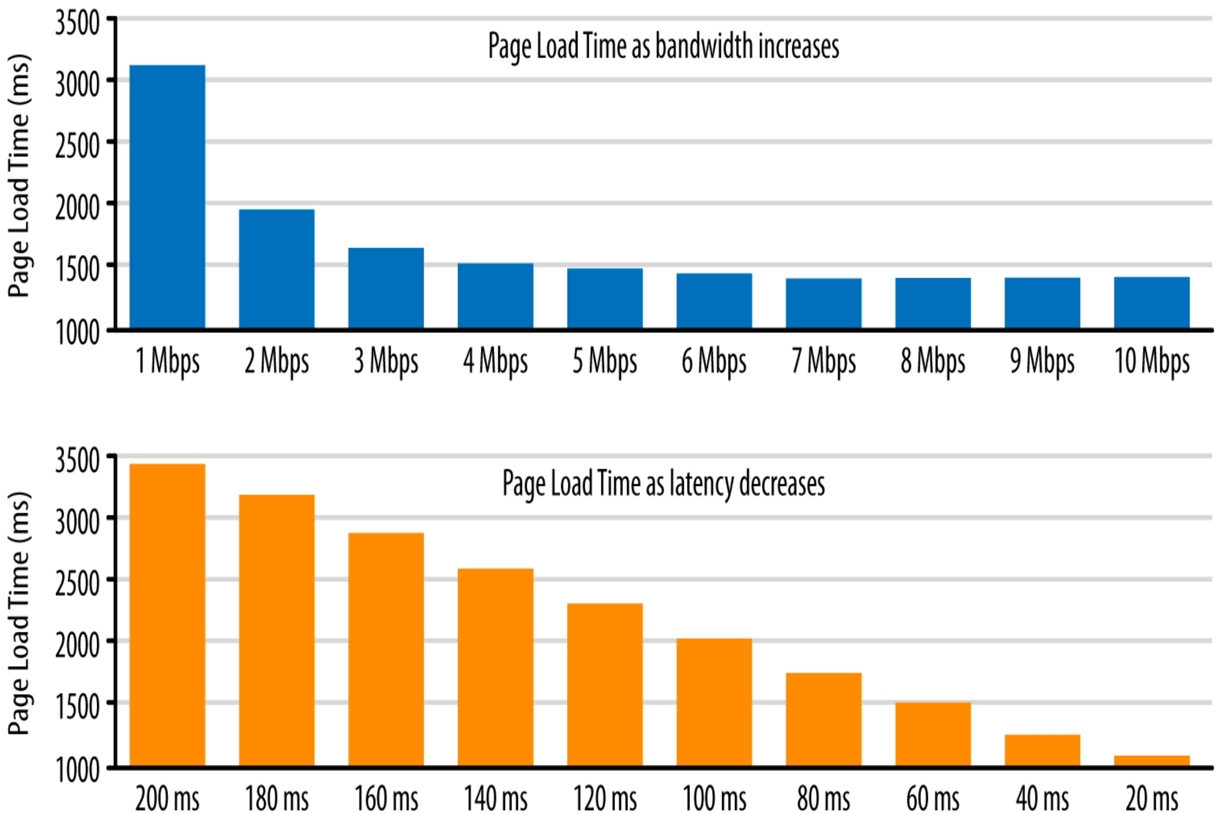
Redefining Internet Transport

QUIC

Red… Internet Transport

Reinventing?

Latency vs Bandwidth Impact on Page Load Time

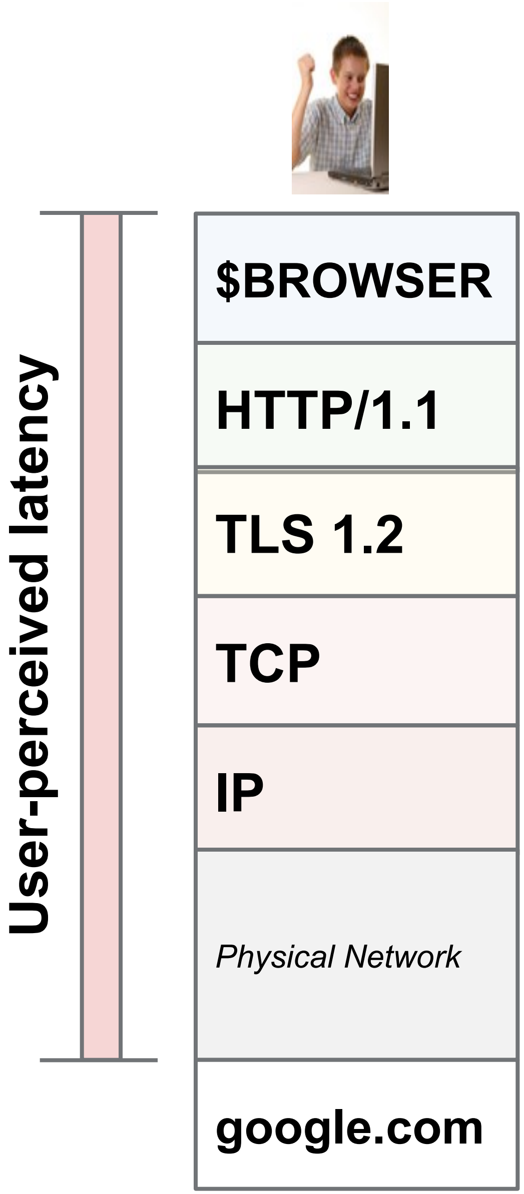


Single digit % perf improvement after 5 Mbps

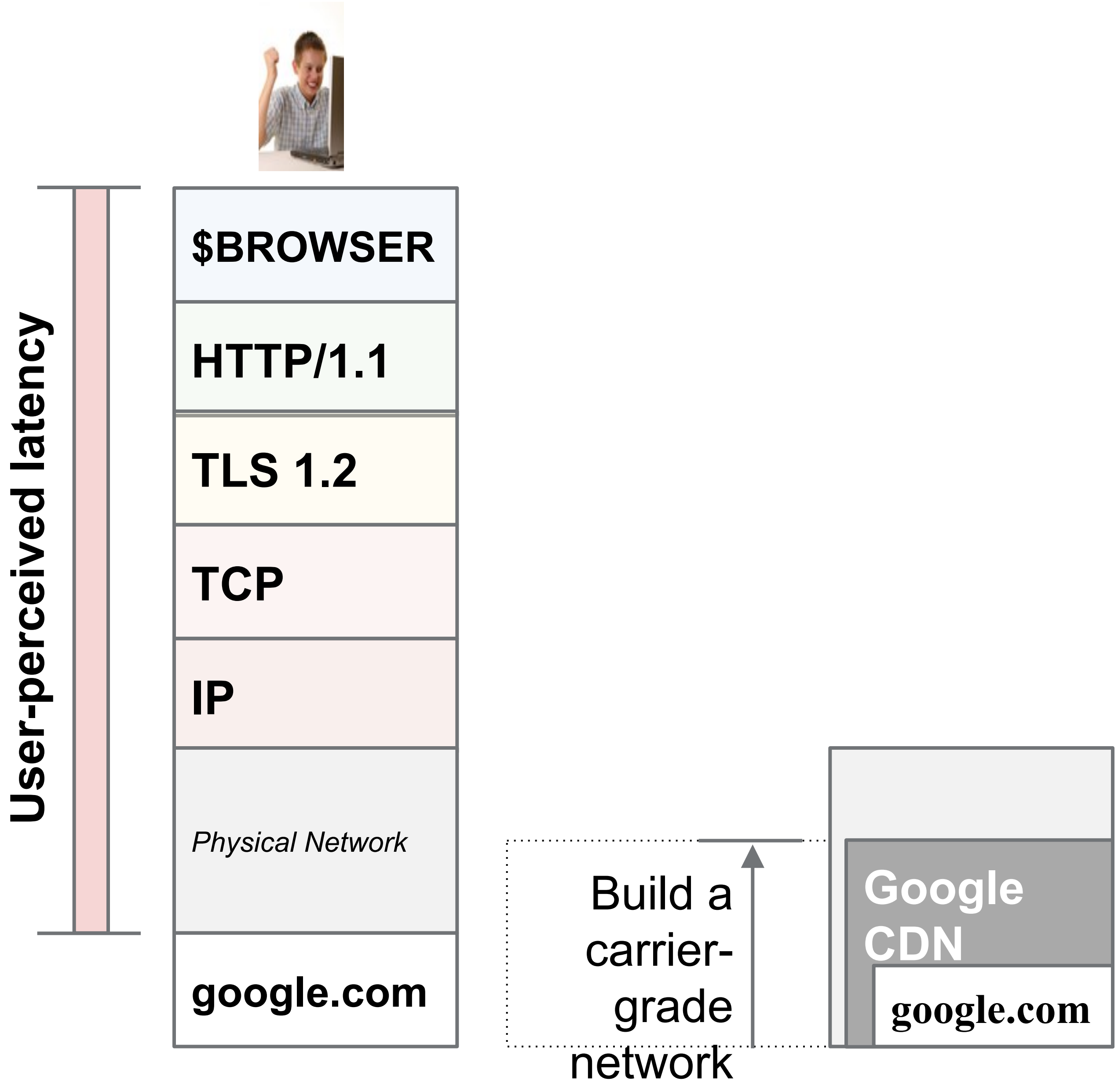
Linear improvement in page load time!

"**To speed up the Internet at large, we should look for more ways to bring down RTT.** What if we could reduce cross-atlantic RTTs from 150 ms to 100 ms? This would have a larger effect on the speed of the internet than increasing a user's bandwidth from 3.9 Mbps to 10 Mbps or even 1 Gbps." - Mike Belshe

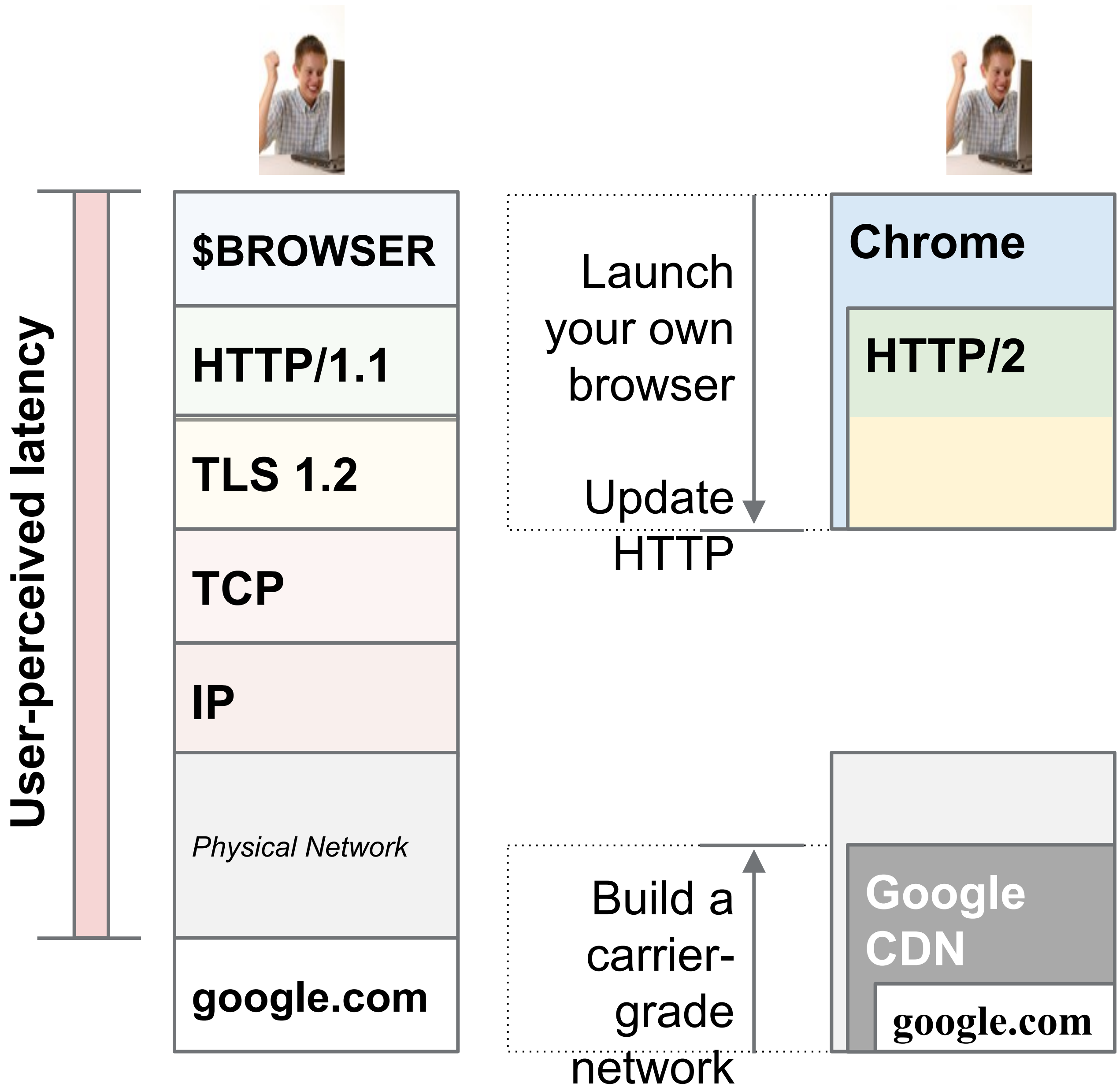
How do you make the web faster?



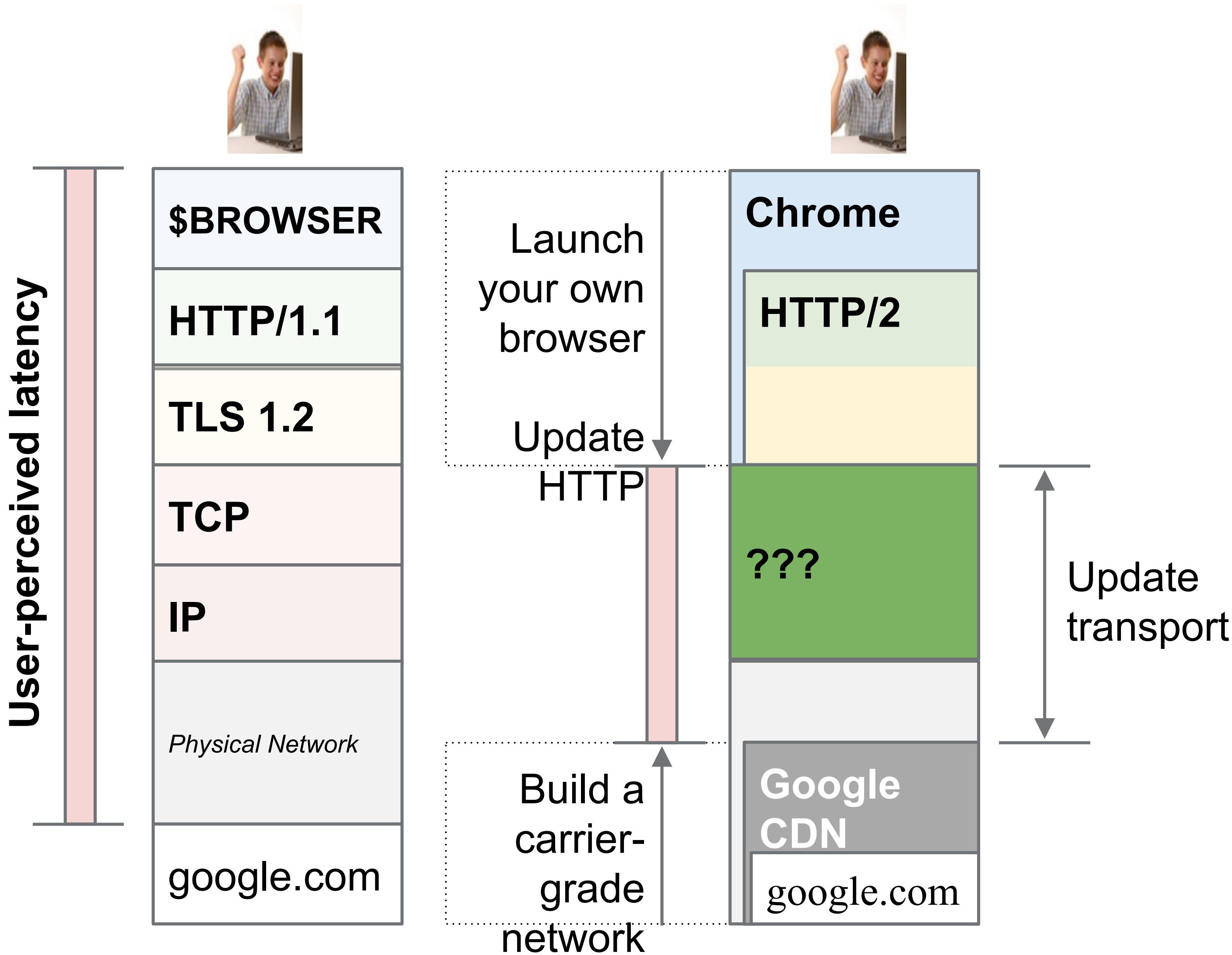
How do you make the web faster?



How do you make the web faster?



How do you make the web faster?



What is QUIC ?

Quick **U**DP **I**nternet **C**onnections

- A reliable, multiplexed transport over UDP

  Always encrypted

  Reduces latency

  Runs in user-space

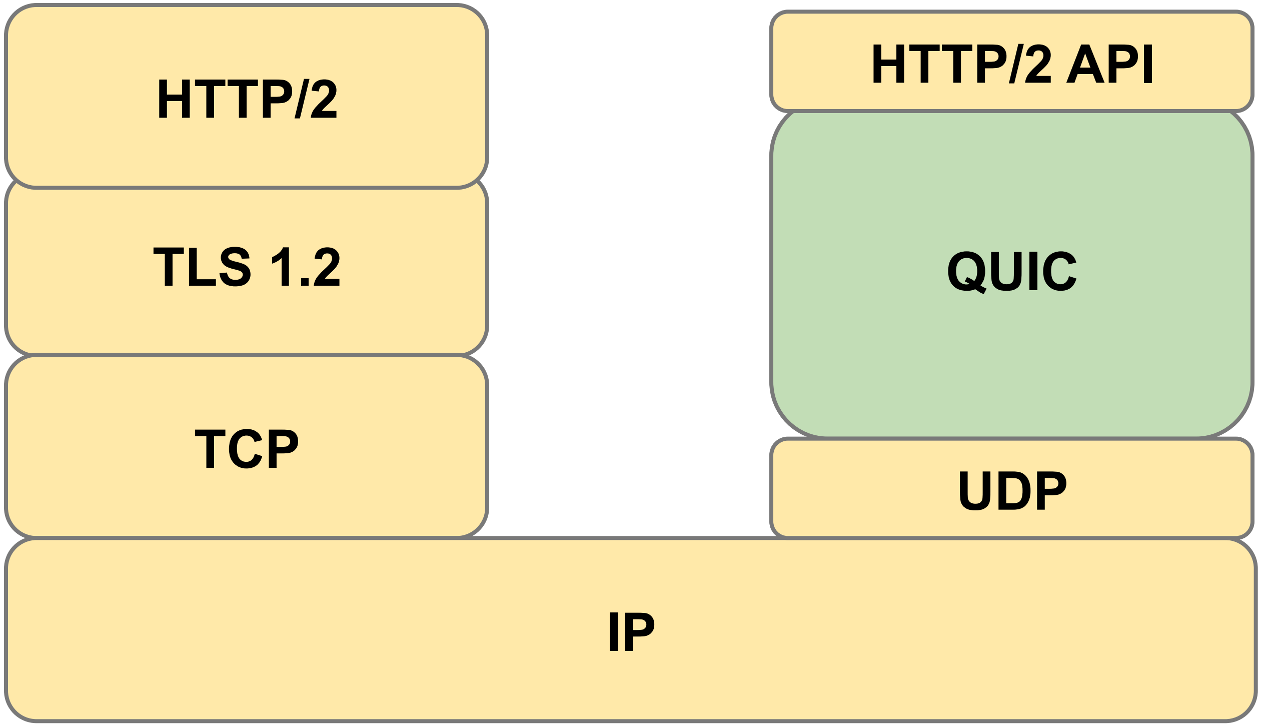
  Open sourced in Chromium

What is QUIC?

New transport designed to reduce web latency

- TCP + TLS + SPDY over UDP
- Faster connection establishment than TLS/TCP
  - 0-RTT usually, 1-RTT sometimes
- Deals better with packet loss than TCP
- Has Stream-level and Connection-level Flow Control
- FEC recovery
- Multipath

Where does QUIC fit?



Always encrypted

Comparable to TLS

Perfect forward secrecy, with more efficient handshake

IP spoofing protection

Signed proof of address

Inspired TLS 1.3's 0-RTT handshake

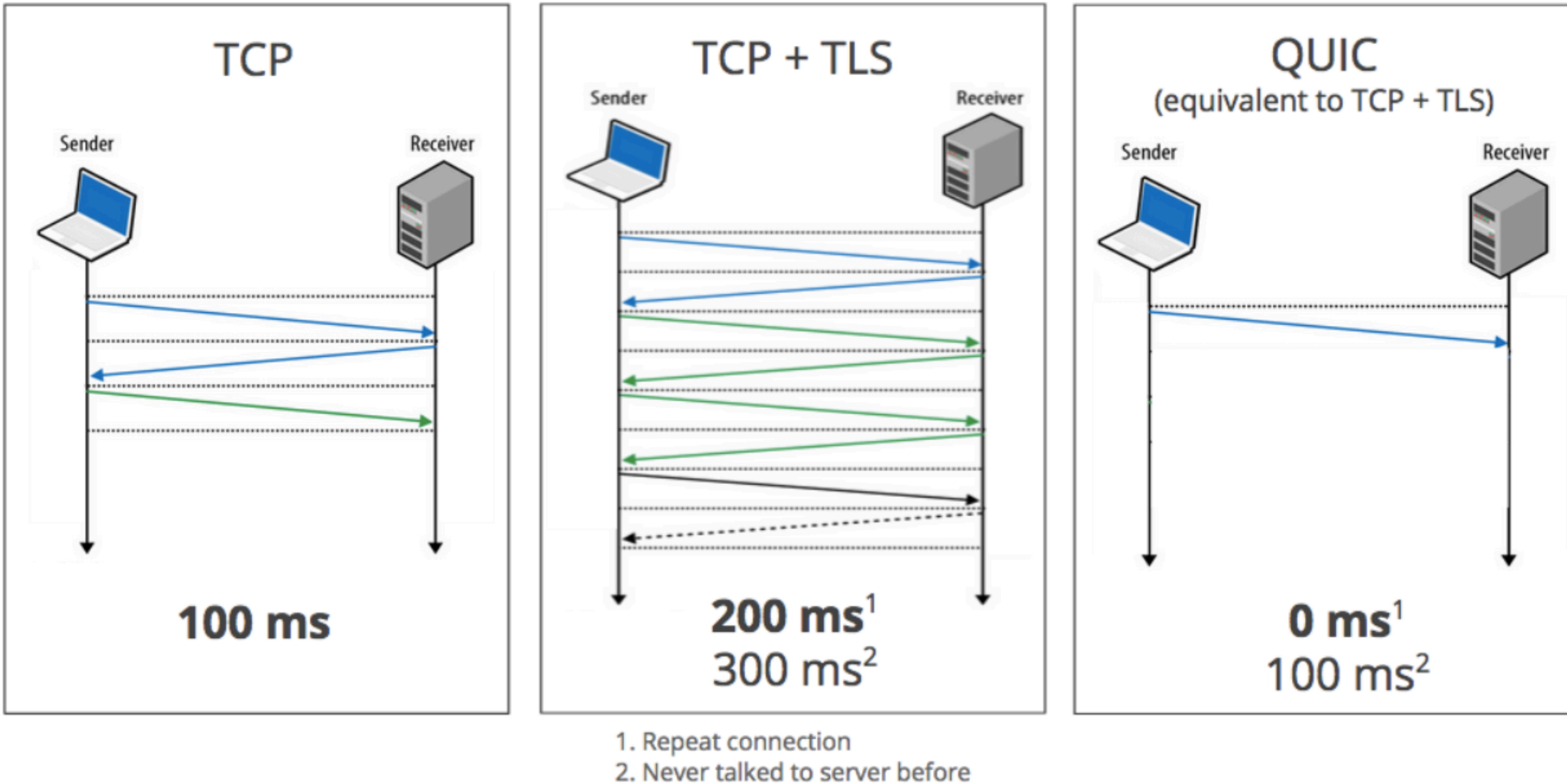
Plan to adopt TLS 1.3 when complete

Connection establishment

Connection identified by Connection ID

- As opposed to common 5-tuple
- 64 bits
- Chosen randomly by the client
- Enables connection mobility across IP, port

Zero RTT connection establishment



First-ever connection - 1 RTT

No cached information available

First CHLO is inchoate (empty)

- Simply includes version and server name

Server responds with REJ

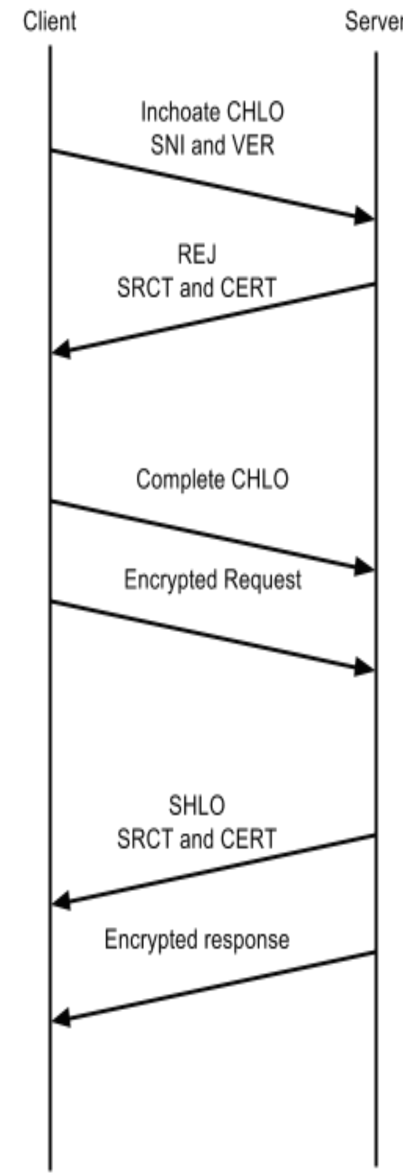
- Includes server config, certs, etc
- Allows client to make forward progress

Second CHLO is complete

- Followed by initially encrypted request data

Server responds with SHLO

- Followed immediately by forward-secure encrypted response data



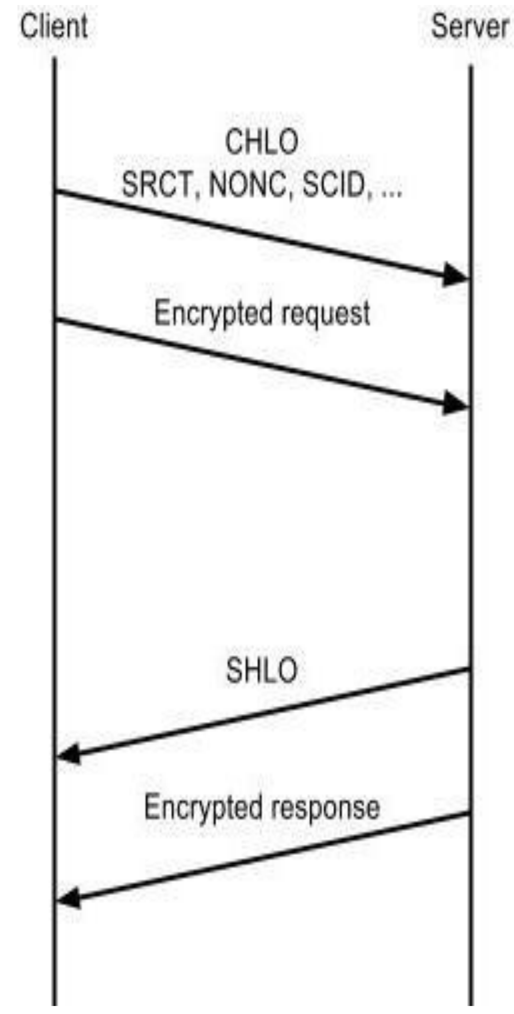
Subsequent connections - 0 RTT

First CHLO is complete

- Based on information from previous connection
- Followed by initially encrypted data.

Server responds with SHLO

- Followed immediately by forward-secure encrypted data



Congestion control & reliability

QUIC builds on decades of experience with TCP

Incorporates TCP best practices

TCP Cubic - fair with TCP

FACK, TLP, F-RTO, Early Retransmit...

More flexibility going forward

Improved congestion feedback, control over acking

Better signaling than TCP

Better signaling than TCP

Retransmitted packets consume new sequence number

- No retransmission ambiguity
- Prevents loss of retransmission from causing RTO

More verbose ACK

- TCP supports up to 3 SACK ranges
- QUIC supports up to 256 NACK ranges
- Per-packet receive times, even with delayed ACKs

ACK packets consume a sequence number

Measuring performance of QUIC

Controlled Experiments

Client Side
Latency, Bandwidth, Quality of Experience, Errors

Server Side
Latency, Bandwidth, QUIC Success Rate

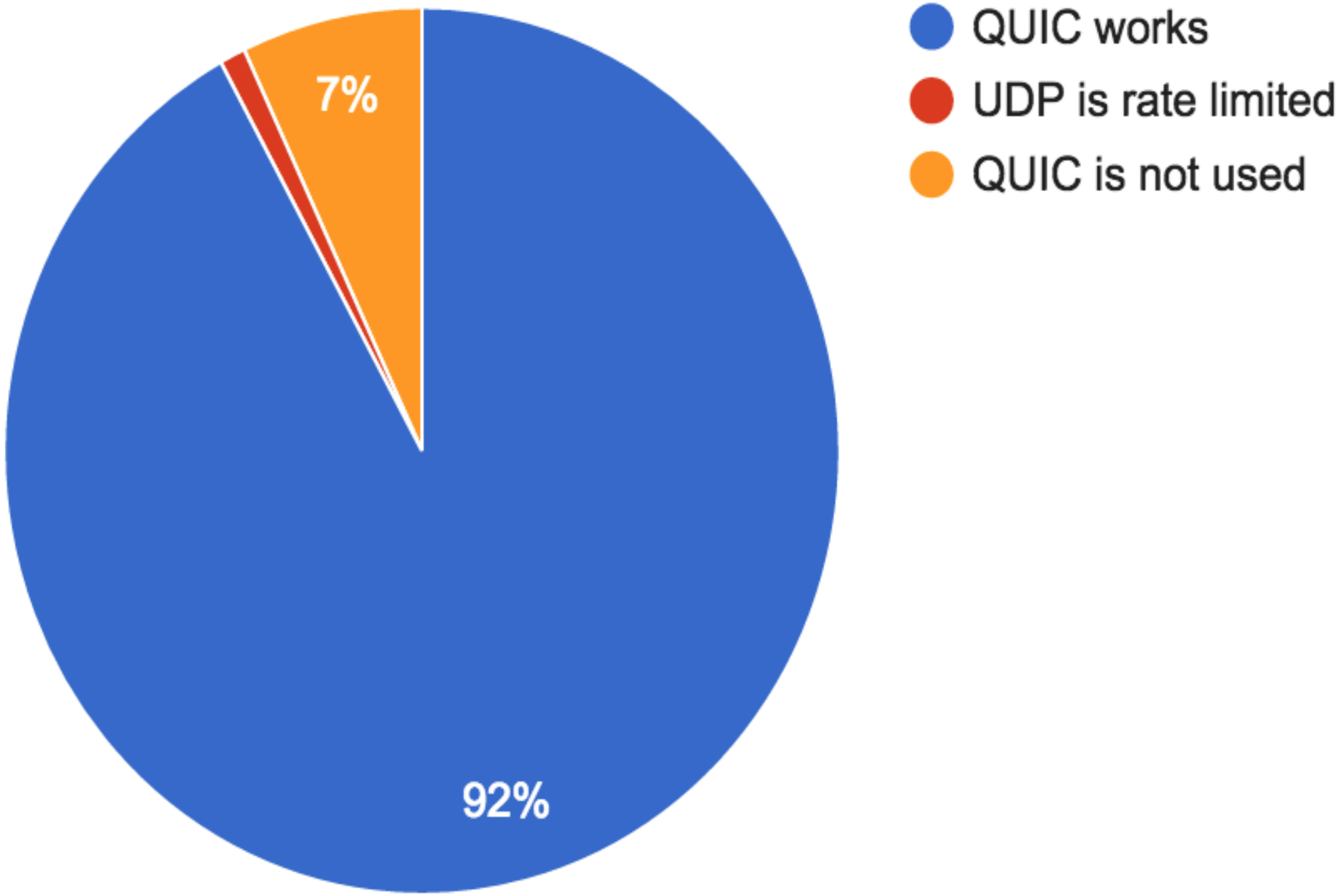
Fine Grained Analysis
By ASN, Server, OS, Version

Initial Deployment timeline of QUIC

Tested at scale, with millions of users

- Chrome Canary: June, 2013
- Chrome Stable: April, 2014
- Ramped up for Google traffic in 2015

Infrastructure Compatibility of QUIC



QUIC handshakes fail when RTTs are greater than 2.5 seconds
or
when UDP is blocked

Performance of QUIC on Google properties

Faster page loading times

- 5% faster on average
- 1 second faster for web search at 99th-percentile

Improved YouTube Quality of Experience

- 30% fewer rebuffers (video pauses)

Where are the gains from?

0-RTT

- Over 50% of the latency improvement (at median and 95th-percentile)

Improved loss recovery

- Over 10x fewer timeout based retransmissions improve tail latency and YouTube video rebuffer rates

Other, smaller benefits

- e.g. head of line blocking, more efficient framing

Client-side protection

What if UDP is blocked?

- Chrome seamlessly falls back to HTTP/TCP

What if the path MTU is too small?

- QUIC handshake fails, Chrome falls back to TCP

What if a client doesn’t want to use QUIC?

- Chrome flag / administrative policy to disable QUIC

When client-side protection is not enough...

As a last resort, Google disables QUIC to specific ASNs

- This is used as a fallback to protocol features

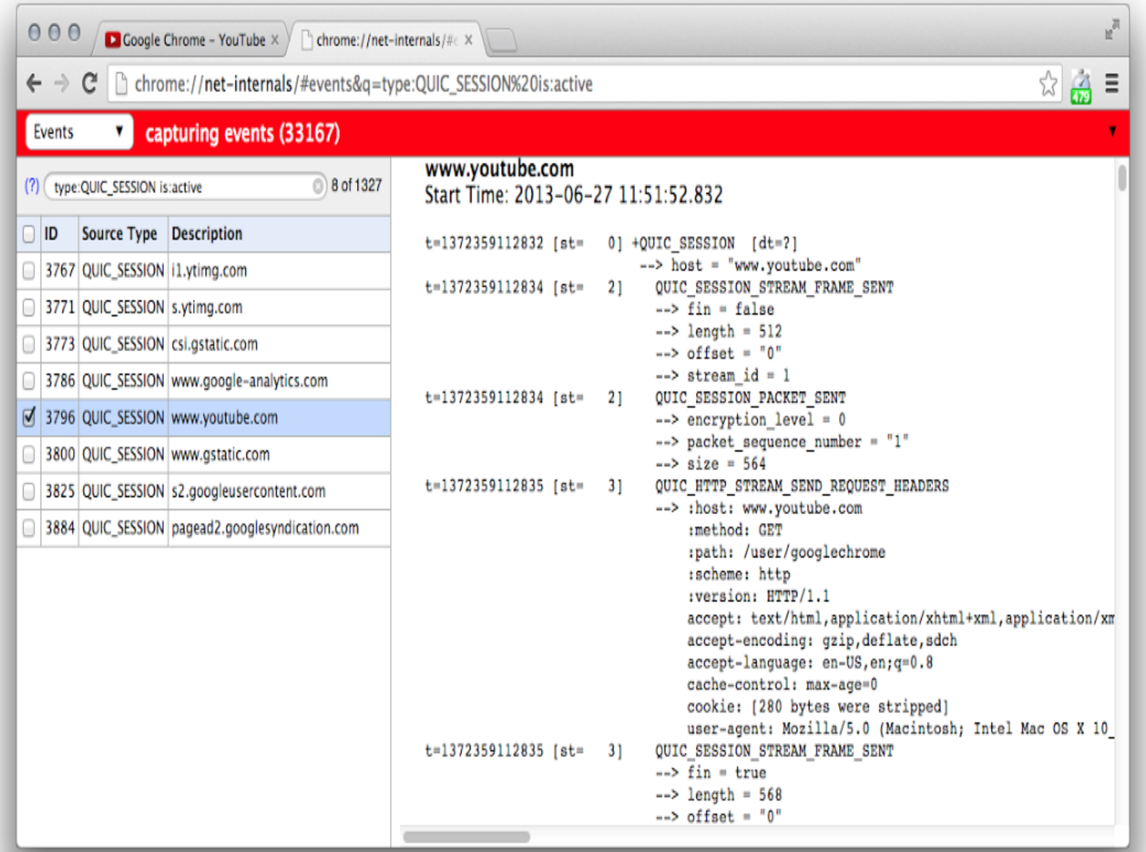
Why do we disable QUIC delivery?

- Degraded quality of experience measured
- Indications of UDP rate limiting at peak times of day
- End user reports (via chromium.org)

Debugging Tools: Chrome

chrome://net-internals

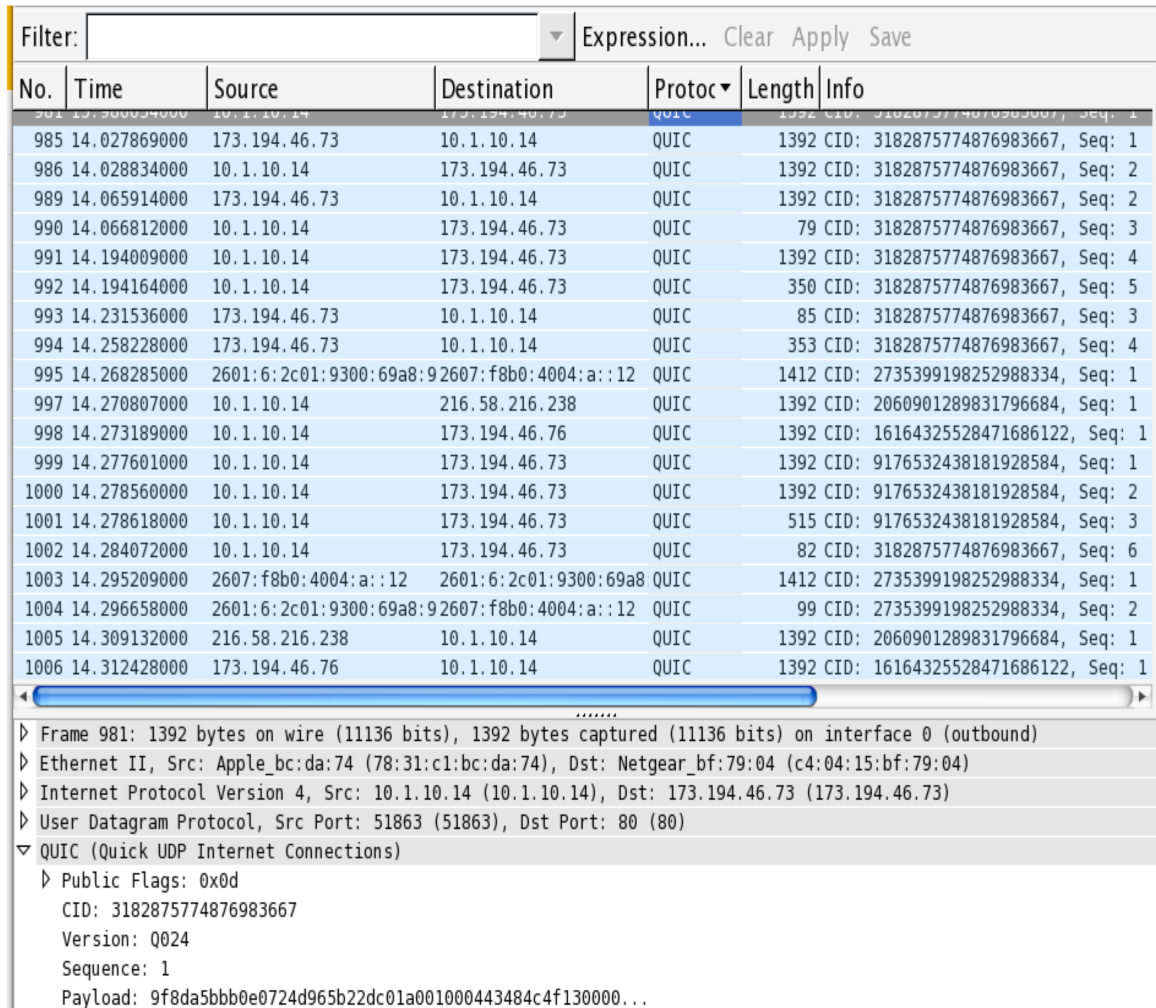
- Active QUIC sessions
- Captures all events
- Important for filing Chromium bugs



Debugging Tools: Wireshark

Parses

- Protocol: QUIC
- CID: Connection ID
- Seq: Sequence number
- Version: ie: Q024
- Public flags: 1 byte
- Payload: Encrypted



Filter: Expression... Clear Apply Save

No.	Time	Source	Destination	Protocol	Length	Info
981	[illegible]	10.1.10.14	173.194.46.73	QUIC	1392	CID: 3182875774876983667, Seq: 1
985	14.027869000	173.194.46.73	10.1.10.14	QUIC	1392	CID: 3182875774876983667, Seq: 1
986	14.028834000	10.1.10.14	173.194.46.73	QUIC	1392	CID: 3182875774876983667, Seq: 2
989	14.065914000	173.194.46.73	10.1.10.14	QUIC	1392	CID: 3182875774876983667, Seq: 2
990	14.066812000	10.1.10.14	173.194.46.73	QUIC	79	CID: 3182875774876983667, Seq: 3
991	14.194009000	10.1.10.14	173.194.46.73	QUIC	1392	CID: 3182875774876983667, Seq: 4
992	14.194164000	10.1.10.14	173.194.46.73	QUIC	350	CID: 3182875774876983667, Seq: 5
993	14.231536000	173.194.46.73	10.1.10.14	QUIC	85	CID: 3182875774876983667, Seq: 3
994	14.258228000	173.194.46.73	10.1.10.14	QUIC	353	CID: 3182875774876983667, Seq: 4
995	14.268285000	2601:6:2c01:9300:69a8:9	2607:f8b0:4004:a::12	QUIC	1412	CID: 2735399198252988334, Seq: 1
997	14.270807000	10.1.10.14	216.58.216.238	QUIC	1392	CID: 2060901289831796684, Seq: 1
998	14.273189000	10.1.10.14	173.194.46.76	QUIC	1392	CID: 16164325528471686122, Seq: 1
999	14.277601000	10.1.10.14	173.194.46.73	QUIC	1392	CID: 9176532438181928584, Seq: 1
1000	14.278560000	10.1.10.14	173.194.46.73	QUIC	1392	CID: 9176532438181928584, Seq: 2
1001	14.278618000	10.1.10.14	173.194.46.73	QUIC	515	CID: 9176532438181928584, Seq: 3
1002	14.284072000	10.1.10.14	173.194.46.73	QUIC	82	CID: 3182875774876983667, Seq: 6
1003	14.295209000	2607:f8b0:4004:a::12	2601:6:2c01:9300:69a8	QUIC	1412	CID: 2735399198252988334, Seq: 1
1004	14.296658000	2601:6:2c01:9300:69a8:9	2607:f8b0:4004:a::12	QUIC	99	CID: 2735399198252988334, Seq: 2
1005	14.309132000	216.58.216.238	10.1.10.14	QUIC	1392	CID: 2060901289831796684, Seq: 1
1006	14.312428000	173.194.46.76	10.1.10.14	QUIC	1392	CID: 16164325528471686122, Seq: 1

Frame 981: 1392 bytes on wire (11136 bits), 1392 bytes captured (11136 bits) on interface 0 (outbound)
Ethernet II, Src: Apple_bc:da:74 (78:31:c1:bc:da:74), Dst: Netgear_bf:79:04 (c4:04:15:bf:79:04)
Internet Protocol Version 4, Src: 10.1.10.14 (10.1.10.14), Dst: 173.194.46.73 (173.194.46.73)
User Datagram Protocol, Src Port: 51863 (51863), Dst Port: 80 (80)
QUIC (Quick UDP Internet Connections)
Public Flags: 0x0d
CID: 3182875774876983667
Version: Q024
Sequence: 1
Payload: 9f8da5bbb0e0724d965b22dc01a001000443484c4f130000...

Future Improvements

- Forward Error Correction
- Connection Mobility
- Multipath
- More congestion control experiments

Open source implementations

Servers

- Open source test server included in Chromium
- Working with other server vendors

Clients

- Open source Chromium client library for desktop and mobile
- Google Chrome and some Google Android apps
- Working with other browsers

QUIC at the IETF

Nov 2013 Initially Presented
Mar 2015 QUIC Crypto
July 2015 BarBoF

FEB 2017

- Formation of QUIC Working Group for Standard Track work based on previous QUIC drafts, their implementation and deployment experience !!
  https://datatracker.ietf.org/wg/quic/charter/

- Generalize the design described in previous IETF drafts:
  draft-hamilton-quic-transport-protocol
  draft-iyengar-quic-loss-recovery
  draft-shade-quic-http2-mapping
  draft-thomson-quic-tls

IETF QUIC WG Milestones

Milestones

Date	Milestone
Feb 2017	Working group adoption of QUIC Applicability and Manageability Statement
Feb 2017	Working group adoption of HTTP/2 mapping document
Feb 2017	Working group adoption of TLS 1.3 mapping document
Feb 2017	Working group adoption of Loss detection and Congestion Control document
Feb 2017	Working group adoption of Core Protocol document
Mar 2018	TLS 1.3 Mapping document to IESG
Mar 2018	Loss detection and Congestion Control document to IESG
Mar 2018	Core Protocol document to IESG
May 2019	Multipath extension document to IESG
Nov 2017	Working group adoption of Multipath extension document
Nov 2018	QUIC Applicability and Manageability Statement to IESG
Nov 2018	HTTP/2 mapping document to IESG

Summary of QUIC

- Reliable, multiplexed transport
- Always encrypted
- Run over UDP
- Lower Latency Connection Establishment
- Optional FEC
- Rapidly Evolving User-Space Implementation
- Open Source

Additional QUIC resources

Design Document of Specification Rationale for QUIC:

Jim Roskind, “QUIC Quick UDP Internet Connections – Multiplexed Stream Transport over UDP,” Dec 2013.

https://docs.google.com/document/d/1RNHkx_VvKWyWg6Lr8SZ-saqsQx7rFV-ev2jRFUoVD34/edit

Source: QUIC in Chromium

Page: www.chromium.org/quic

Public Mailing lists: quic@ietf.org
proto-quic@chromium.org (old)

IETF WG:
https://datatracker.ietf.org/wg/quic/documents/

Towards the Real-Time Web !

- Some people referred to the following as the Enabling Technologies for the "Real-Time Web" !! http://www.infoq.com/presentations/Real-time-Web-WebSocket-SPDY :
  - HTML5,
  - WebSocket,
  - SPDY => HTTP/2,
  - QUIC and …
  - WebRTC (Web Real-Time Communications) (www.webrtc.org)
  - W3C WebRTC WG (API) http://www.w3.org/2011/04/webrtc-charter.html
  - IETF RTCweb WG http://datatracker.ietf.org/wg/rtcweb/charter/

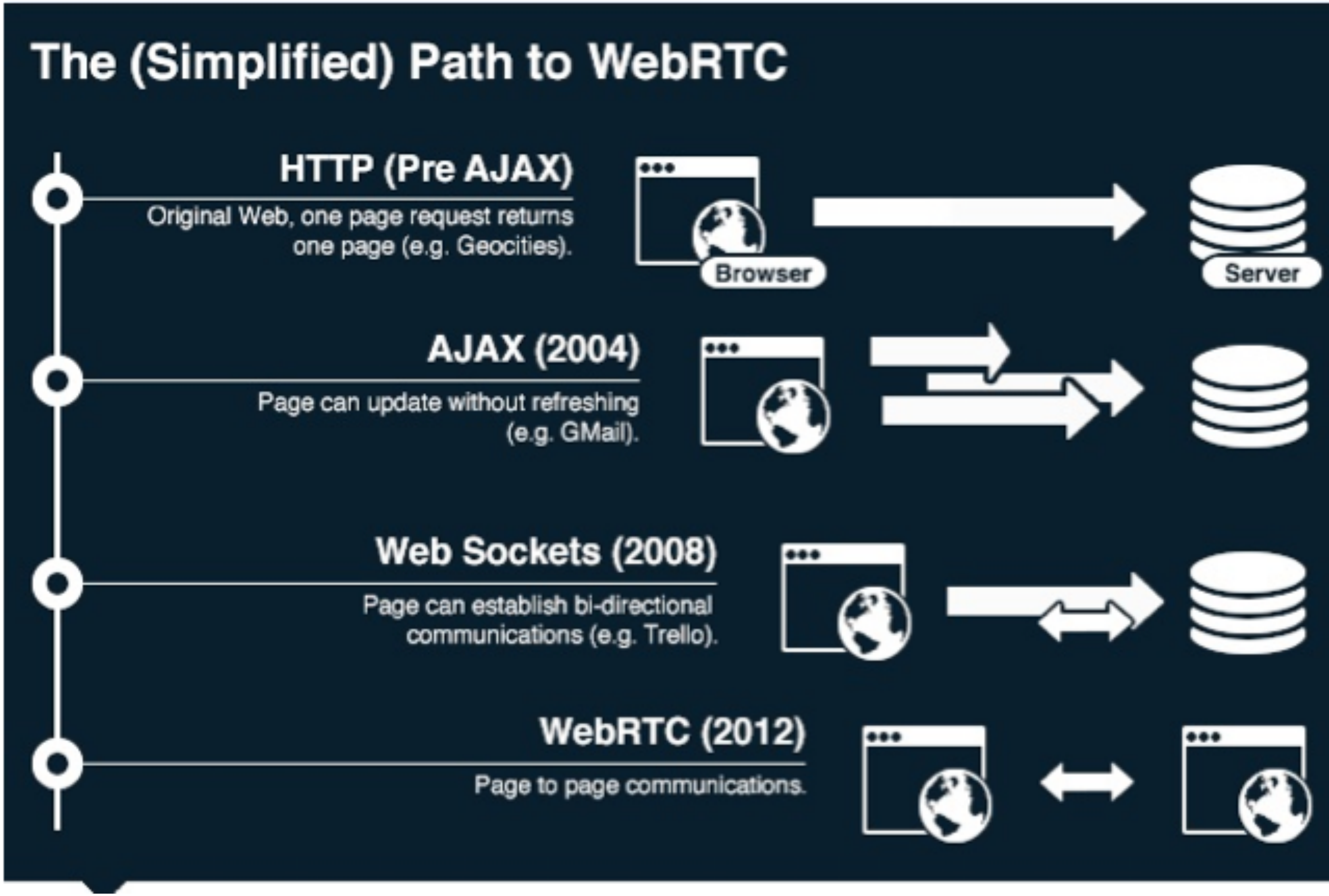
  "These two specifications aim to provide an environment where Javascript embedded in any page, viewed in any compatible browser, when suitably authorized by its user, is able to set up communication using audio, video and auxiliary data, where the browser environment does not constrain the types of application in which this functionality can be used." – from IETF Draft: draft-ietf-rtcweb-overview-18, Mar 3, 2017
- See the link below for a demo on how to implement

  a Real-Time Video Conference App using HTML5 with your Browser ONLY !
- http://html5videoguide.net/presentations/WebDirCode2012

Towards the Real Time Web

WebRTC

The Evolution Path from Web-Surfing to WebRTC

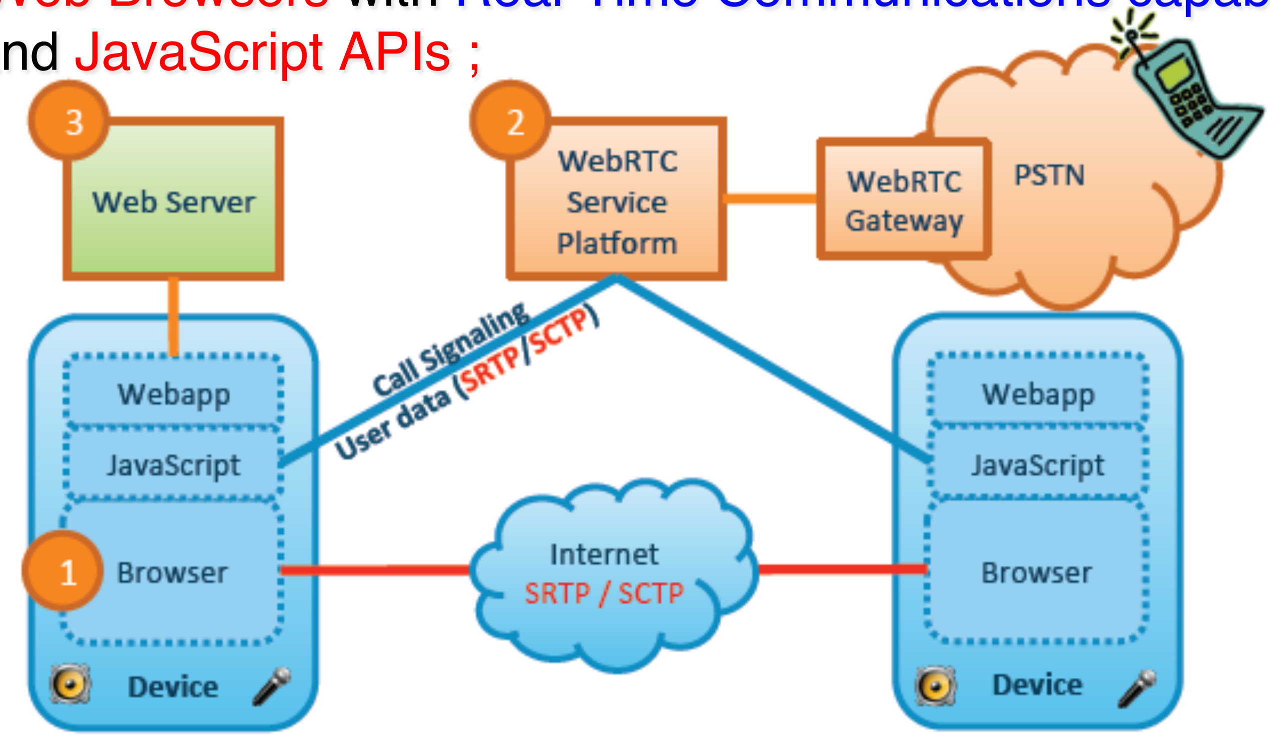


Source: Jimmy Lee / jimmylee.info

http://venturebeat.com/2012/08/13/webrtc-is-almost-here-and-it-will-change-the-web/

What is WebRTC ?

- A Google-driven W3C standardization effort (w/ support from IETF) which enables Web Browsers with Real-Time Communications capabilities via HTML5 and JavaScript APIs ;



No Need for Plugins anymore !

Key Components of WebRTC include:

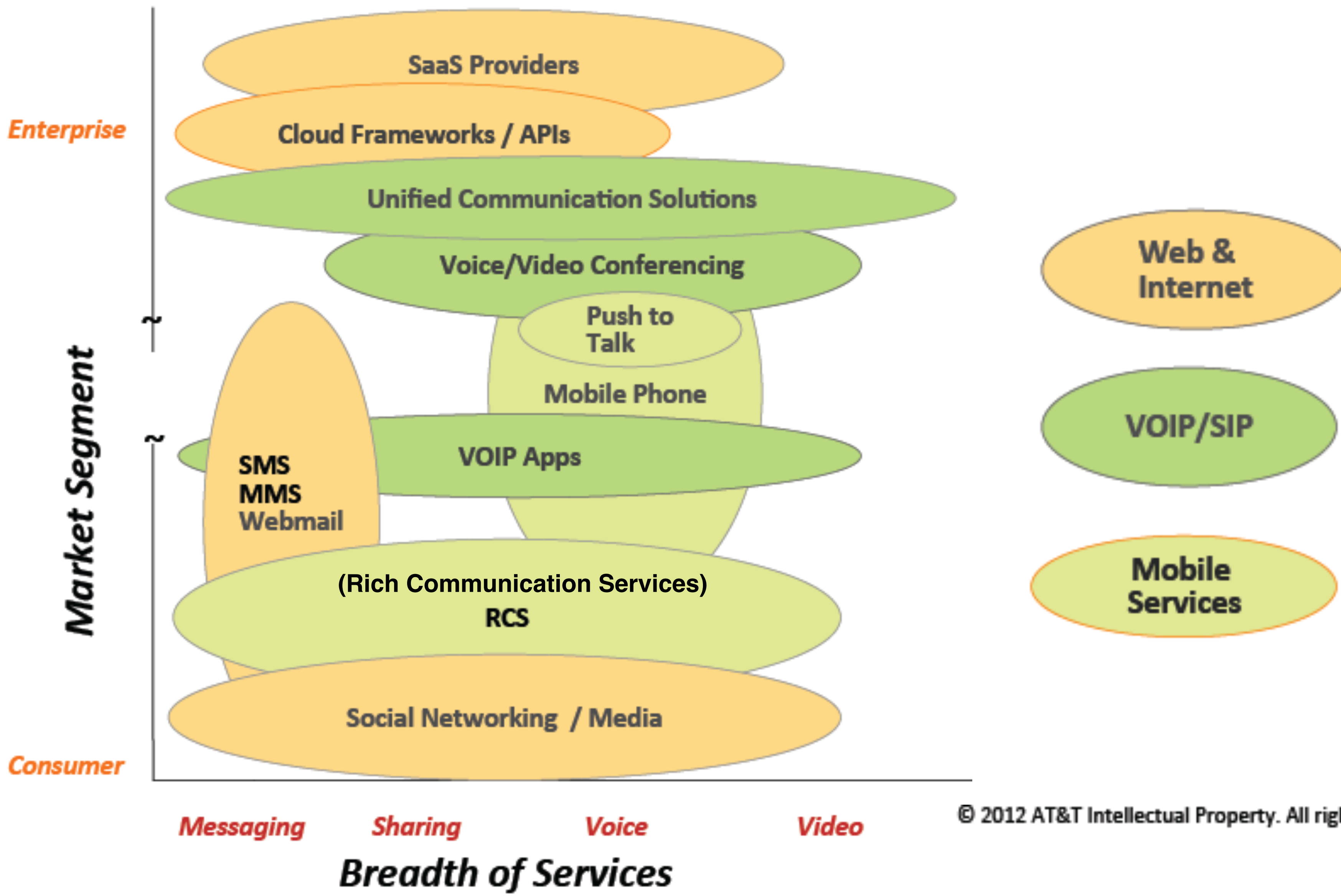
① A Browser supporting the WebRTC APIs

- GetUserMedia , RTCPeerConnection, MediaStream, DataChannel

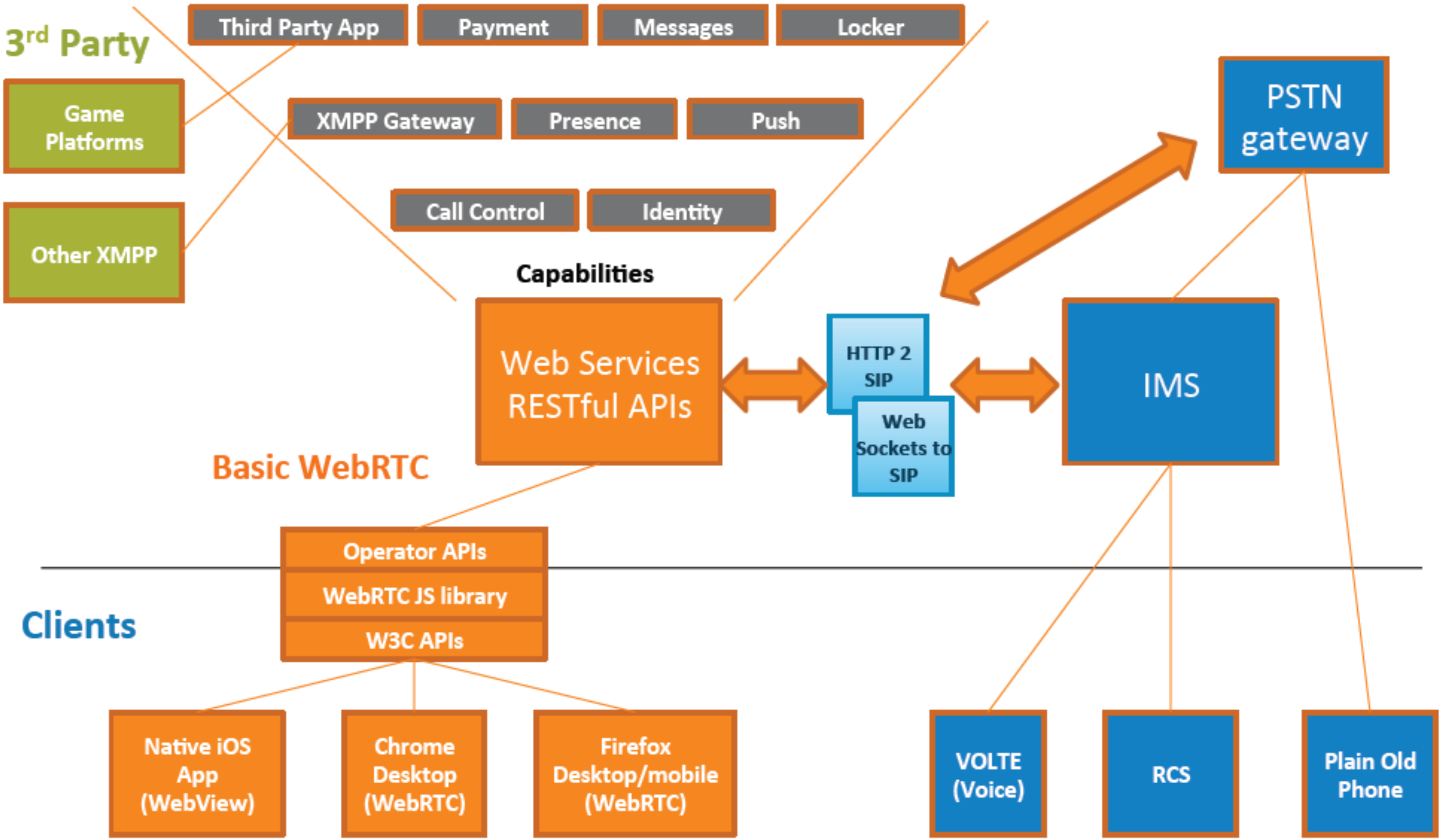
② WebRTC Service Platform with WebRTC API and/or IETF Protocol Support for Signaling, e.g. using SIP, Jingle or other Messaging Protocols.

③ A Web-based application written in Javascript which accesses WebRTC APIs provided by the Browser and the WebRTC Service Platform

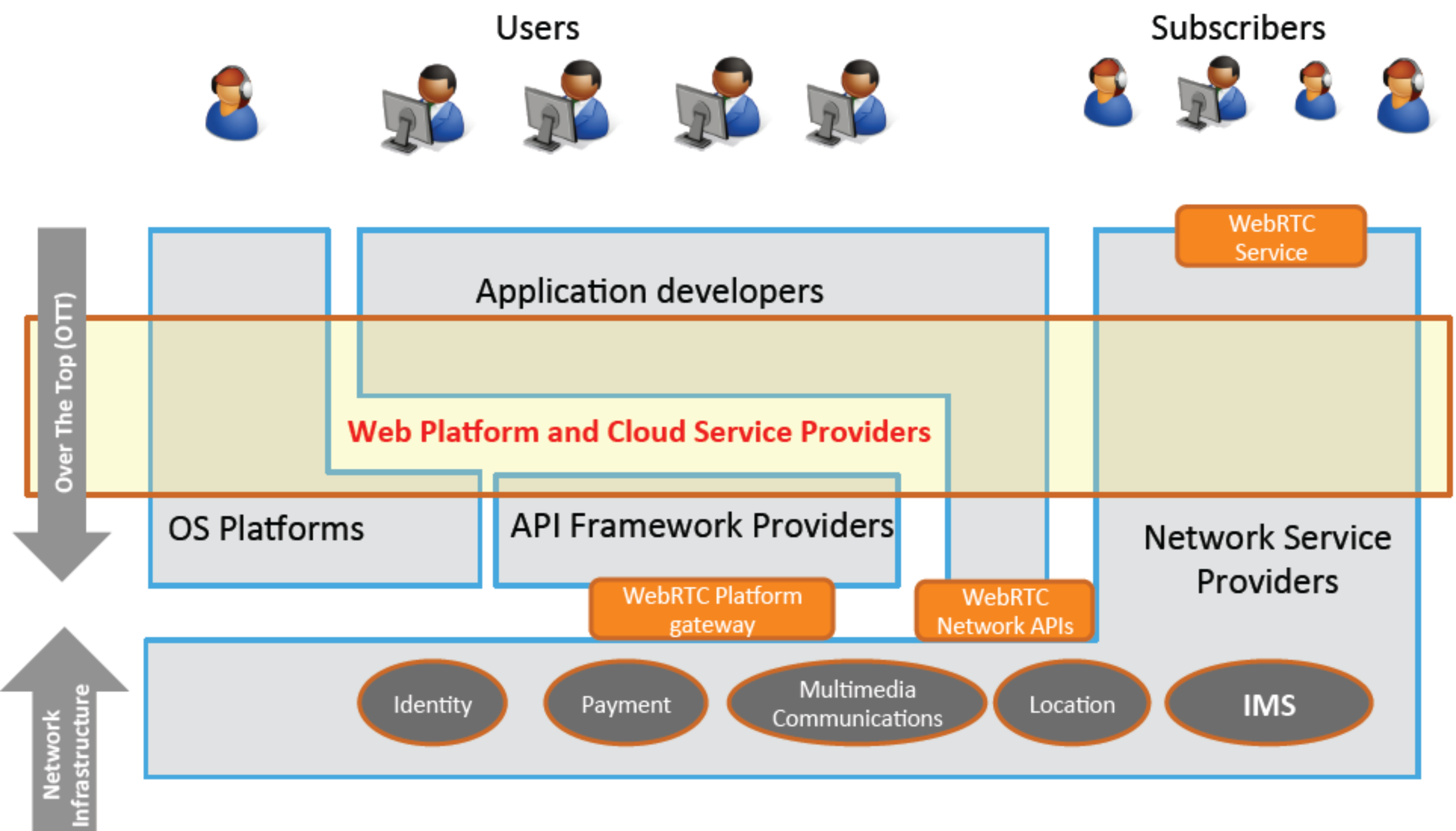
Pre-WebRTC Messaging & Real-Time Communications Services in the Market



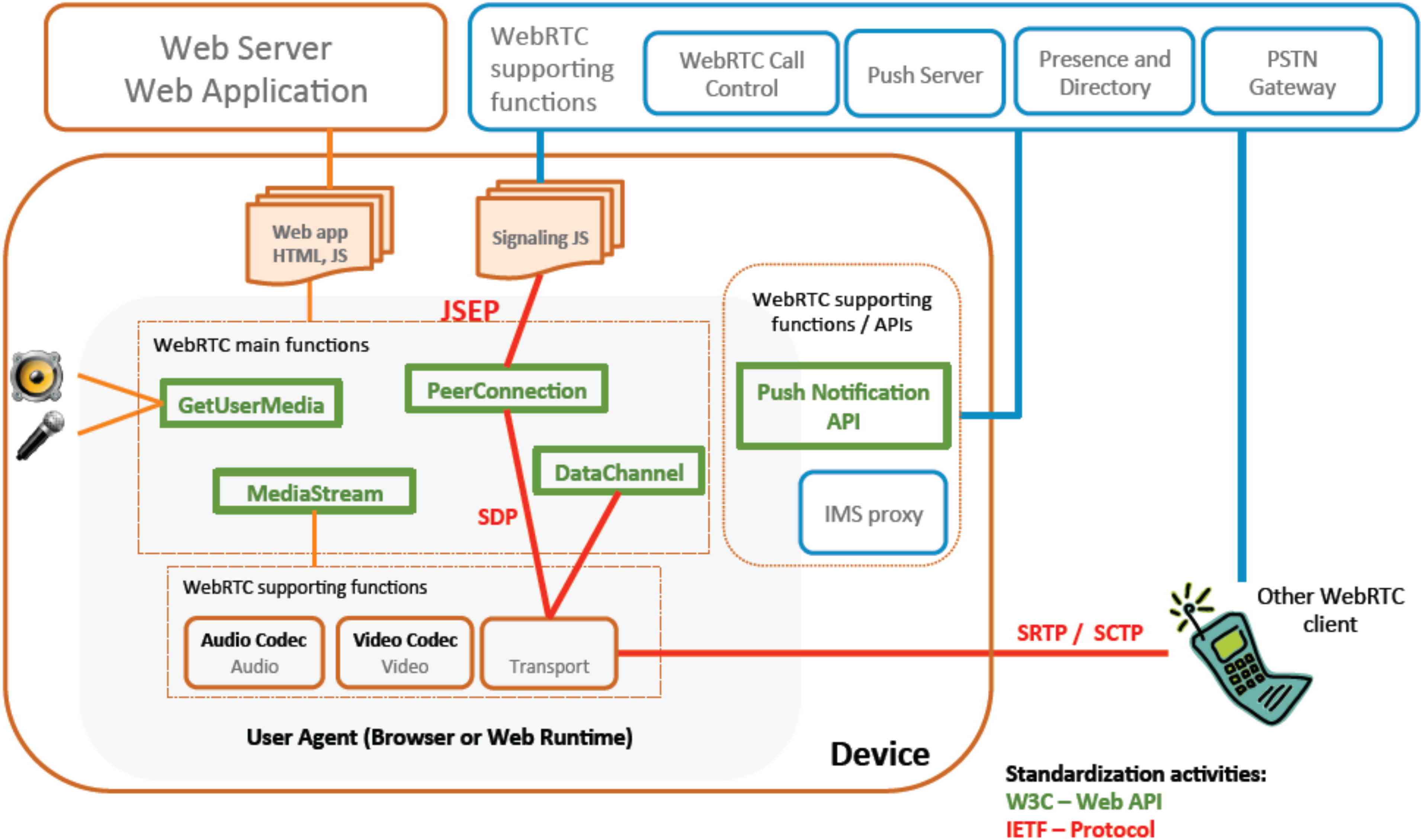
WebRTC-enabled Opportunities



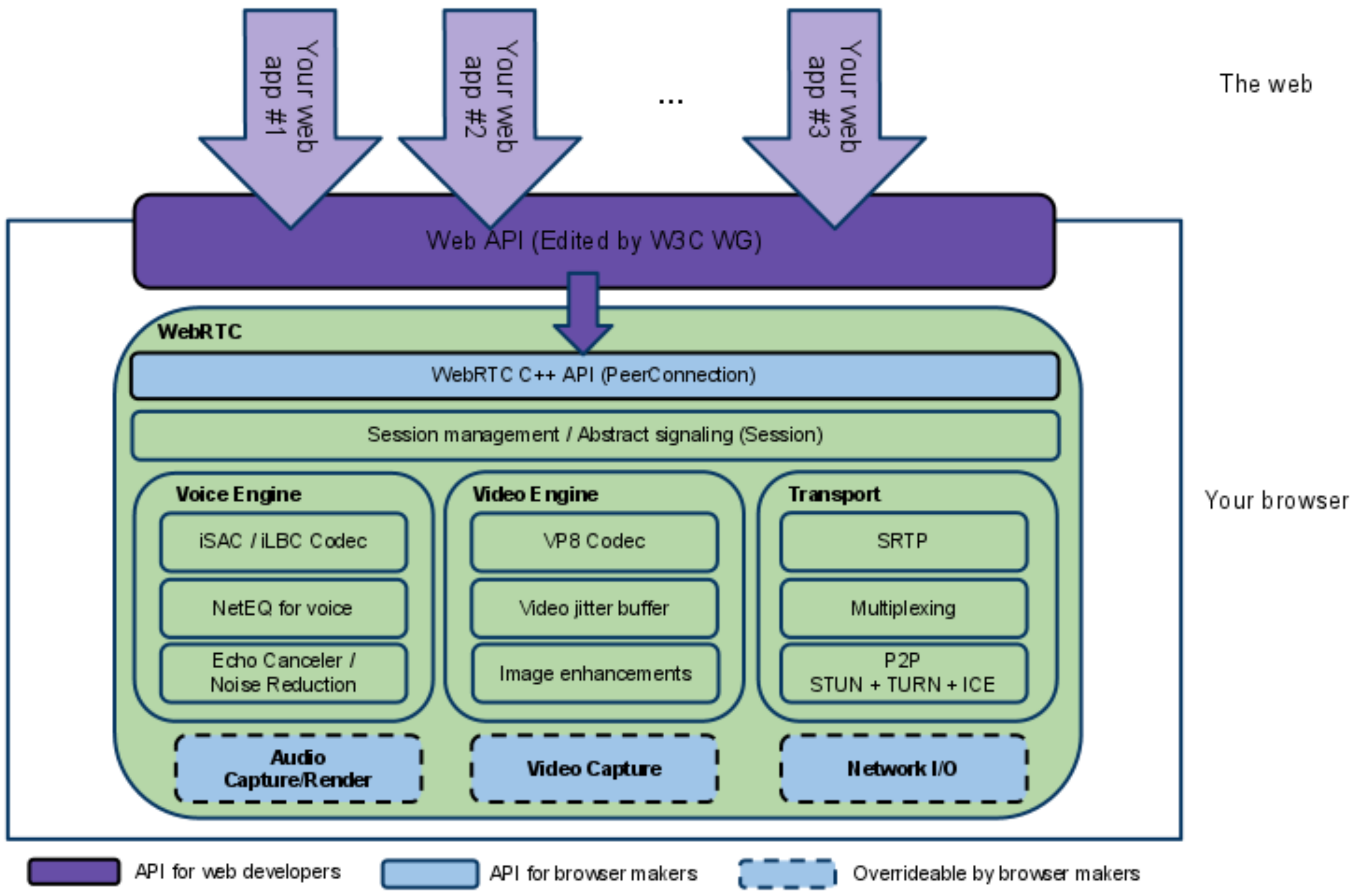
The Ecosystem of Real-Time Communication Services



WebRTC Standards and Supporting Functions

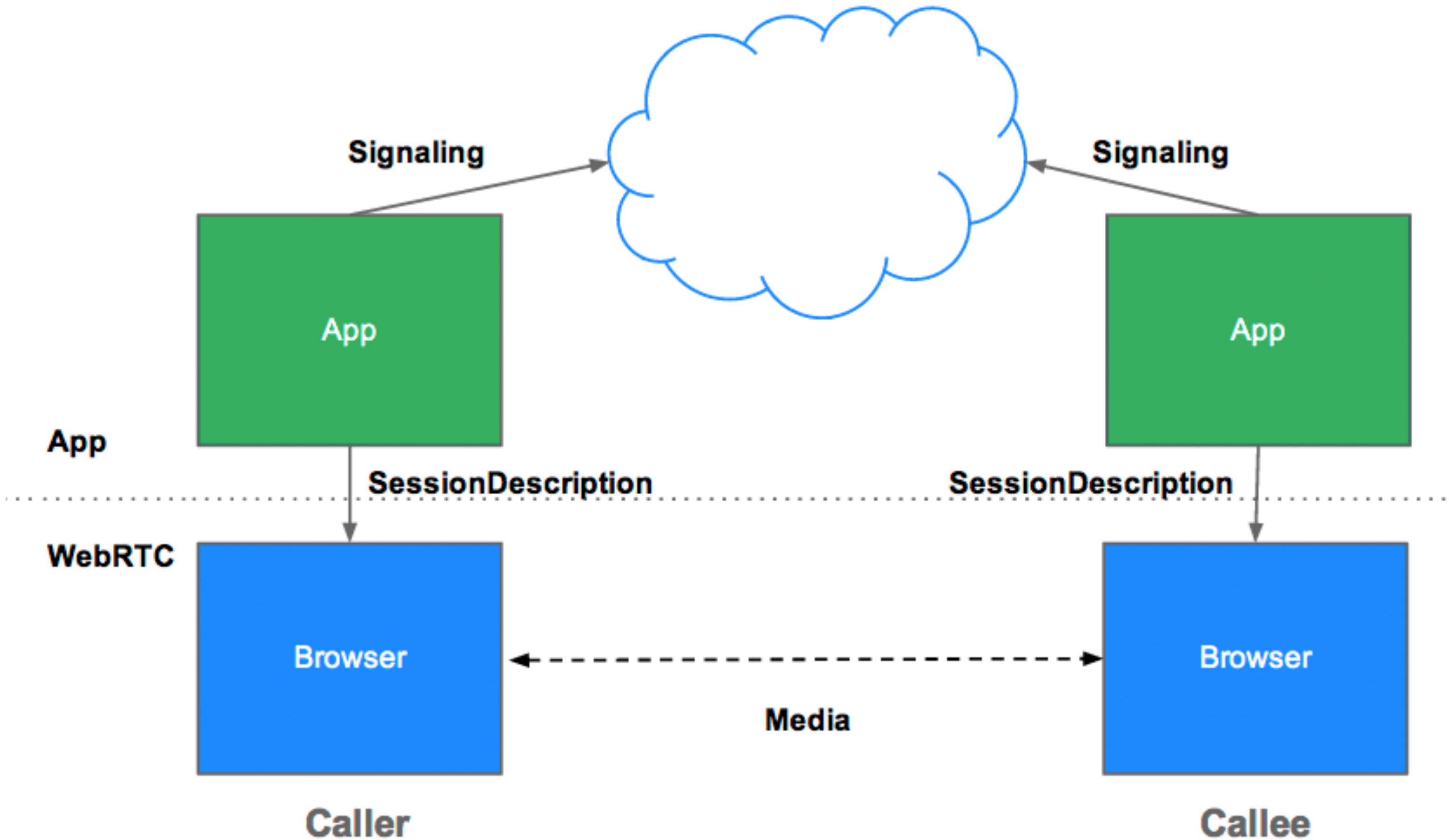


WebRTC Architecture



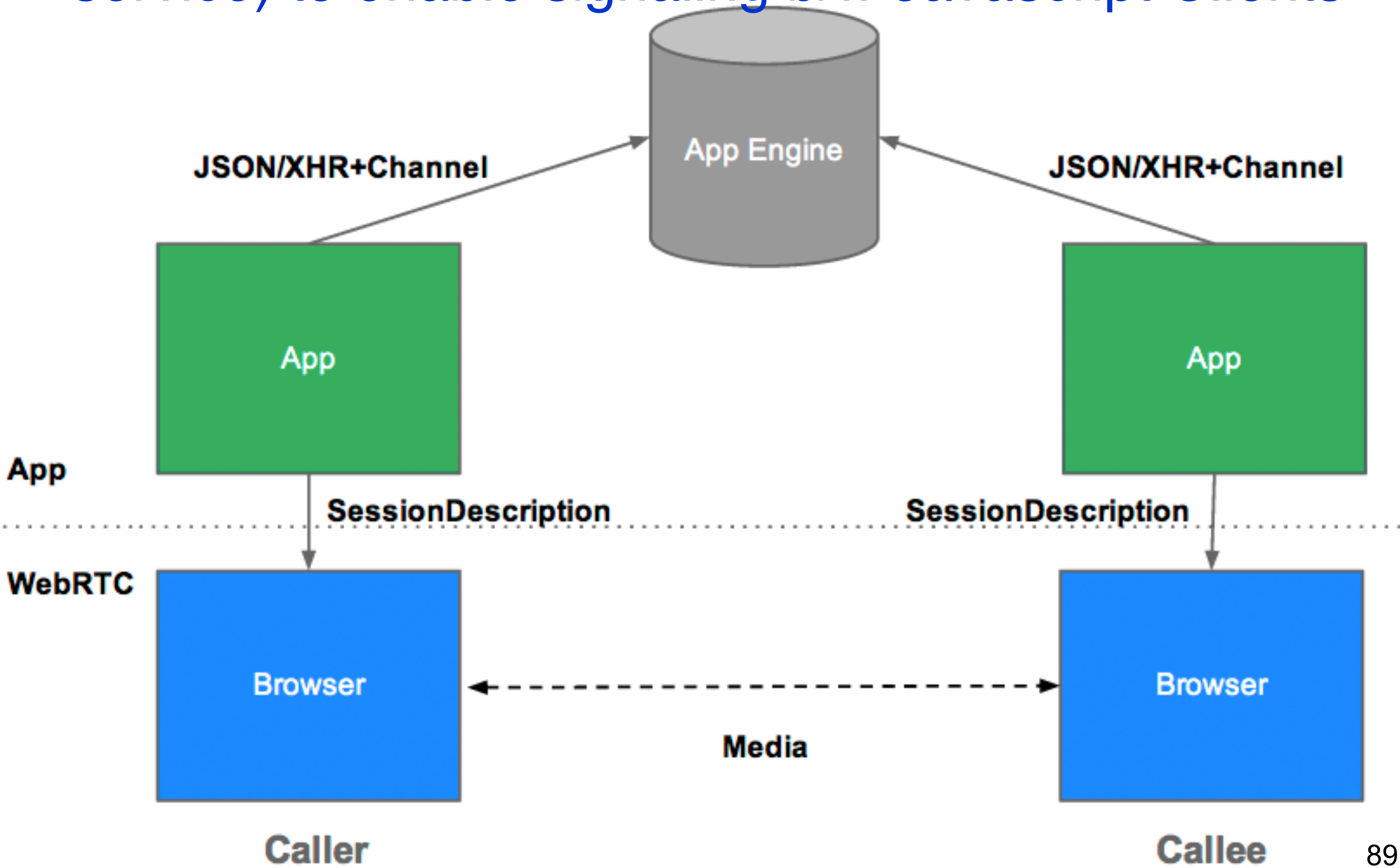
Source: webrtc.org

Javascript Session Establishment Protocol (JSEP) Architecture



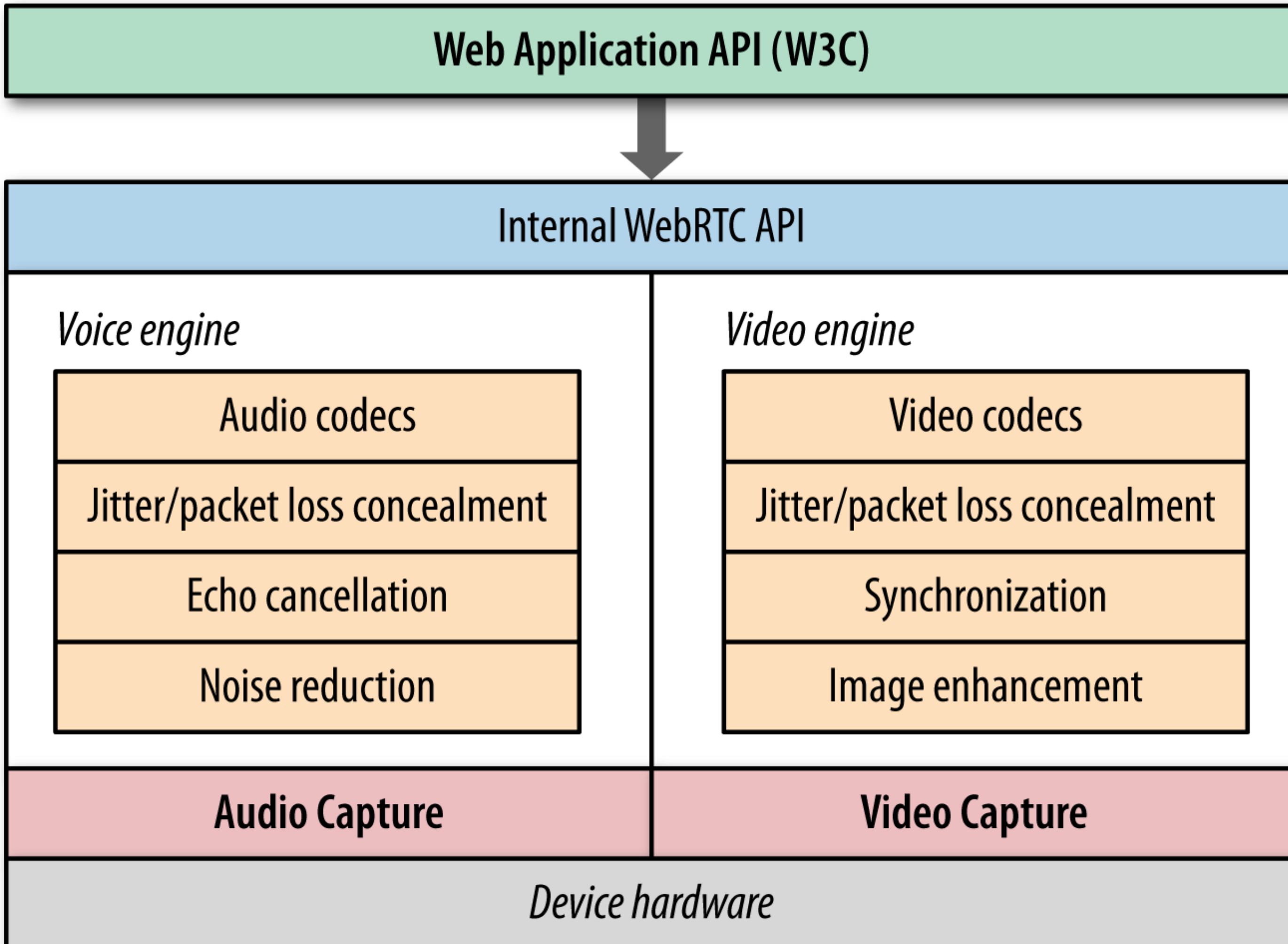
Source: Sam Dutton, http://www.html5rocks.com/en/tutorials/webrtc/basics/

A Sample Realization: A Demo App, AppRTC, which uses the Google App Engine's Channel API (Messaging service) to enable signaling b/w Javascript Clients



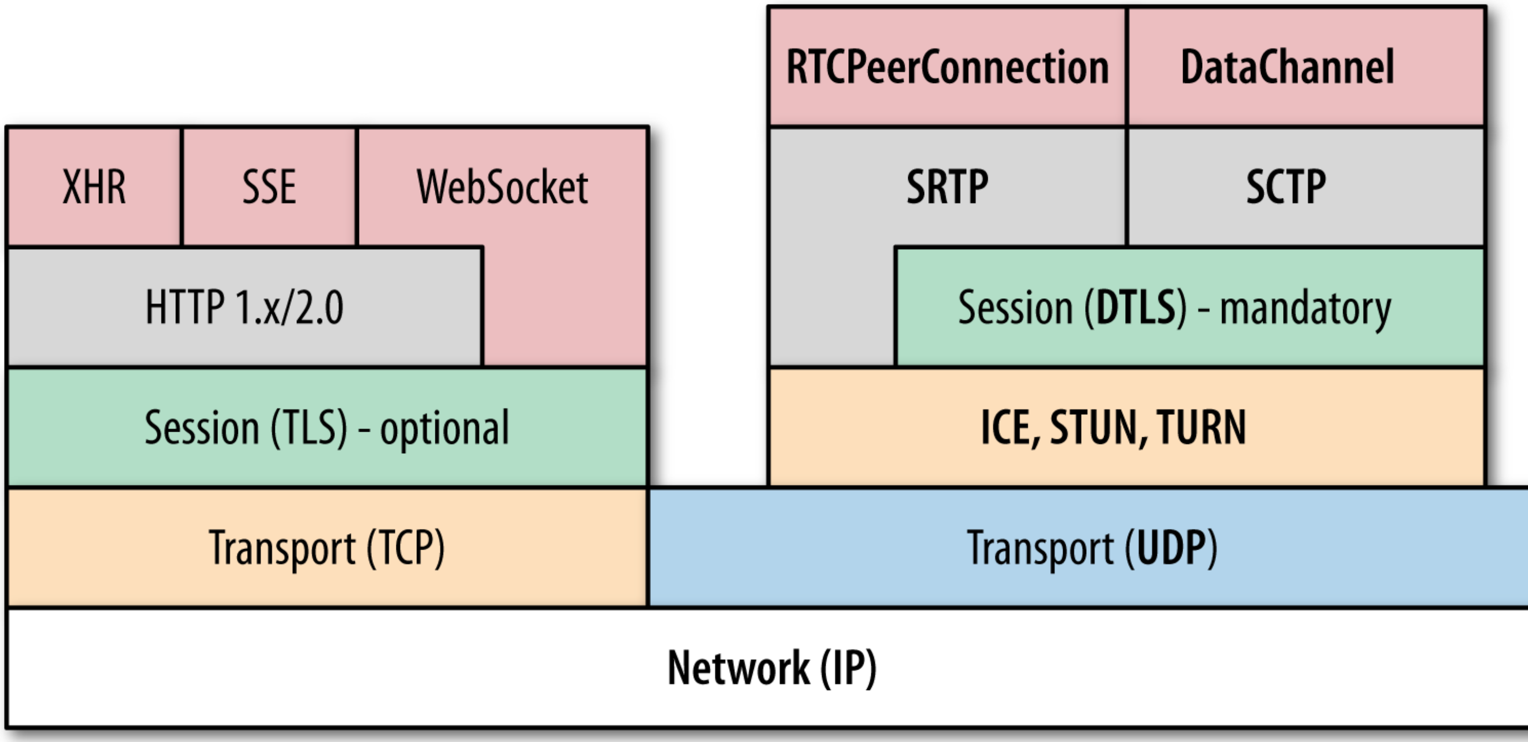
Source: Sam Dutton, http://www.html5rocks.com/en/tutorials/webrtc/basics/ ;

WebRTC Audio and Video Engines



Source: Ilya Grigorik, Ch.18 of High Performance Browser Networking, O'Reilly Publisher, http://chimera.labs.oreilly.com/books/1230000000545/index.html

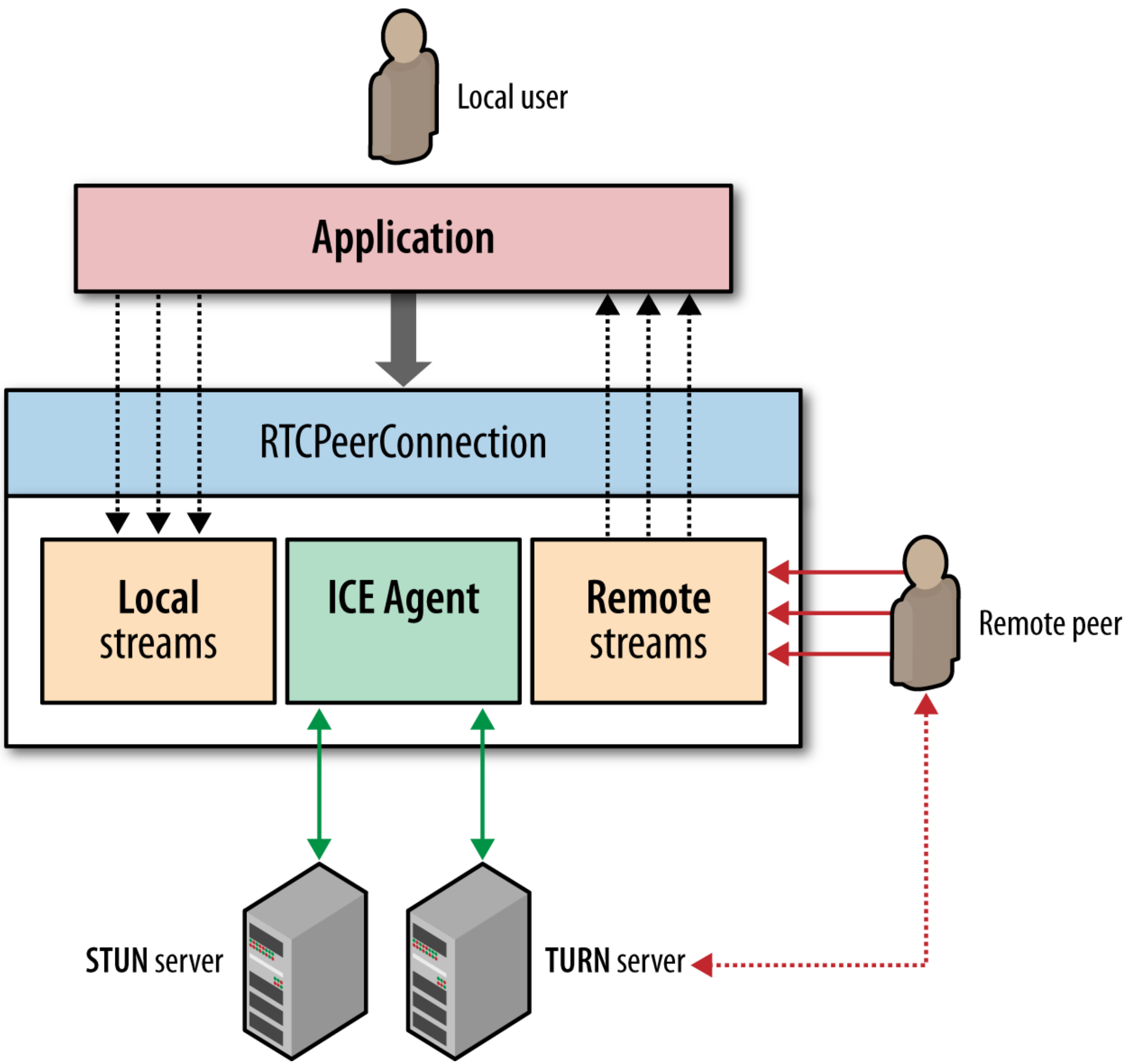
The WebRTC Networking Protocol Stack



ICE: Interactive Connectivity Establishment (RFC 5245)
STUN: Session Traversal Utilities for NAT (RFC 5389)
TURN: Traversal Using Relays around NAT (RFC 5766)
SDP: Session Description Protocol (RFC 4566)
DTLS: Datagram Transport Layer Security (RFC 6347)
SCTP: Stream Control Transport Protocol (RFC 4960)
SRTP: Secure Real-Time Transport Protocol (RFC 3711)

Source: Ilya Grigorik, Ch.18 of High Performance Browser Networking, O'Reilly Publisher, http://chimera.labs.oreilly.com/books/1230000000545/index.html

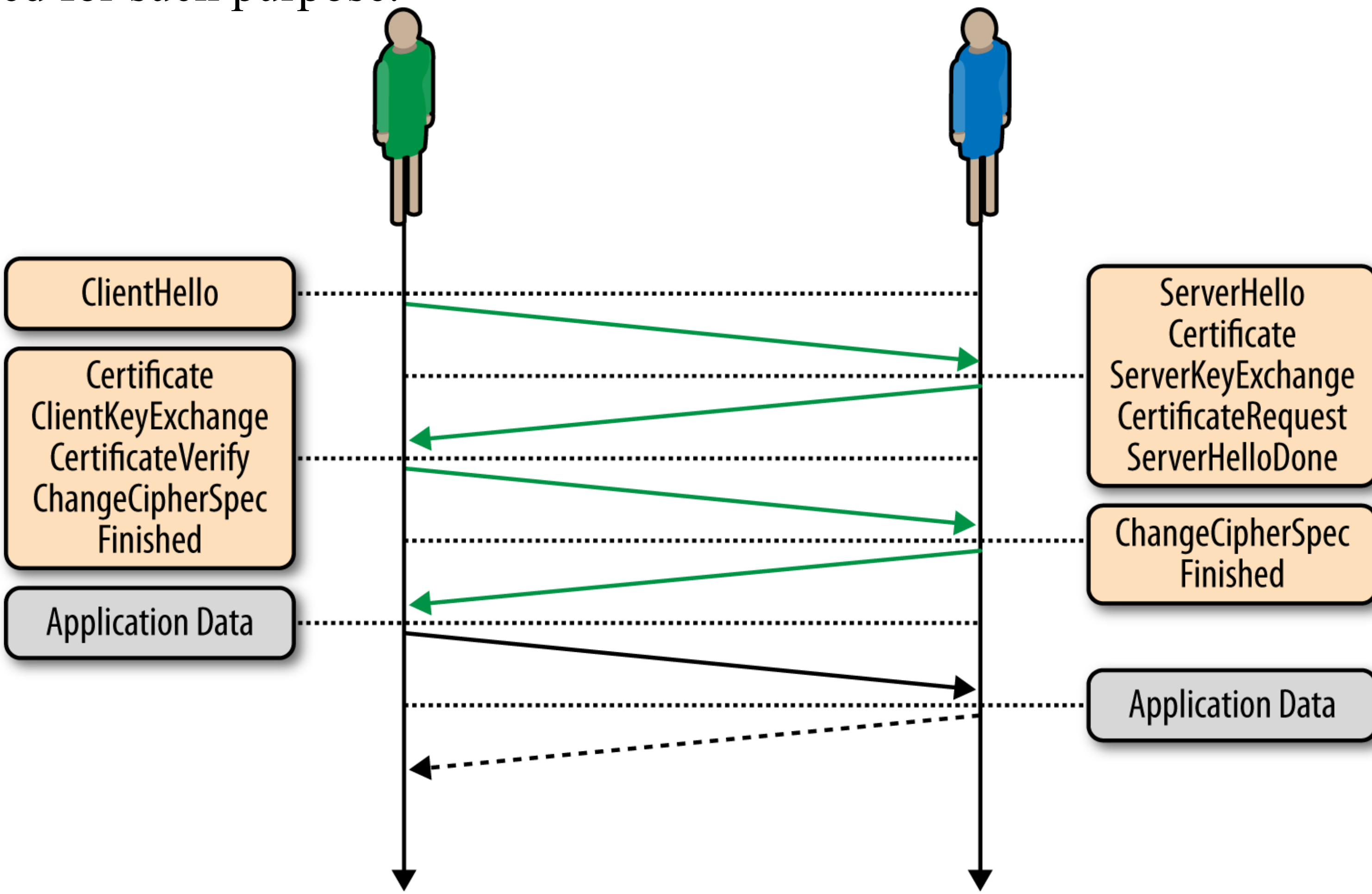
RTCPeerConnection API



Source: Ilya Grigorik, Ch.18 of High Performance Browser Networking, O'Reilly Publisher, http://chimera.labs.oreilly.com/books/1230000000545/index.html

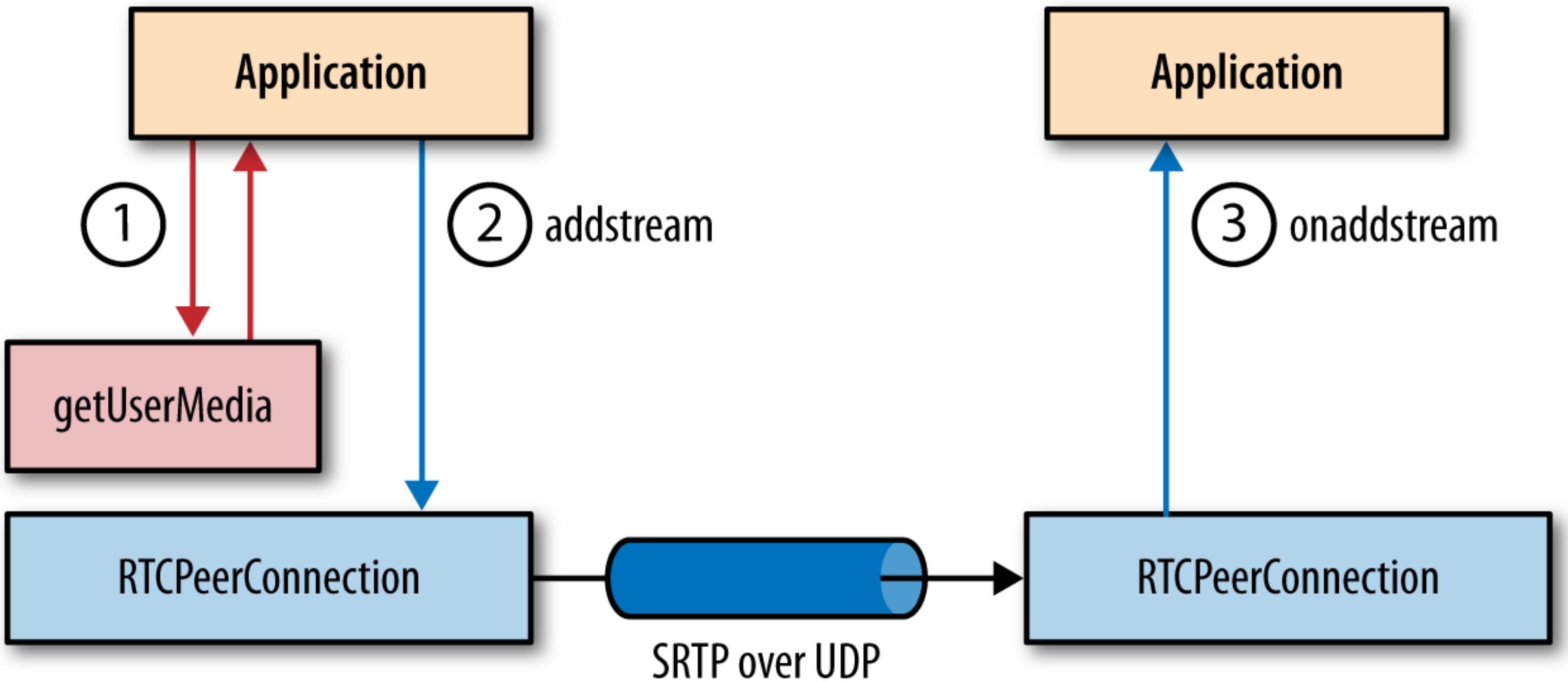
Peer-to-Peer Secure Handshake over DTLS

WebRTC standards require ALL transferred data – audio, video and application data/ payloads to be ENCRYPTED during transit ; DTLS is used for such purpose.



Source: Ilya Grigorik, Ch.18 of High Performance Browser Networking, O'Reilly Publisher, http://chimera.labs.oreilly.com/books/1230000000545/index.html

Video and Audio Delivery via Secure RTP (SRTP) over UDP



Source: Ilya Grigorik, Ch.18 of High Performance Browser Networking, O'Reilly Publisher, http://chimera.labs.oreilly.com/books/1230000000545/index.html

Deployment Status of WebRTC (circa June 2016)

- WebRTC is powering many of the Top Communications Apps:
  - Google Hangouts, Facebook Messager, Amazon Mayday,
  - Snapchat, Slack
  - Whatsapp also uses some WebRTC components according to [**]
  - Skype is moving to WebRTC

  3 Billion+ WebRTC apps downloaded so far !
- 1.5 Billion+ WebRTC browsers
  - Chrome, Firefox, Opera, Microsoft Edge
  - WebRTC for WebKit browser (of Android & IOS) under development

[**] webrtchacks.com/whats-up-with-whatsapp-and-webrtc

IE	Edge *	Firefox	Chrome	Safari	Opera	iOS Safari *	Opera Mini *	Android Browser *	Chrome for Android
8			45					4.3	
9			46					4.4	
10		43	47			8.4		4.4.4	
11	13	44	48	9	34	9.2	8	47	47
	14	45	49	9.1	35	9.3			
		46	50		36				
		47	51						

caniuse.com/webrtc

Additional References

- http://www.webrtc.org
- Sam Dutton, http://www.html5rocks.com/en/tutorials/webrtc/basics/
- Ilya Grigorik, Ch.18 of High Performance Browser Networking, O'Reilly:
  - http://chimera.labs.oreilly.com/books/1230000000545/index.html
- Cullen Jenngins, Ted Hardie, Magnus Westerlund, "Real-Time Communications over the Web," IEEE Communications Magazine, Vol. 51, pp.20-26, 2013
- Justin Uberti, Sam Dutton, ``Real-Time Communication with WebRTC,'' Google I/O 2013
  - http://io13webrtc.appspot.com/#1
  - http://www.youtube.com/watch?v=p2HzZkd2A40&t=21m12s
- AppRTC, a WebRTC demo hosted on the Google App Engine,
  - http://www.webrtc.org/demo
  - https://apprtc.appspot.com/
- Another set of WebRTC Demo Apps:
  - http://generative.edb.utexas.edu/webrtc-demos/
- https://bloggeek.me/quic-webrtc/
- Cullen Jennings, "What's Next with WebRTC," Sept 2016