HTTP/2からQUICへ続く Webプロトコルの進化

大津 繁樹

IIJ Technical WEEK 2015

2015年11月11日

自己紹介

- 大津 繁樹
- 株式会社 インターネットイニシアティブ
  - プロダクト本部 アプリケーション開発部サービス開発2課

- NodeJS Technical Steering Committee メンバー
  - (主にTLS/CRYPTO/OpenSSLバインディングを担当)
- IETF httpbis WG で HTTP/2相互接続試験等仕様策定に参画。

内容

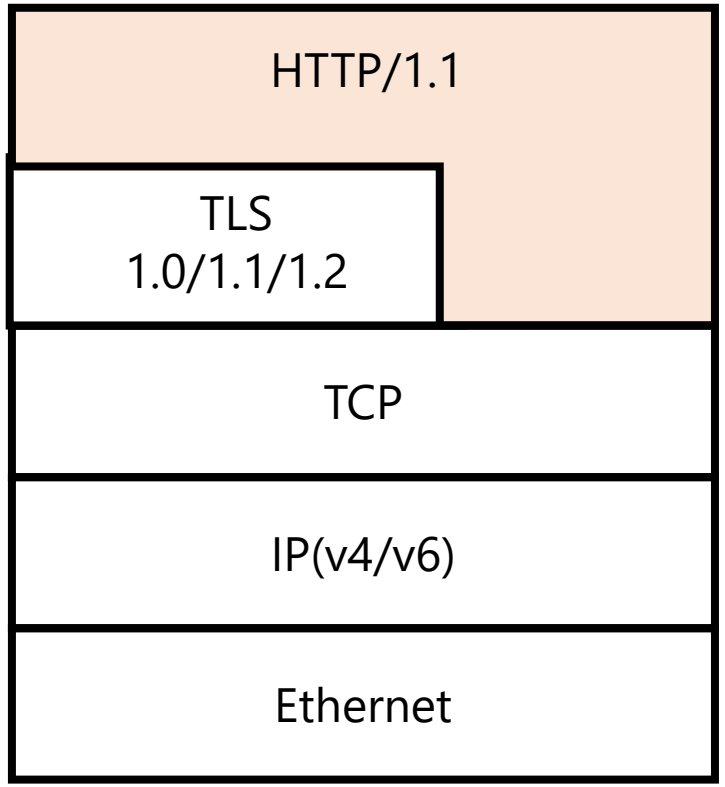
Webプロトコルの進化とこれからについて次のフェーズ毎にその概要と見通しを解説します。

1. HTTP/1.1 からHTTP/2へ
2. HTTP/2 からQUIC へ
3. QUIC からTLS1.3(*) へ

(*注意) 内容は2015年11月2日時点での TLS1.3 draft (*1)を元にしています。今後の仕様策定作業で本プレゼンの内容が変更になる場合がありますのでご注意ください。

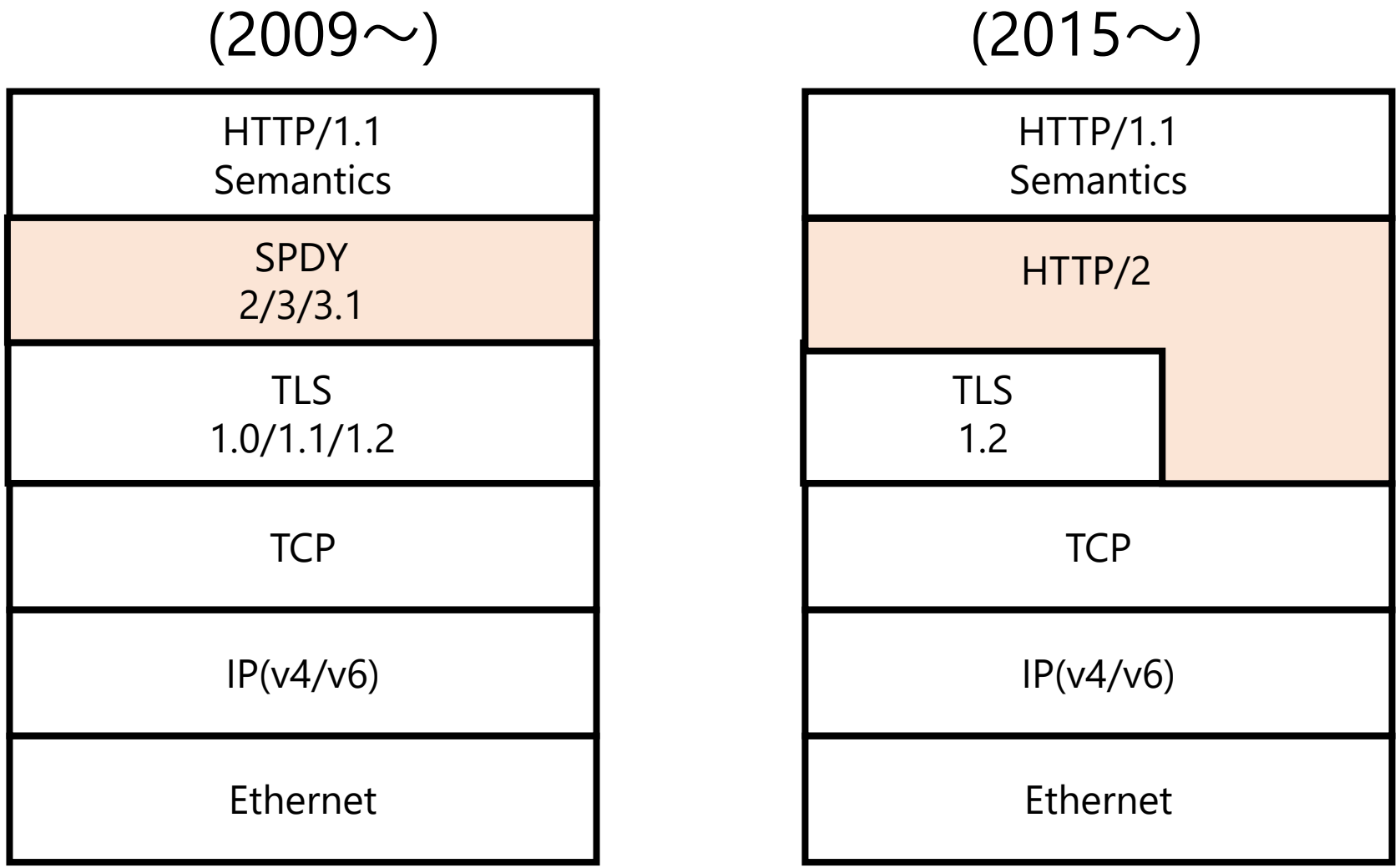
(*1 https://github.com/tlswg/tls13-spec/ の master HEAD)

0. HTTP/1.1 の時代 (1999〜)



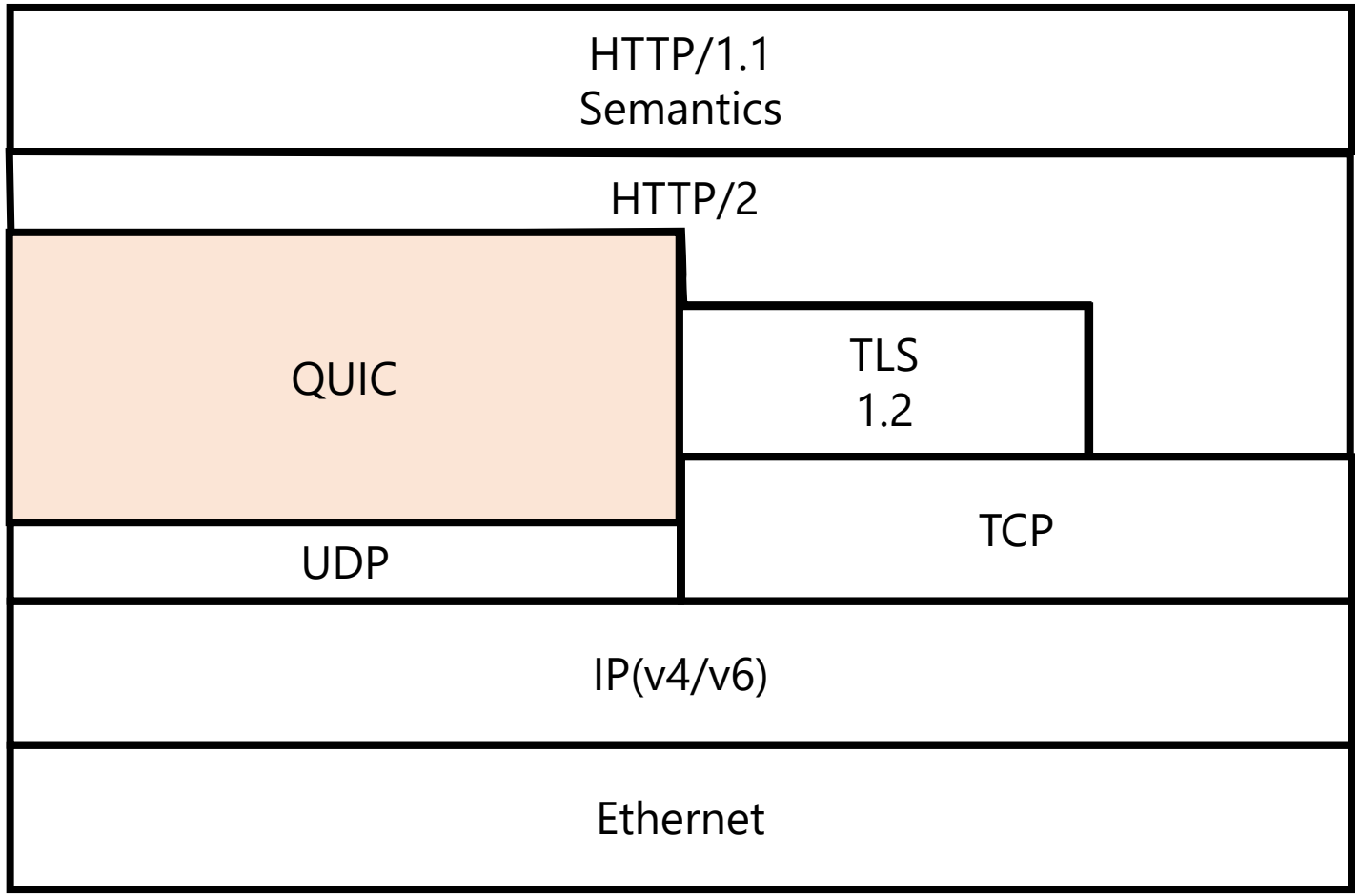
(* TLS1.1 2006〜, TLS1.2 2008〜)

1. HTTP/1.1からHTTP/2へ



2. HTTP/2からQUICへ

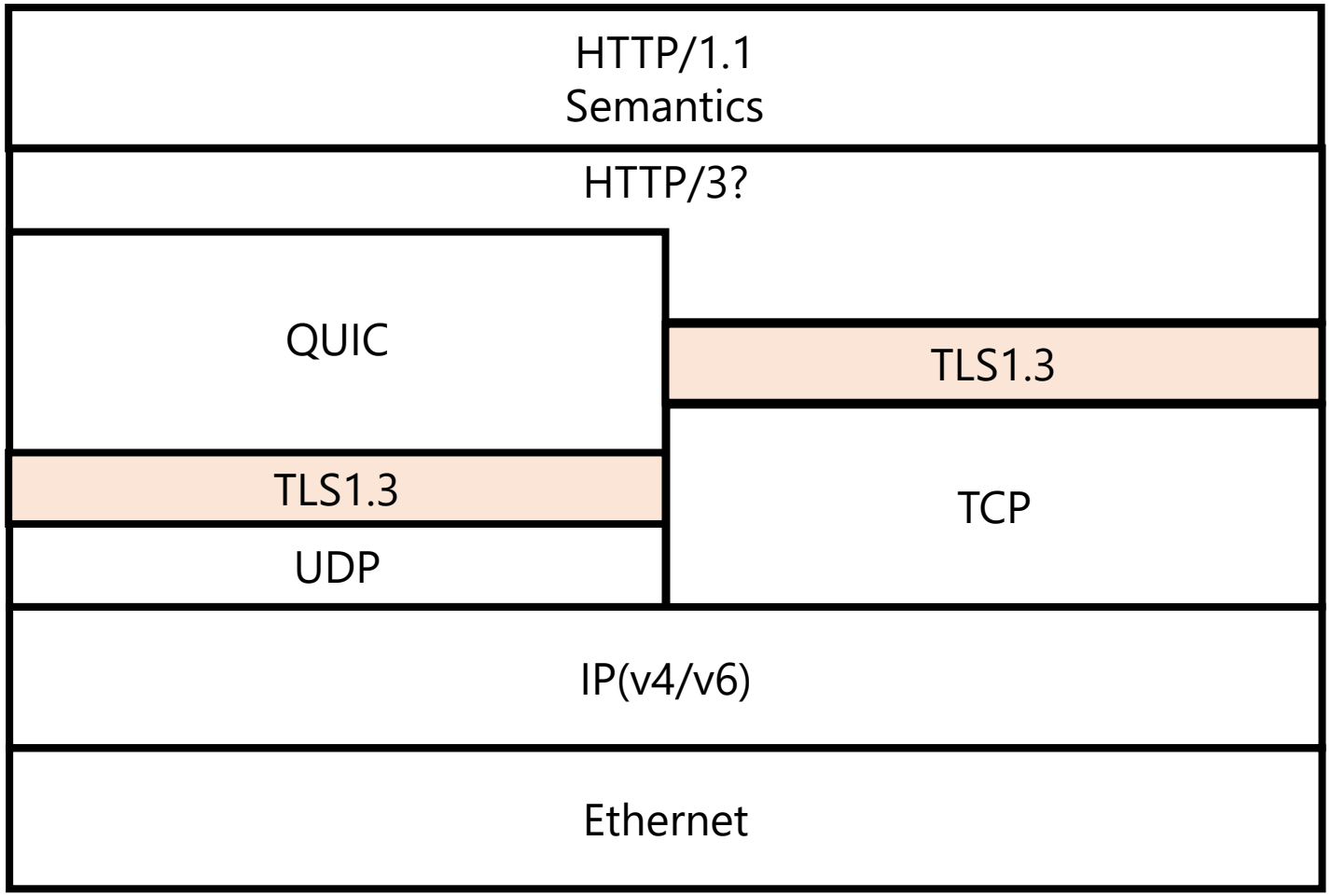
(2013～)



(* HTTP/2 2015～)

3. QUICからTLS1.3へ

(2016 or 2017?〜)



HTTP/1.1からHTTP/2へ

HTTPプロトコルの年表

1990 1995 2000 2005 2010 2015

Webの始まり

HTTP/0.9

HTTP/1.0
RFC1945

HTTP/1.1
RFC2068

HTTP/1.1
RFC2616

HTTP-NG
中止

暗黒の時代

httpbis WG

HTTP/1.1
RFC7230-5

SPDY/2

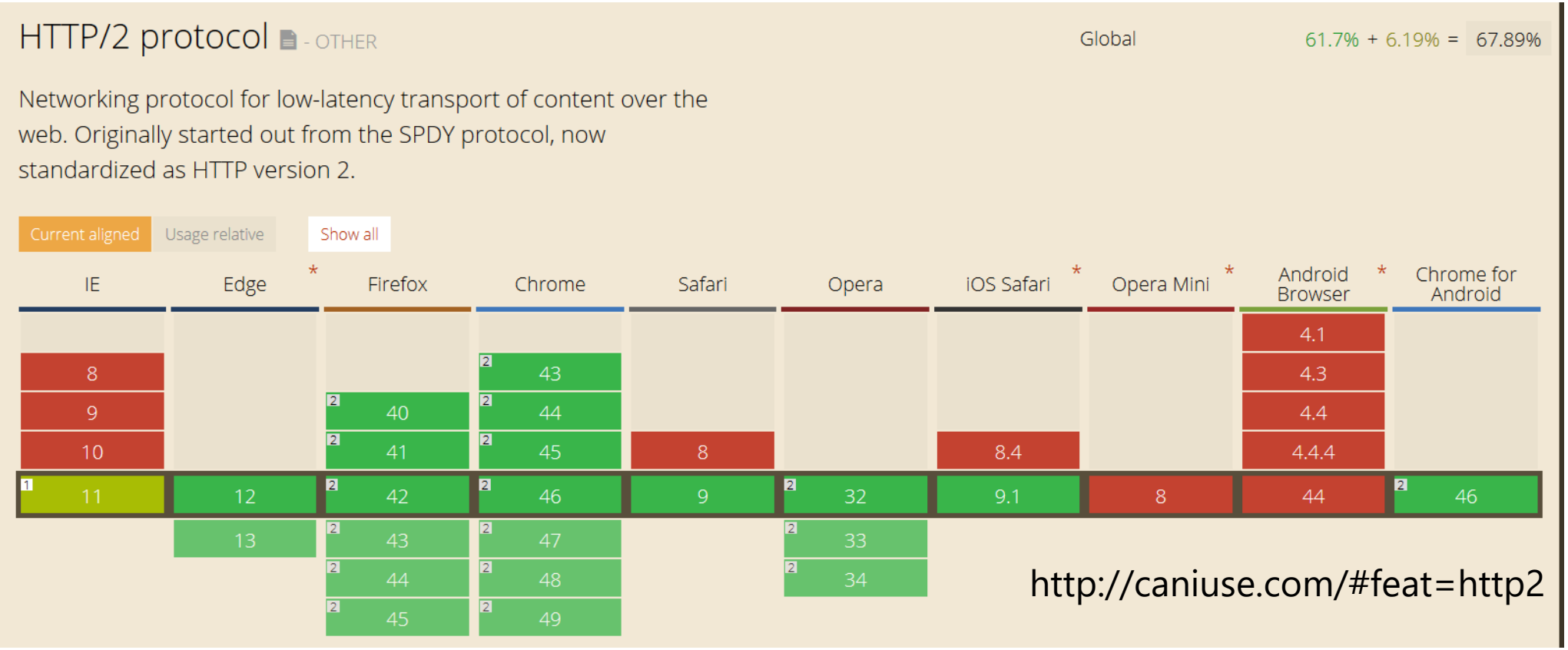
SPDY/3

SPDY/3.1

HTTP/2
RFC7540

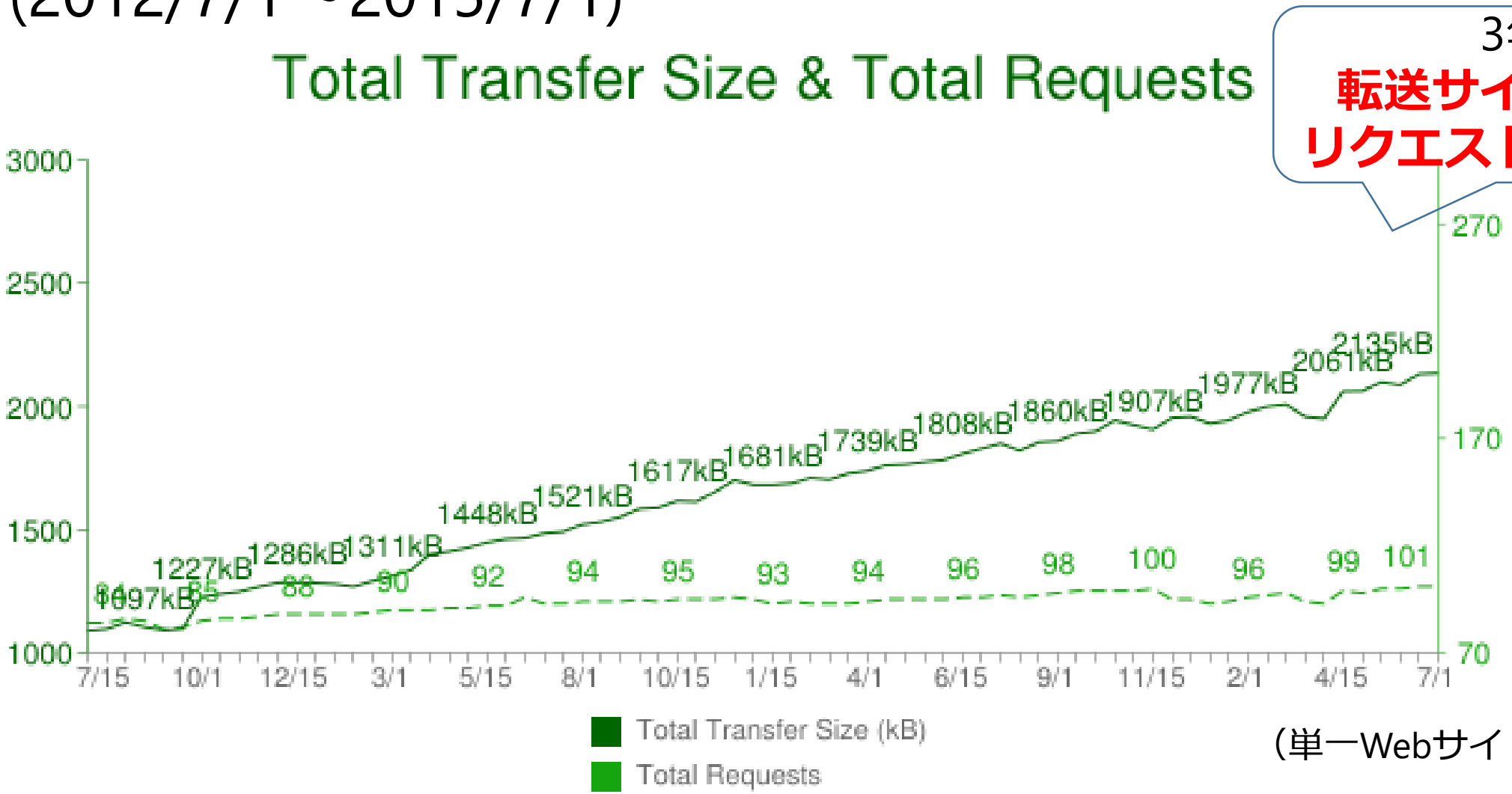
HPACK
RFC7541

HTTP/2サポート状況



サーバ側は nghttp, jetty, h2o, nginx, apache2 等続々登場中

HTTP転送サイズとリクエスト数の遷移
(2012/7/1～2015/7/1)

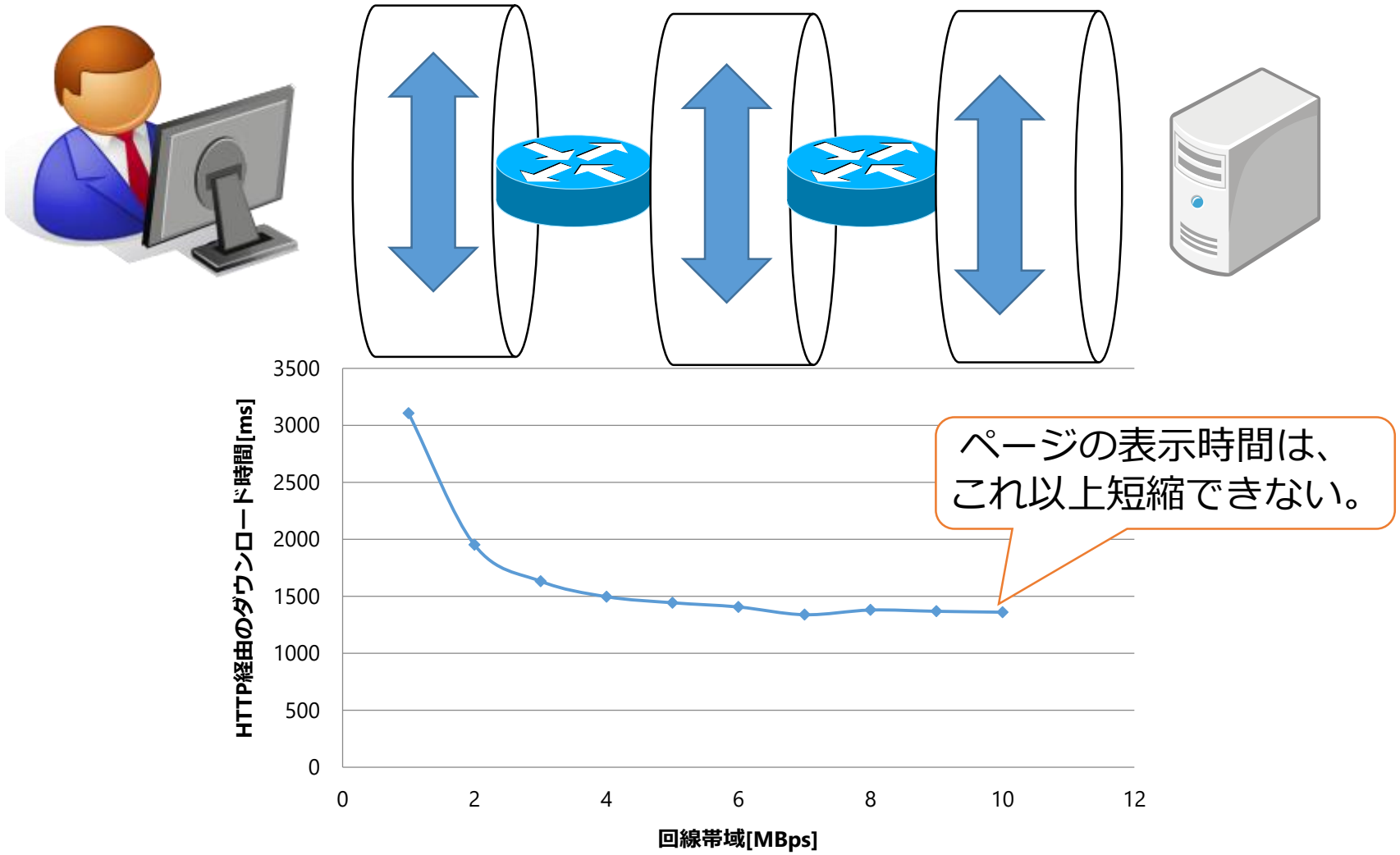


3年で
転送サイズ: 96%増
リクエスト数：20%増

（単一Webサイトの統計平均）

http://httparchive.org/trends.php?s=All&minlabel=Jul+1+2012&maxlabel=Jul+1+2015#bytesTotal&reqTotal

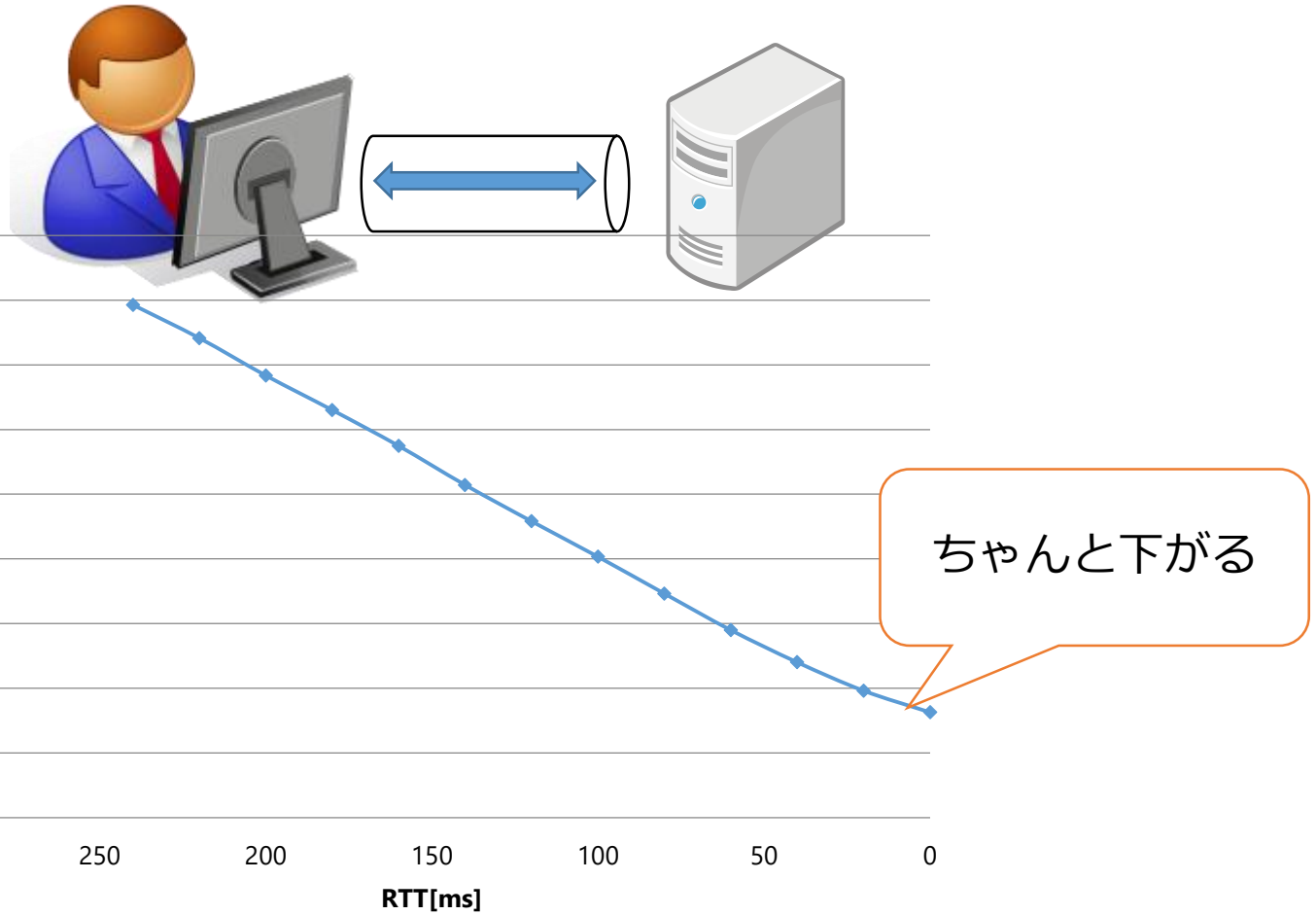
回線帯域を増速していくと



More Bandwidth does'nt matter よりデータ引用
http://docs.google.com/a/chromium.org/viewer?a=v&pid=sites&srcid=Y2hyb21pdW0ub3JnfGRldnxneDoxMzcyOWI1N2I4YzI3NzE2

Make the Web Faster: Google の試験

RTT(Round Trip Time)を小さくしていくと

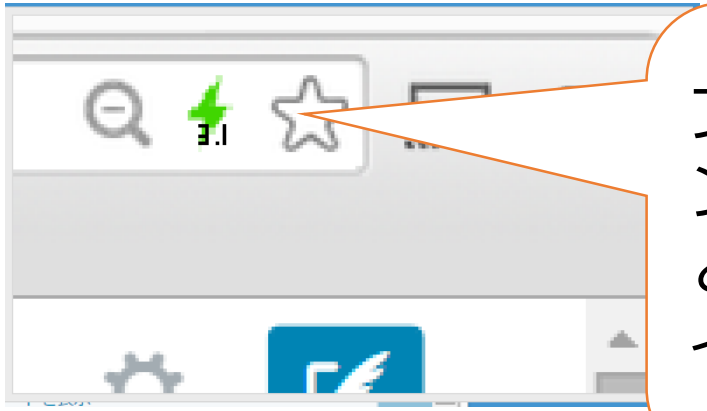


Webページの表示速度を速くするには、回線速度増強よりRTT（の影響）を小さくするかが重要。でも物理的な制限で難しい。

More Bandwidth does'nt matter よりデータ引用
http://docs.google.com/a/chromium.org/viewer?a=v&pid=sites&srcid=Y2hyb21pdW0ub3JnfGRldnxneDoxMzcyOWI1N2I4YzI3NzE2

ＳＰＤＹ(スピーディ）の登場（2009年）

- RTTの影響をできるだけ避けるべくGoogleはSPDYを開発した。
- SPDYは、Ｗｅｂページの表示速度を速くするためのプロトコルとして当初社内プロジェクトから生まれた。
- 既に3年以上に渡りGoogleの全サービスで利用され、TwitterやFacebook、LINEなど大規模なシステムに導入されている。
- Googleは2016年初めにSPDYを廃止する予定

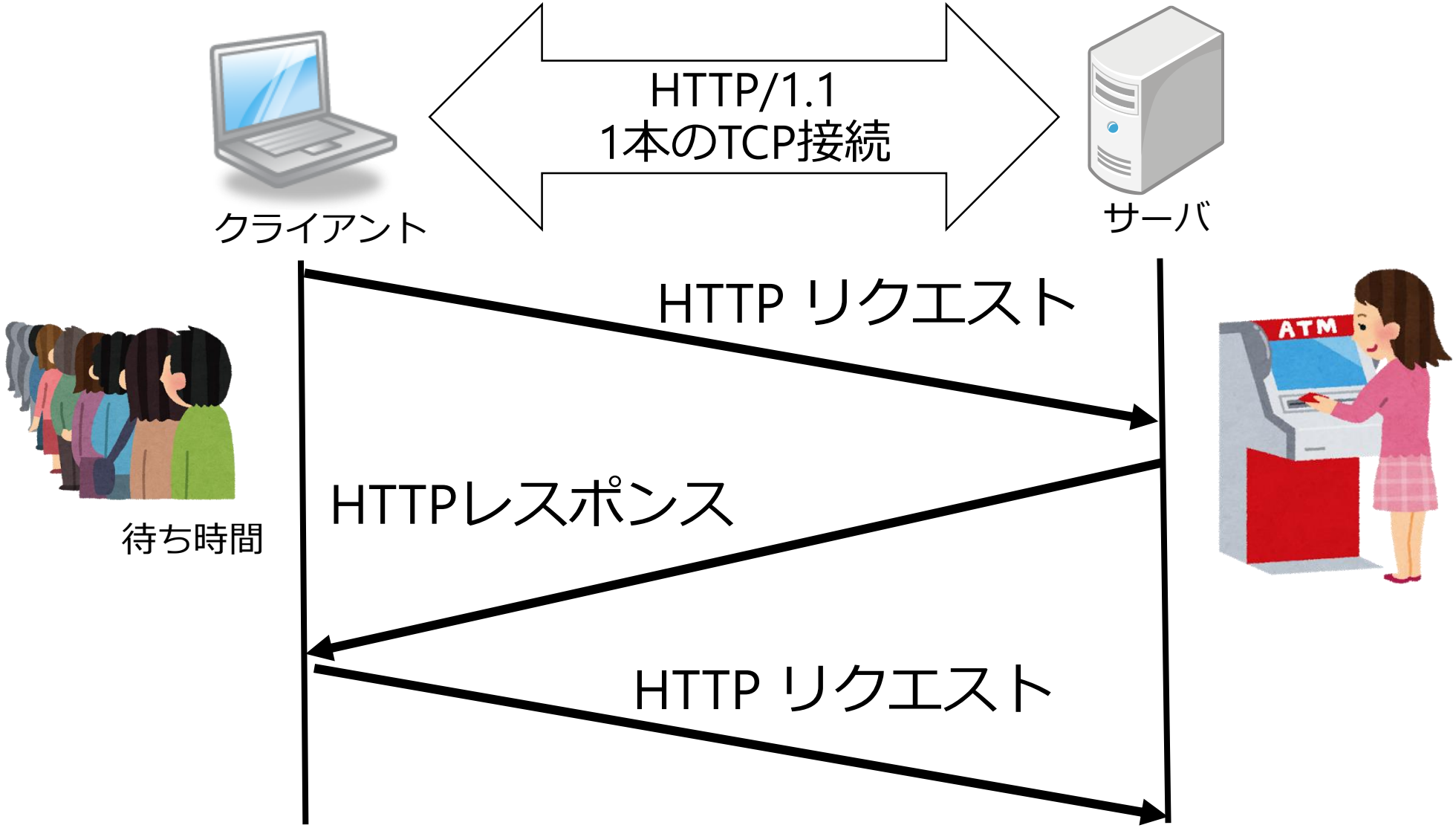


ブラウザの拡張プラグイン(SPDY Indicator)を使うとSPDYを使っているサイトがわかる

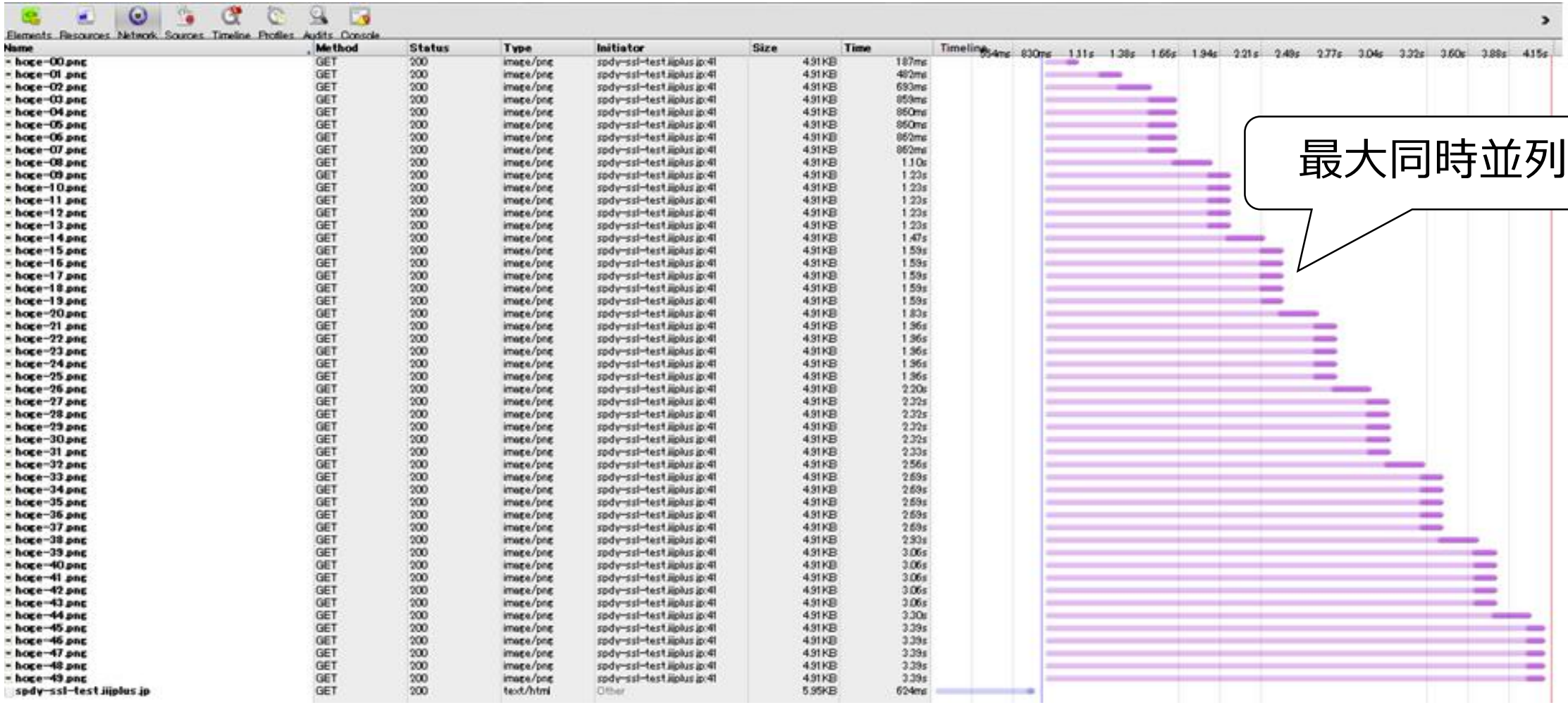
HTTP/1.1の問題点のおさらい

1. HTTP Head of Line Blocking
2. ネットワーク通信の利用が非効率
3. 曖昧でテキスト処理が煩雑

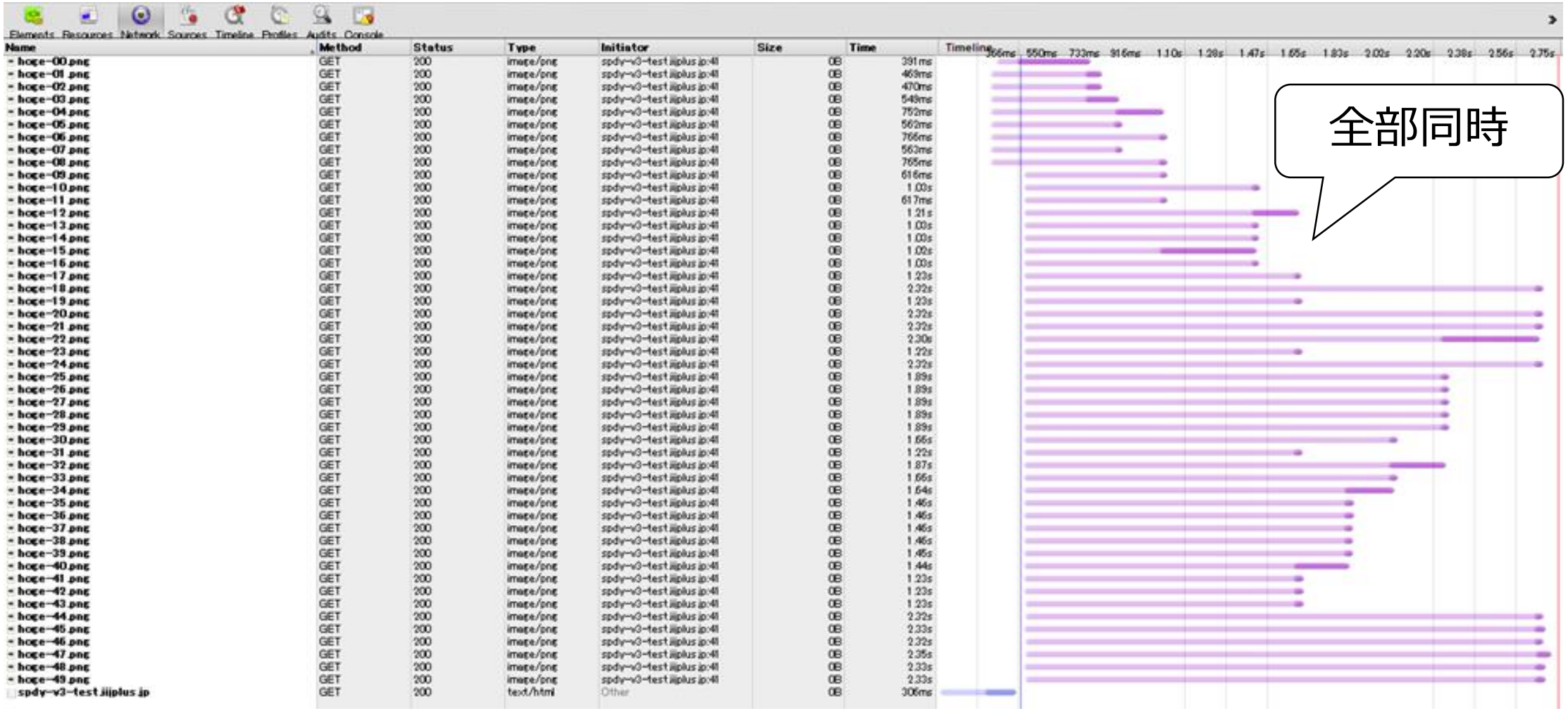
1. HTTP Head of Line Blocking



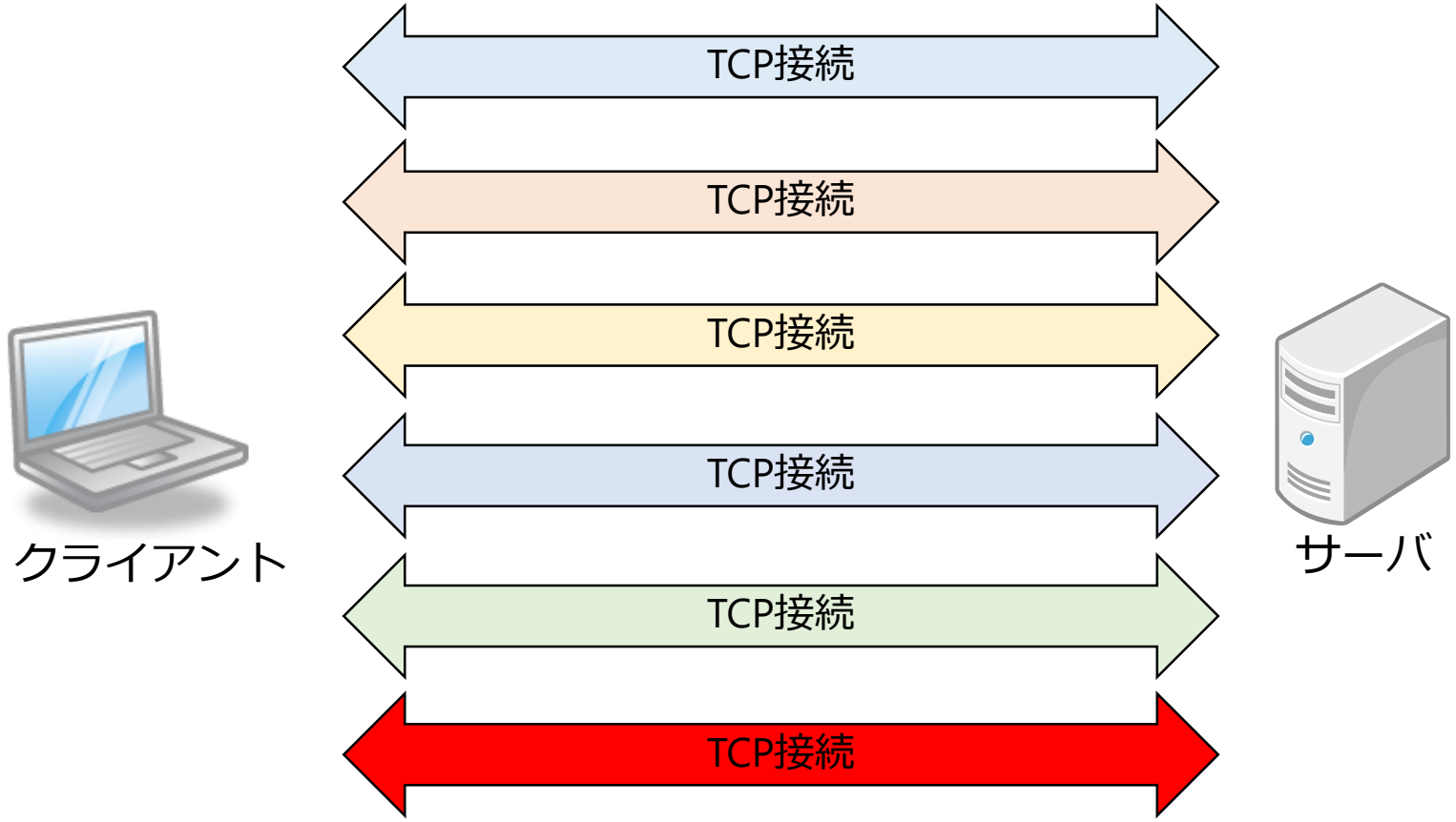
HTTP/1.1
ブラウザは最大同時4～6TCP接続に制限



HTTP/2
100以上の同時リクエストが可能

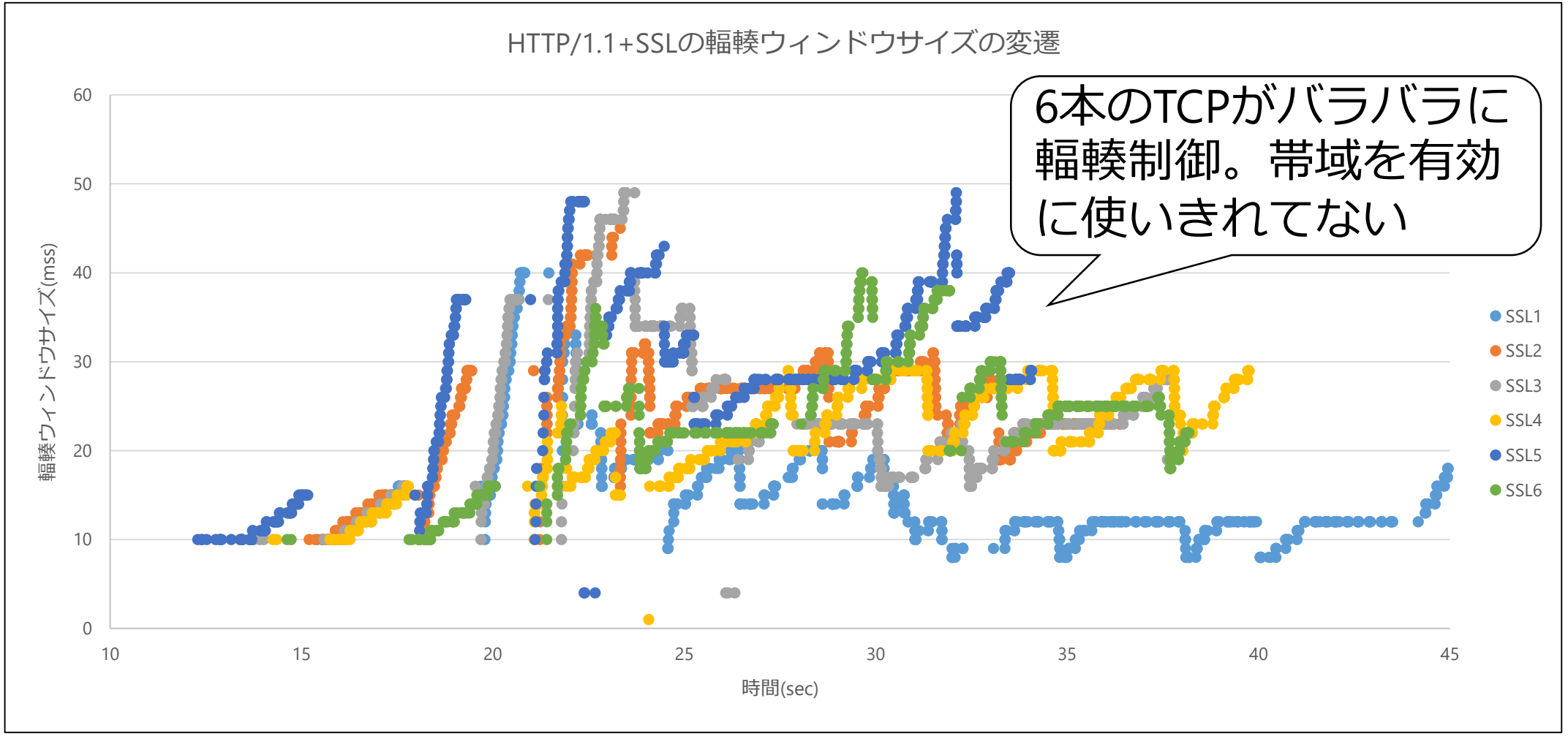


2. HTTP/1.1はネットワーク通信の利用が非効率

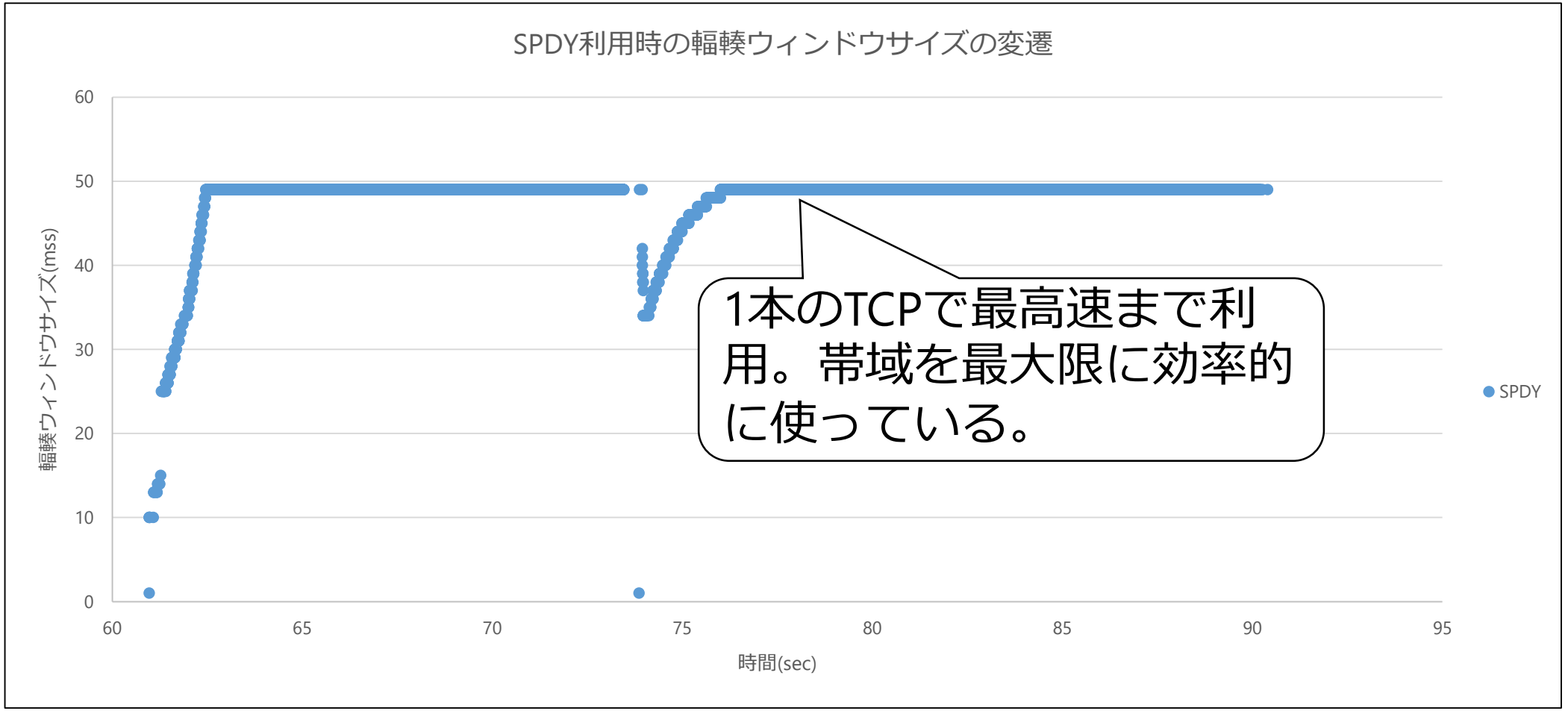


それぞれのTCP接続が独立して輻輳制御を行う

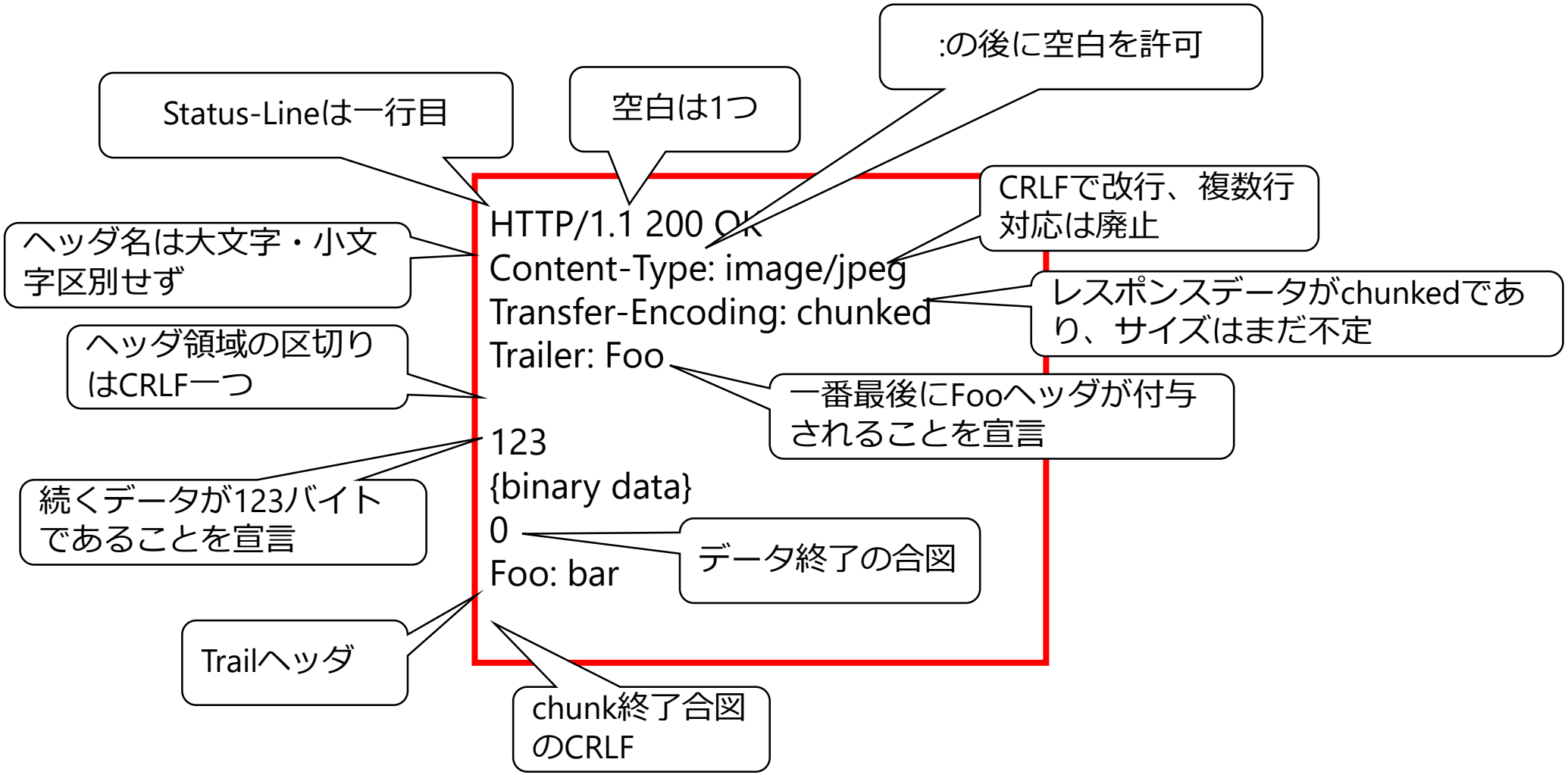
HTTP/1.1は非効率なプロトコル



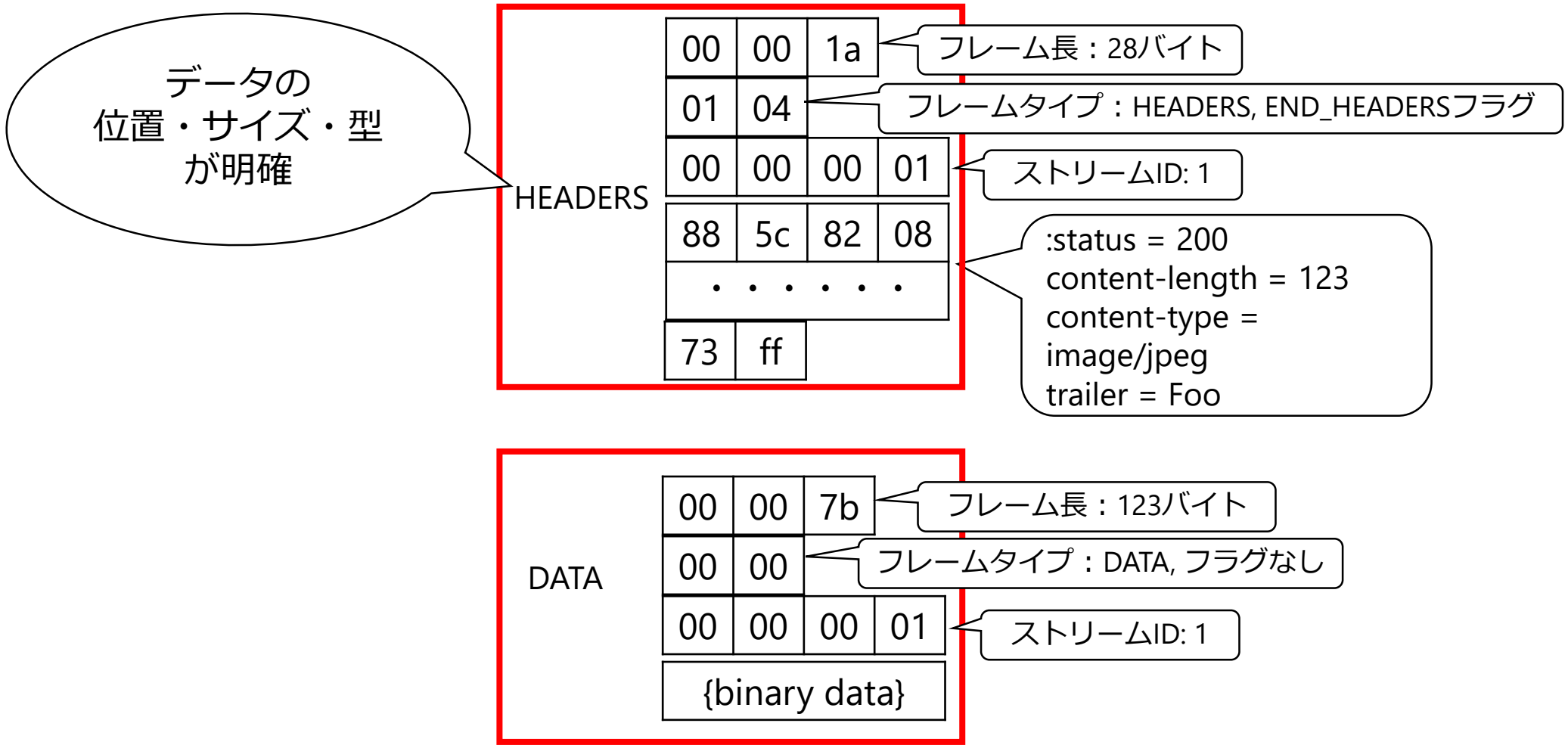
HTTP/2(SPDY)は効率的なプロトコル



3. HTTP/1.1は処理が煩雑なテキストプロトコル



HTTP/2はきっちりしたバイナリープロトコル



(* 記載スペースの都合上Trailer HEADERSは省いています)

HTTP/2の仕組み概要

HTTP/2の技術的な特徴

- HTTP/1.1のセマンティックスを変えない。
- サーバへのTCP接続数を１つに限定
- TLSと連携してプロトコルを自動選択
- バイナリープロトコル（テキストデータの曖昧さを排除）
- 全２重多重化通信
- フロー制御、優先度指定
- サーバプッシュ機能

HTTP/2の技術的な特徴

- SPDYのプロトコルアーキテクチャはそのまま利用
- SPDYの無駄なヘッダフィールドやフレームタイプを統廃合し、簡略化
- SPDYの実運用で明らかとなったフロー制御・優先度制御といった課題へ対応する
- TLS利用を前提とするSPDYに対し、平文接続も利用可能にする **（ただしほとんどのブラウザーは平文接続をサポートせず）**
- ヘッダ圧縮脆弱性(CRIME)対策として新しくHTTPに特化したヘッダ送受信仕様(**HPACK**)を策定する

HTTP/2初期ニゴシエーション

(1) TLS + ALPN

暗号化通信

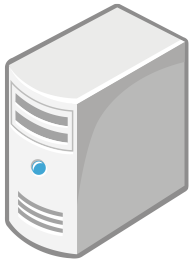
TLS接続時にALPN拡張フィールドを利用してHTTP/2に接続を行う。

現状のブラウザーは
TLS接続のみサポート

(2) HTTP Upgrade

平文通信

HTTP/1.1の接続後 Upgradeヘッダを使って、HTTP/2 に接続をアップグレードする。



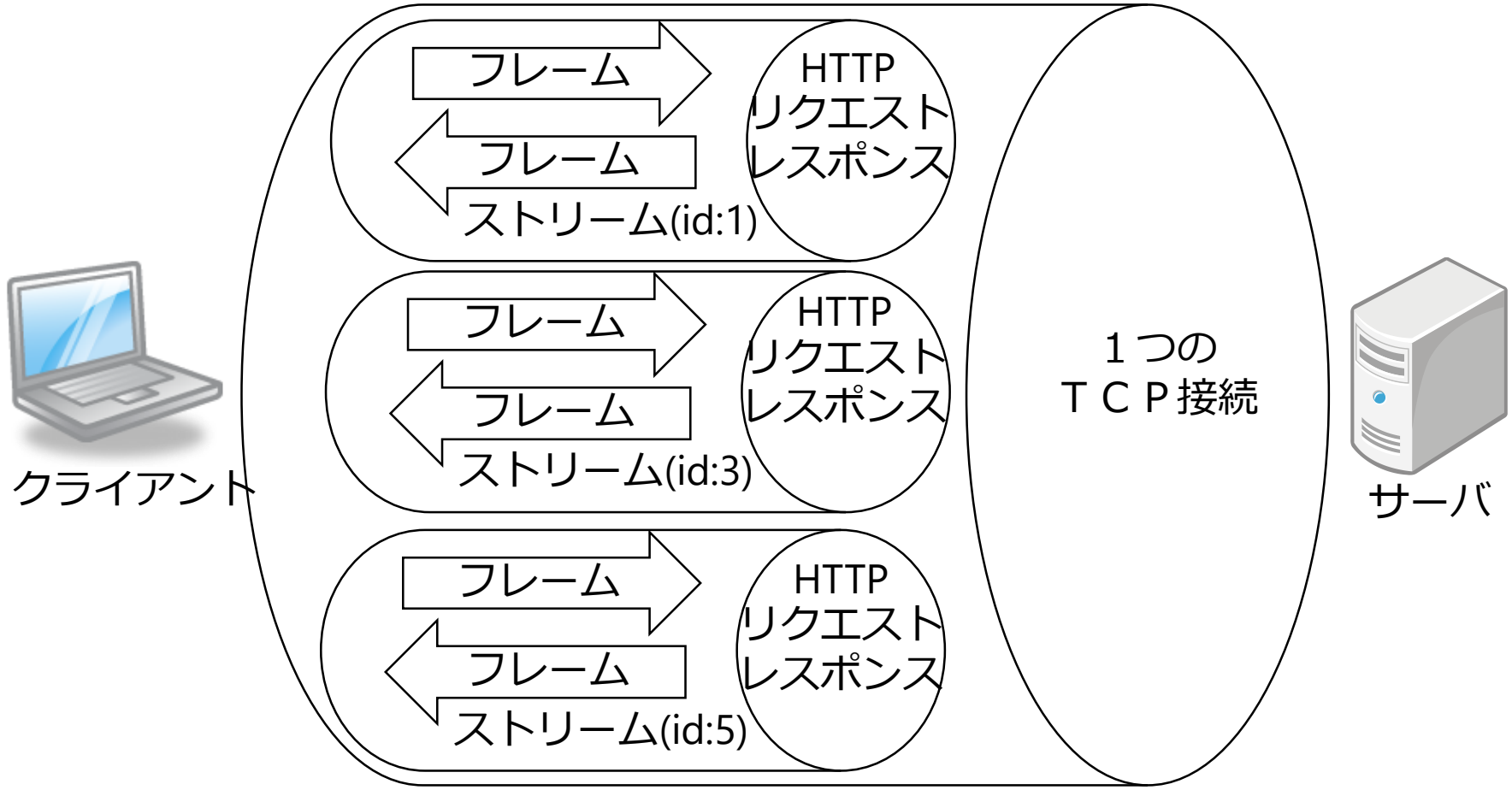
WebSocket
と一緒

(3) 事前知識によるDirect接続

あらかじめサーバがHTTP/2対応とわかっている場合、直接第２段階の接続方法を行う。
(DNSレコードや HTTPヘッダによるリダイレクト)

詳細仕様
検討中
ALT-SVC

HTTP/2の全二重多重化通信



仮想的なストリームチャンネルを生成して多重化を実現

HTTP/2のデータフレーム(binary)

デフォルトは14bit(16K),
24bit(16M)まで拡張可

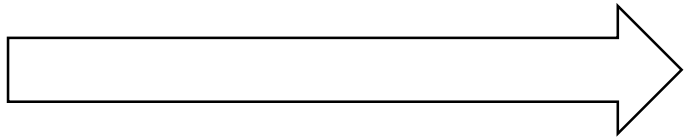
フレームヘッダ
9バイト

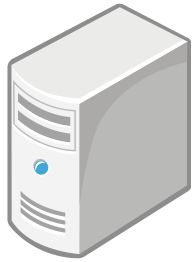
タイプ	フレーム種類	タイプ	フレーム種類
0x00	**DATA**	0x05	PUSH_PROMISE
0x01	**HEADERS**	0x06	PING
0x02	PRIORITY	0x07	GOAWAY
0x03	RST_STREAM	0x08	WINDOW_UPDATE
0x04	SETTINGS	0x09	CONTINUATION

HPACK

- HTTP/2で用いられるヘッダ圧縮技術
- もともとSPDYでは gzipを使ったヘッダ圧縮を行っていたが暗号化通信下でも情報が漏えい可能となるCRIME攻撃手法が公開され、クライアントからサーバへのヘッダ圧縮が無効になっていた。
- そこでHTTP/2専用でHTTPヘッダに特化した圧縮技術仕様HPACKの開発を行った。

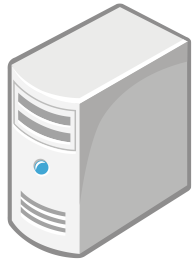
```
GET / HTTP/1.1
Host: www.google.co.jp
User-Agent: Mozilla/5.0 XXXXX
Accept: text/html,application/xhtml+xml,XXX
Accept-Language: ja,en-US;q=0.7,en;q=0.3
Accept-Encoding: gzip, deflate
Connection: keep-alive
```





HPACK概要

X番のエントリー(nameX, valueX)送信します。



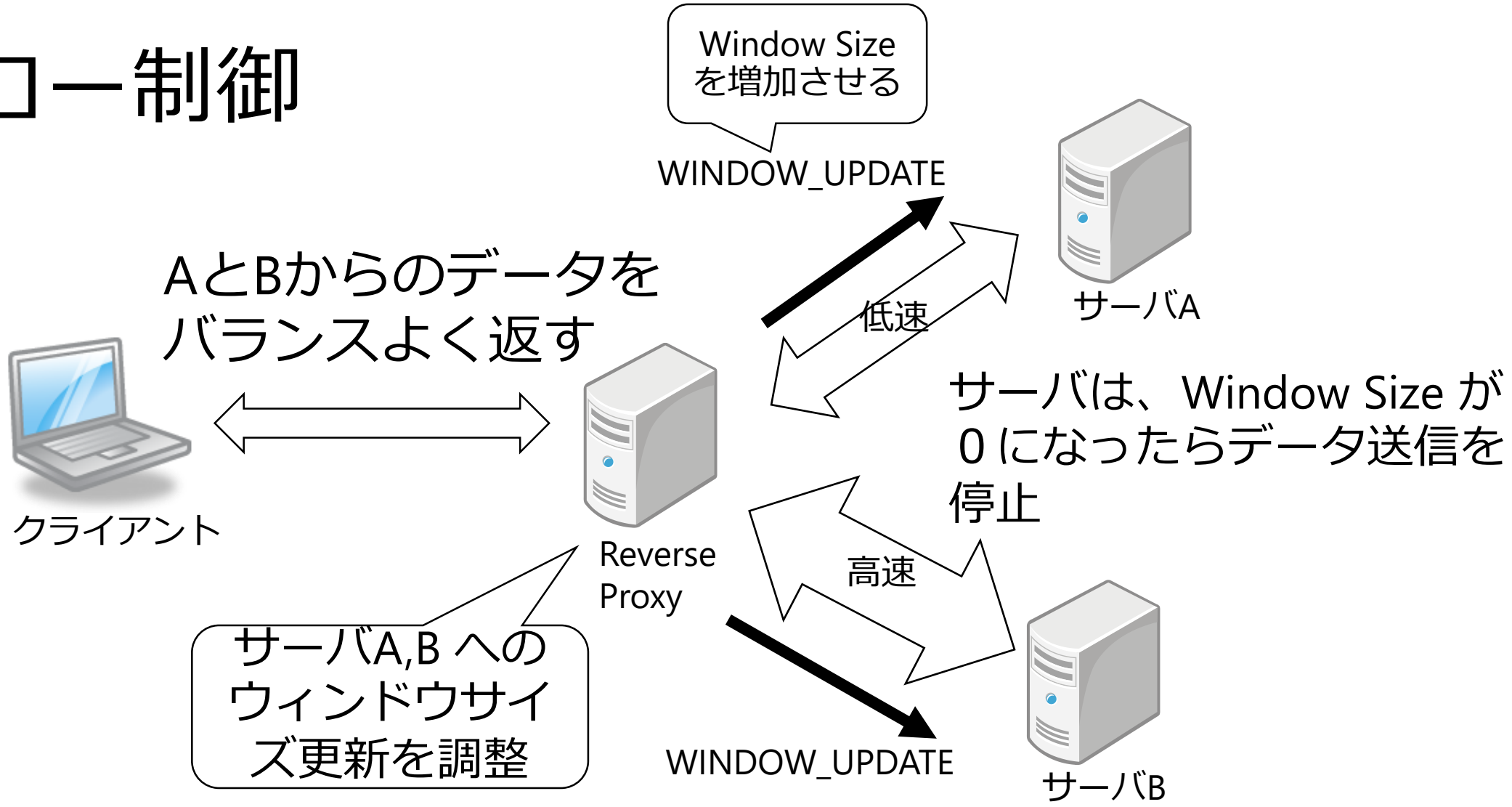
ヘッダテーブル
1. name1, value1
2. name2, value2
3. name3., value3
.....

X番のエントリーのnameXを使うけど値は別のvalueX'のヘッダを送信します。後で使うからテーブルに追加しておいて。

ヘッダテーブル
1. name1, value1
2. name2, value2
3. name3., value3
.....

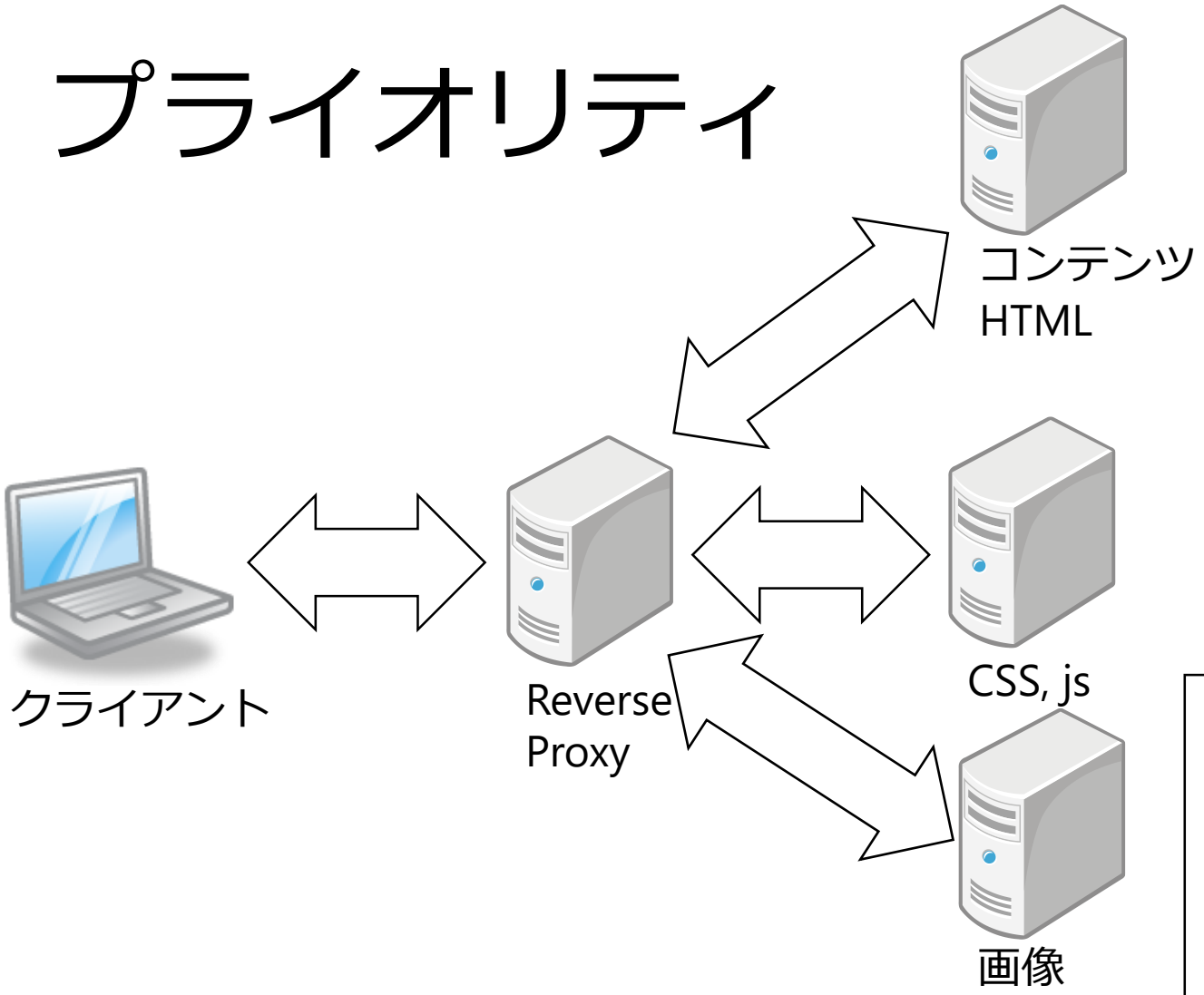
クライアント・サーバ両者でインデックス番号が付いたヘッダテーブルを保持。
両者でヘッダテーブルのインデックスを参照する情報を送信。
最も圧縮ができる時は、番号を通知するだけでよい。

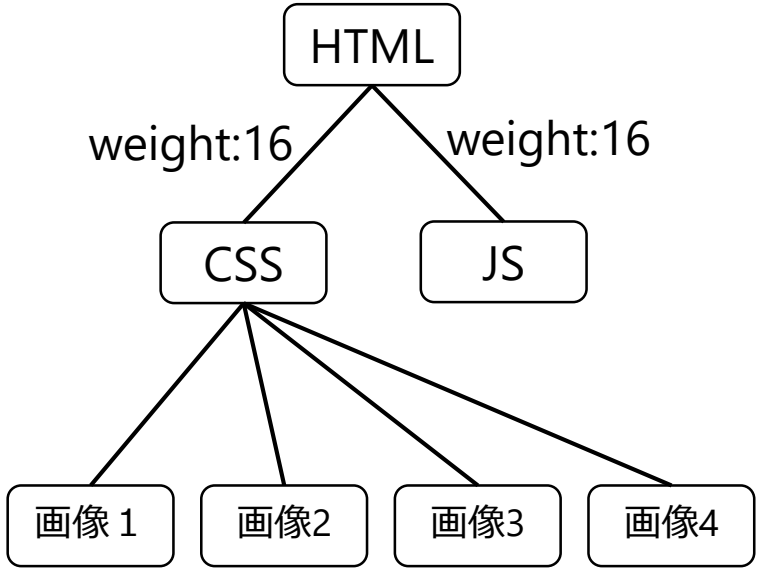
フロー制御



TCPコネクション、ストリーム毎にフロー制御が可能

プライオリティ





依存性と重みを指定

プライオリティのユースケース

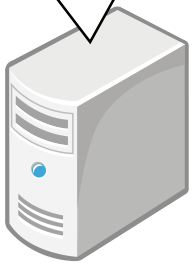
- ファイルタイプ(HTML/CSS/JS/画像）に応じた返答順序の指定
- タブ切り替えによる重みの上げ下げ
- 分割されたビデオデータなど順番が明示的に決められている場合

サーバプッシュ機能

予約された画像リクエストはクライアントからサーバに送らずに，クライアントはサーバ側からの画像データの送付を待つ

サーバはコンテンツの中身を判断し，あらかじめコンテンツに含まれている画像の**リクエストを予約**する．

クライアント



サーバ

コンテンツリクエスト

画像のHTTPリクエストを予約

コンテンツのレスポンス

画像データ

サーバから送信された画像データは，クライアントのキャッシュに保存

キャッシュ

デモ

- HTTP HoLの有無によるHTTP/1.1(SSL)とHTTP/2の見え方の違いをデモします。
- 少数の画像 vs 多数の画像
- 表示に遅延あり vs 遅延なし

即表示

遅延ありの場合リクエストの3秒後に表示されます

HTTP/2からQUICへ

TCPの問題点

- TCP Head of Line Blocking.
- 3方向ハンドシェイクのコスト.
- initcwnd（初期輻輳ウィンドウ)が小さくスロースタート
- パケットロスによる大きなバックオフ（待ち時間）
- カーネルのソケットバッファの増大
- NATのタイムアウトとIPのローミング
- 経路途中のTCP Buffer Bloat

TCP Fast Open

initCWND10

TCP cubic

TCP_NOT_SENT_LOWAT

Random packet drop in router

解決策は提示されているが、OSや中継機の機能追加が必要
これは非常に時間がかかる。

TLSの問題点

- TLSのハンドシェイクネゴシエーションのコスト
- ChangeCipherSpecが来るまで待たないといけない
- 負荷分散のバグによるClientHelloのサイズ制限
- サーバから送信する証明書チェーンの肥大
- 再接続や再ネゴシエーションはコストがかかり最適化できてない

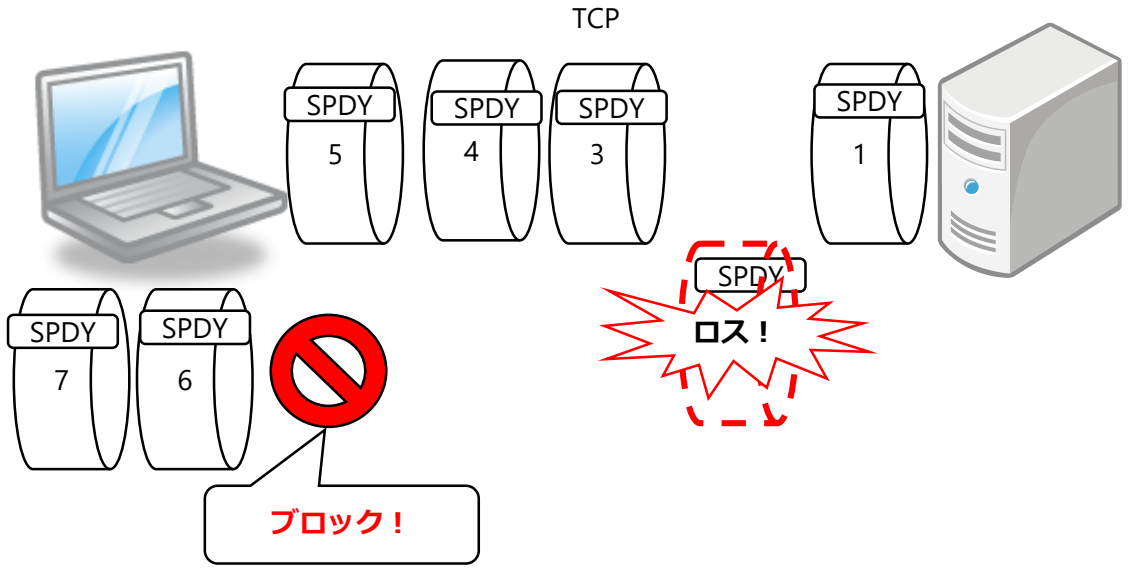
TLS False Start

Ballooning Extension

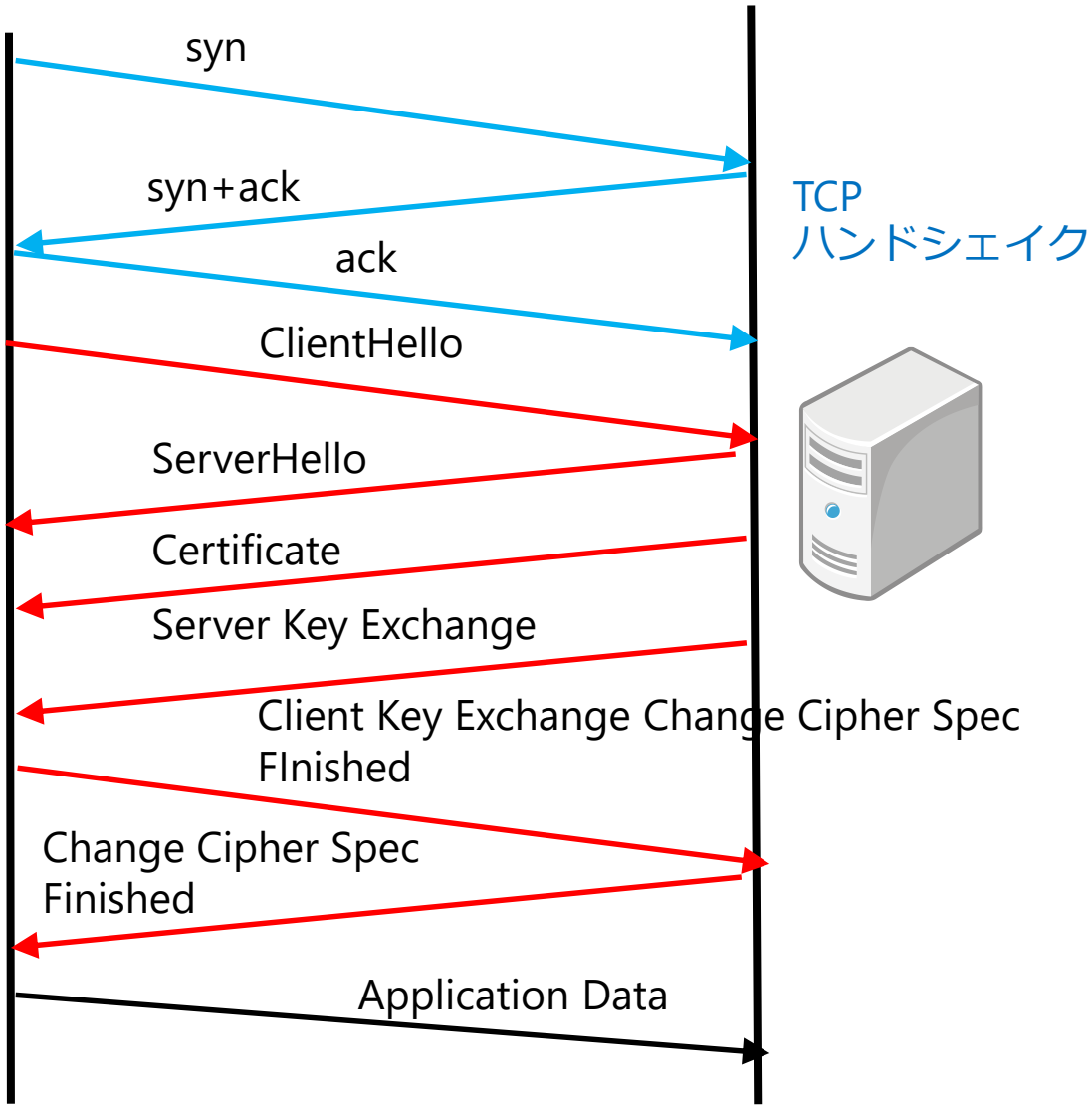
TLS Ticket, Channel ID

TLSのライブラリや中継機のバージョンアップが必要

TCP HoLブロックとTCP+TLSハンドシェイク

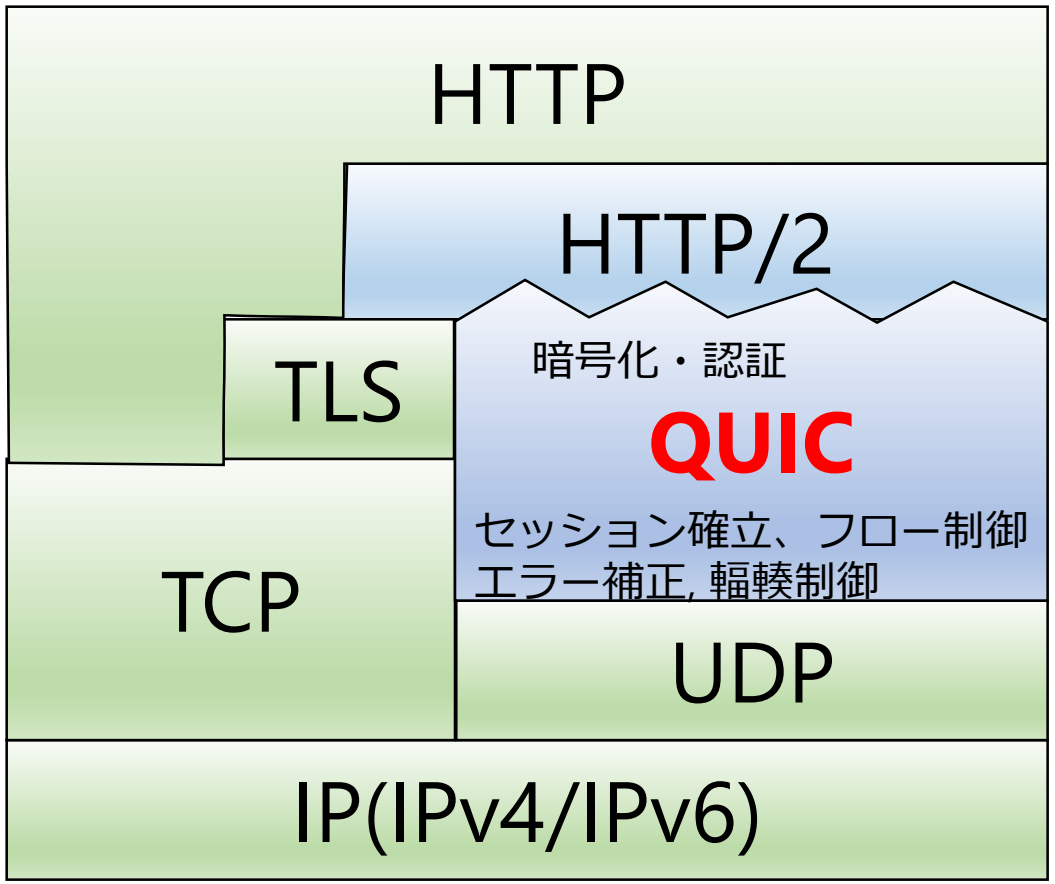


TLS
ハンドシェイク



QUIC(**Q**uic **U**DP **I**nternet **C**onnections)

- Googleが独自に開発しているプロトコル
- UDP上でTCP+TLS相当＋αの機能を実装
  - 最短0-RTTで再接続、暗号通信を必須化
  - TCPと同様の輻輳制御で帯域の公平化
  - 独自の誤り訂正と再送機能でパケットロスによるTCPのHead of Line Blockingを回避
  - HTTP/2のフレーム制御機能を取り込み
- Googleの全サービスで運用中

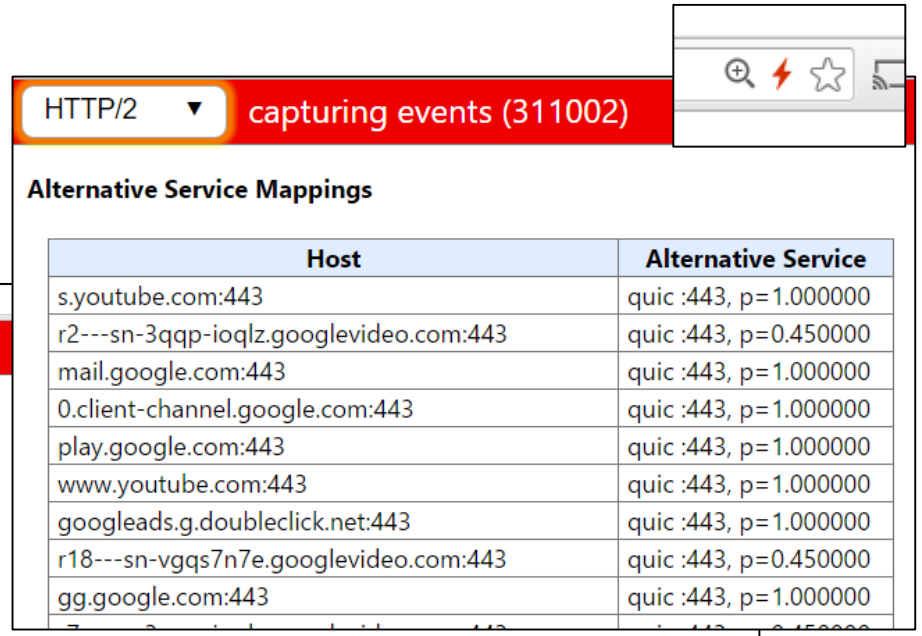


QUICによる性能向上結果(Google発表分)

- 92%でQUICが利用できている(デスクトップ＋モバイル環境)
- 平均5%のページ読み込み時間の短縮
- 速度下位1%の接続で1秒の短縮 (接続環境が悪い程効果が高い)
- 最適化しているGoogle検索ページでも平均3%の読み込み時間の短縮
- Youtubeで rebuffer を30%短縮（ビデオ停止時間が短い）
- 75%が 0-RTTの恩恵を受けている(QUICによる性能向上効果の半分が0-RTT)

データ参照先 http://blog.chromium.org/2015/04/a-quic-update-on-googles-experimental.html
https://docs.google.com/presentation/d/15e1bLKYeN56GL1oTJSF9OZiUsI-rcxisLo9dEyDkWQs/edit?usp=sharing

ChromeでQUICを確認



HTTP/2 ▼ capturing events (311002)

Alternative Service Mappings

Host	Alternative Service
s.youtube.com:443	quic :443, p=1.000000
r2---sn-3qqp-ioqlz.googlevideo.com:443	quic :443, p=0.450000
mail.google.com:443	quic :443, p=1.000000
0.client-channel.google.com:443	quic :443, p=1.000000
play.google.com:443	quic :443, p=1.000000
www.youtube.com:443	quic :443, p=1.000000
googleads.g.doubleclick.net:443	quic :443, p=1.000000
r18---sn-vgqs7n7e.googlevideo.com:443	quic :443, p=0.450000
gg.google.com:443	quic :443, p=1.000000

chrome://net-internals/#quic

QUIC ▼ capturing events (134270)

- QUIC Enabled: true
- Alternative Service Probability Threshold: undefined
- Origin To Force QUIC On: :0
- QUIC connection options:
- Consistent Port Selection Enabled: false

QUIC sessions

View live QUIC sessions

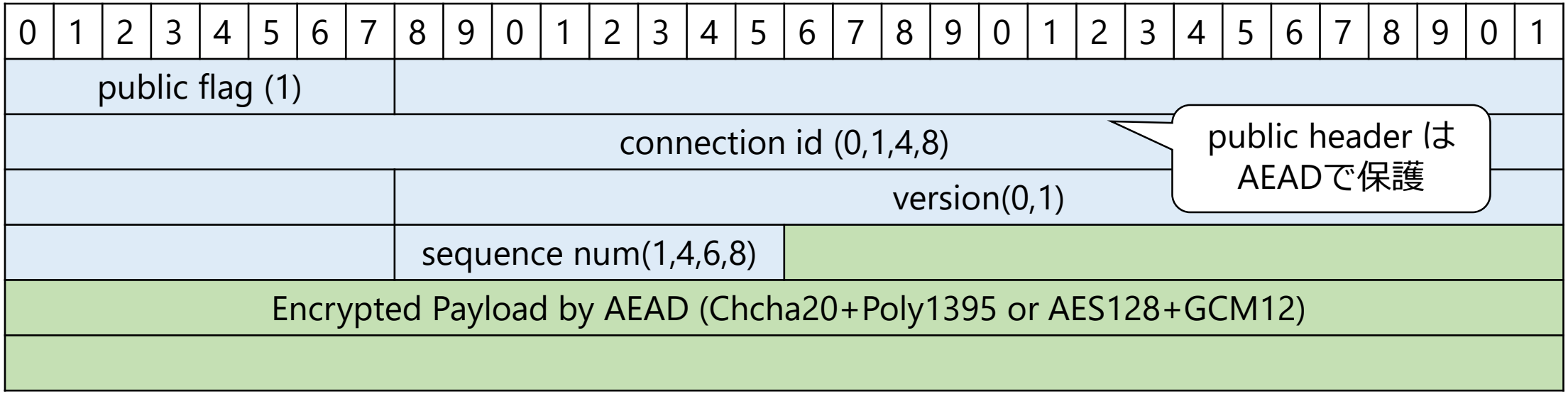
Host	Secure	Version	Peer address	Connection UID	Active stream count	Active streams	Total stream count	Packets Sent	Packets Lost	Packets Received	Connected
0.client-channel.google.com:443	true	QUIC_VERSION_27	64.233.188.189:443	8474098053639222975	1	15	8	37	0	40	true
clients4.google.com:443	true	QUIC_VERSION_27	216.58.220.238:443	10242948267502299464	0	None	1	7	0	7	true
clients6.google.com:443 gg.google.com:443	true	QUIC_VERSION_27	216.58.220.238:443	11374158713995592306	0	None	0	4	0	5	true
csi.gstatic.com:443	true	QUIC_VERSION_27	64.233.184.94:443	1723919714625107115	0	None	0	2	0	2	true
googleads.g.doubleclick.net:443	true	QUIC_VERSION_27	173.194.117.237:443	11997643291256147053	0	None	5	15	0	15	true
i.ytimg.com:443 s.ytimg.com:443	true	QUIC_VERSION_27	216.58.220.238:443	12219795968108955823	0	None	19	106	0	174	true
pagead2.googlesyndication.com:443	true	QUIC_VERSION_27	173.194.117.250:443	14793345242681170653	0	None	5	89	0	164	true
pubads.g.doubleclick.net:443	true	QUIC_VERSION_27	216.58.220.226:443	10960468474006060118	0	None	1	6	0	5	true
r13---sn-vgqs7n7l.googlevideo.com:443	true	QUIC_VERSION_27	173.194.133.50:443	2724579964172757350	0	None	1	6	0	7	true
r2---sn-3qqp-ioqlz.googlevideo.com:443	true	QUIC_VERSION_27	27.80.250.205:443	50675375175223475	0	None	4	76	0	147	true
r7---sn-3qqp-ioqlz.googlevideo.com:443	true	QUIC_VERSION_27	27.80.250.210:443	585134865502118168	0	None	21	6033	0	12132	true
s.youtube.com:443	true	QUIC_VERSION_27	173.194.117.227:443	8342181436207278967	0	None	9	22	0	22	true
tpc.googlesyndication.com:443	true	QUIC_VERSION_27	216.58.220.225:443	9080136759649470080	0	None	1	6	0	8	true
www.google.com:443	true	QUIC_VERSION_27	173.194.120.80:443	2009475195212577496	0	None	3	41	0	77	true
www.googleapis.com:443	true	QUIC_VERSION_27	216.58.220.234:443	13554648342023425740	0	None	0	2	0	2	true
www.youtube.com:443	true	QUIC_VERSION_27	173.194.120.73:443	5759212590014597285	0	None	10	58	0	91	true
yt3.ggpht.com:443	true	QUIC_VERSION_27	173.194.120.75:443	17763280540468029064	0	None	1	4	0	5	true

QUICの実装

- Chrome/GFE
  - テスト用のサーバ・クライアントが付随
  - 最近急速にバージョンアップを繰り返し、現在 QUIC_VERSION_30
- Microsoftのプロトタイプ(*1)
  - C++で3000行程度。非公開。
- libquic, goquic https://github.com/devsisters/libquic, goquic
  - libquic:Chromeの net/quic以下を外部依存を覗いてまとめたもの
  - goquic: libquicとwrapper クラスを cgo でバインディングしたもの
  - QUIC_VERSION_24までしか対応してない。
  - forkして QUIC_VERSION_30まで対応しました。
    https://github.com/shigeki/libquic/tree/QUIC_VERSION_30

(*1 https://docs.google.com/presentation/d/1BjPUowoOoG0ywmq5r8QNqnC9JPELUe02jvgyoOW3HFw/edit?usp=sharing)

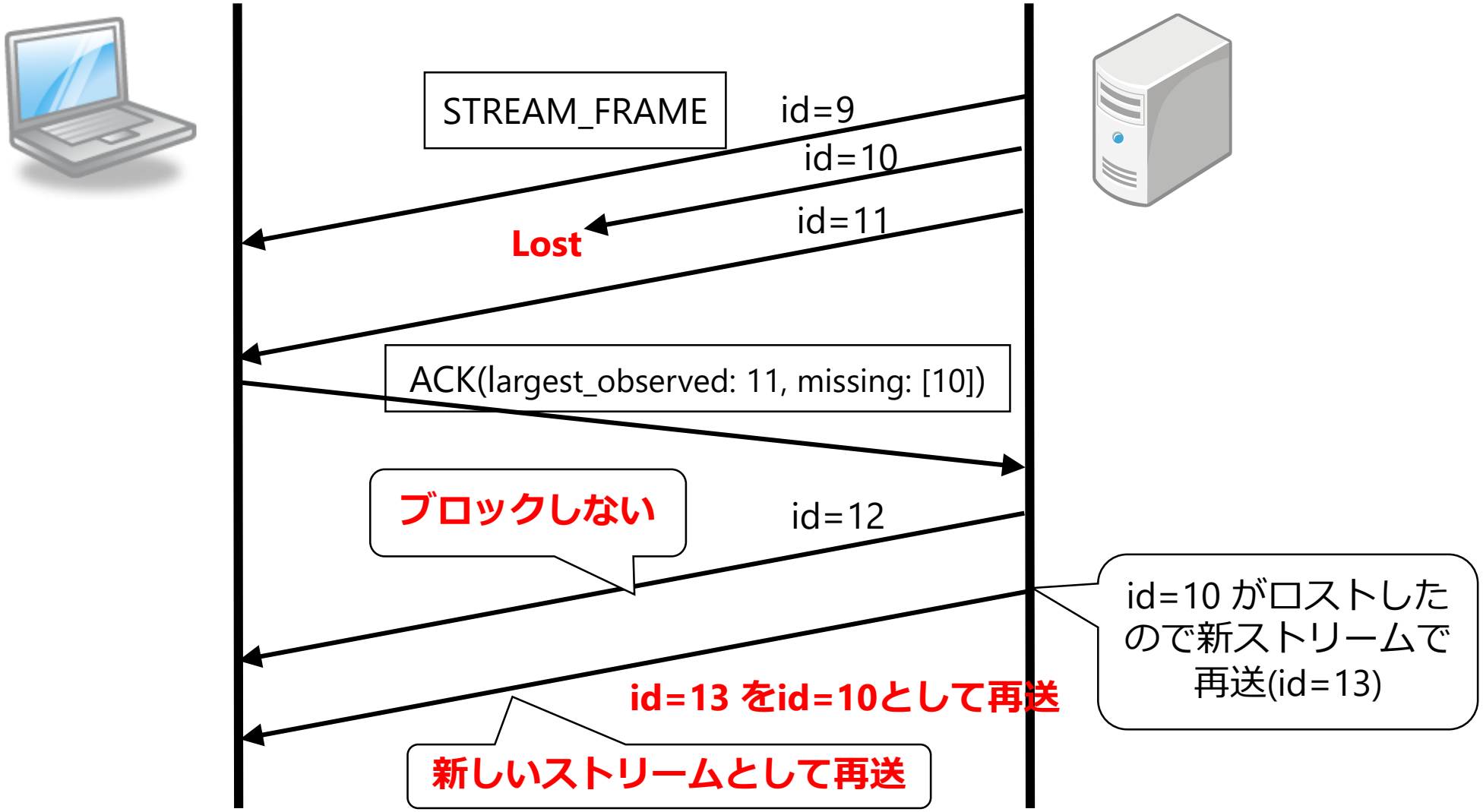
QUIC Wireフレーム書式(Public Header)



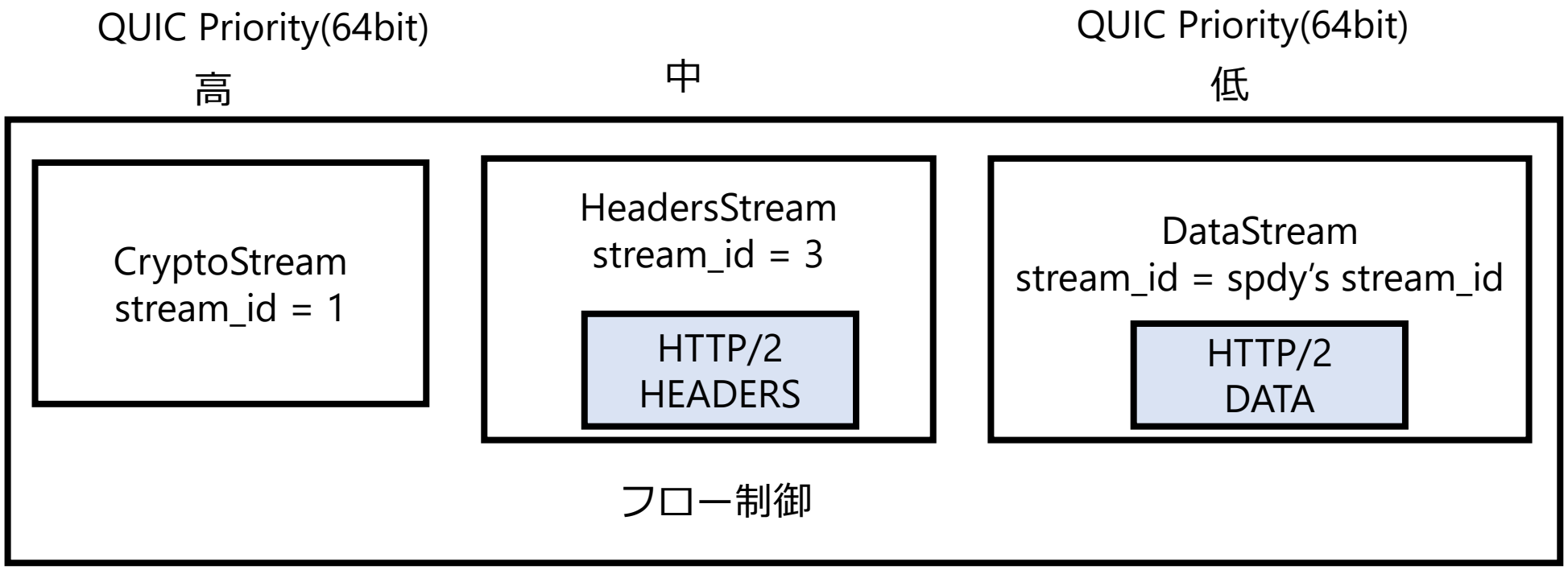
クライアント固有のConnectionIDベースでセッション管理

src ip/port が変更されてもセッションが継続できる。

欠損フレームを新しいストリームで再送

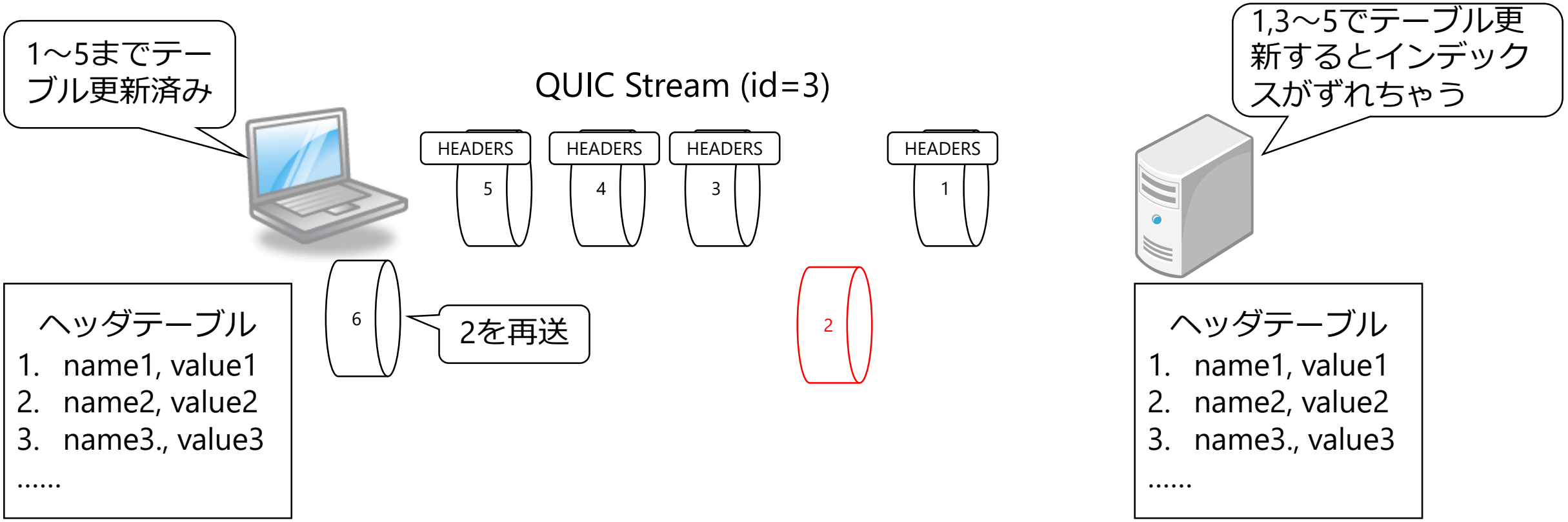


QuicストリームとHTTP/2バインディング



Stream 1 は、暗号化ハンドシェイク専用ストリーム
Stream 3 は、ヘッダ情報のやりとり専用ストリーム

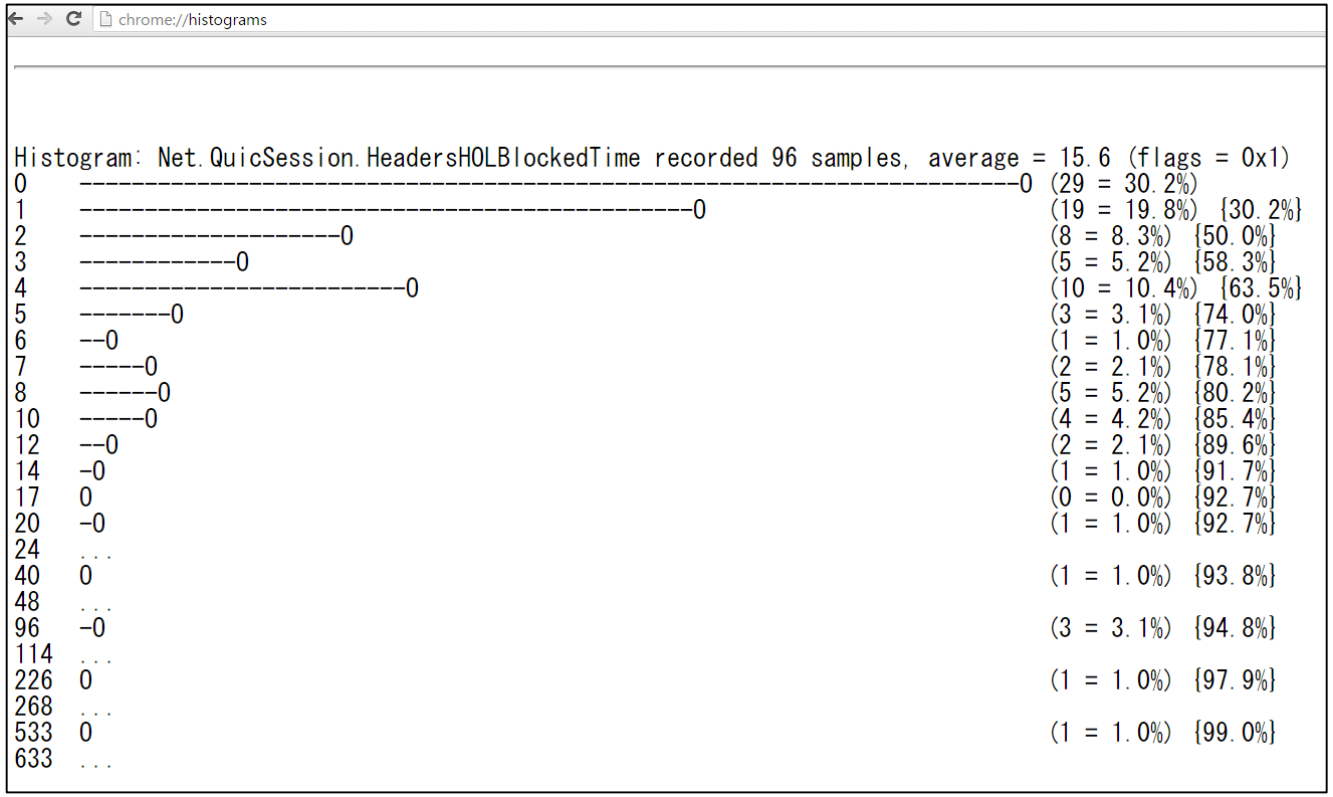
QUICの問題点:
HPACKによるQUICのHoLブロック



HEADERフレームの欠損によるヘッダテーブルの不一致を防ぐには欠損したフレーム(6)が届くまでヘッダテーブルの更新をブロックするしかない。

QUICの問題点：
HPACKによるQUICのHoLブロック

最近の Canary では chrome://histograms からHPACK HoLの状況を確認できます。



QUICのせいでWebの閲覧速度が遅いという感じがするなら一度この値を確認してください。

デモ

TCP HoL耐性デモ

- packet ロストで HTTP/2 vs QUIC の通信がどう変わるかデモします。
- HTTP/2の方は TCP HoLで通信が途絶えますが、QUICは独自再送機能で通信は継続されます。

0-RTT実演デモ　（時間があれば）

- HTTP/2 over TLS, QUIC (0-RTTを disabled), QUIC(0-RTT)で初期接続時間の比較を行います。

QUICからTLS1.3へ

注意： 内容は2015年11月2日時点での TLS1.3 draft (*)を元にしています。今後の仕様策定作業で本プレゼンの内容が変更になる場合がありますのでご注意ください。

(* https://github.com/tlswg/tls13-spec/ の master HEAD)

TLS1.3の目的

1. ハンドシェイクデータをできるだけ暗号化して隠匿する
2. ハンドシェイクレイテンシーの削減
   - 再接続の場合は、 0-RTT
   - フルハンドシェイクは、 1-RTT
3. ハンドシェイクで交換する項目の見直し、簡素化
4. レコード層暗号化の見直し(RC4廃止、CBC不採用等)

TLS1.3の特徴

- 様々な機能・項目を見直し・廃止
  - 時代に合わなくなったもの、より効率的な仕組みに変更修正されたものなど、TLS1.2の機能・項目を数多く廃止。（後述）
- よりセキュアに
  - 平文通信が必要な部分を極力少なくして情報を隠匿。
  - AEAD必須やパディング等将来的な攻撃に備える。
- 性能向上
  - QUICで使われているような 0-RTTの接続をサポートし、接続レイテンシーを下げられるようにしている。将来的にQUICはTLS1.3をサポート(*)する方向。

(* QUICとTLSでフレームフォーマットが異なるので全く同一にならない可能性があります。)

TLS1.3の廃止項目一覧とその理由

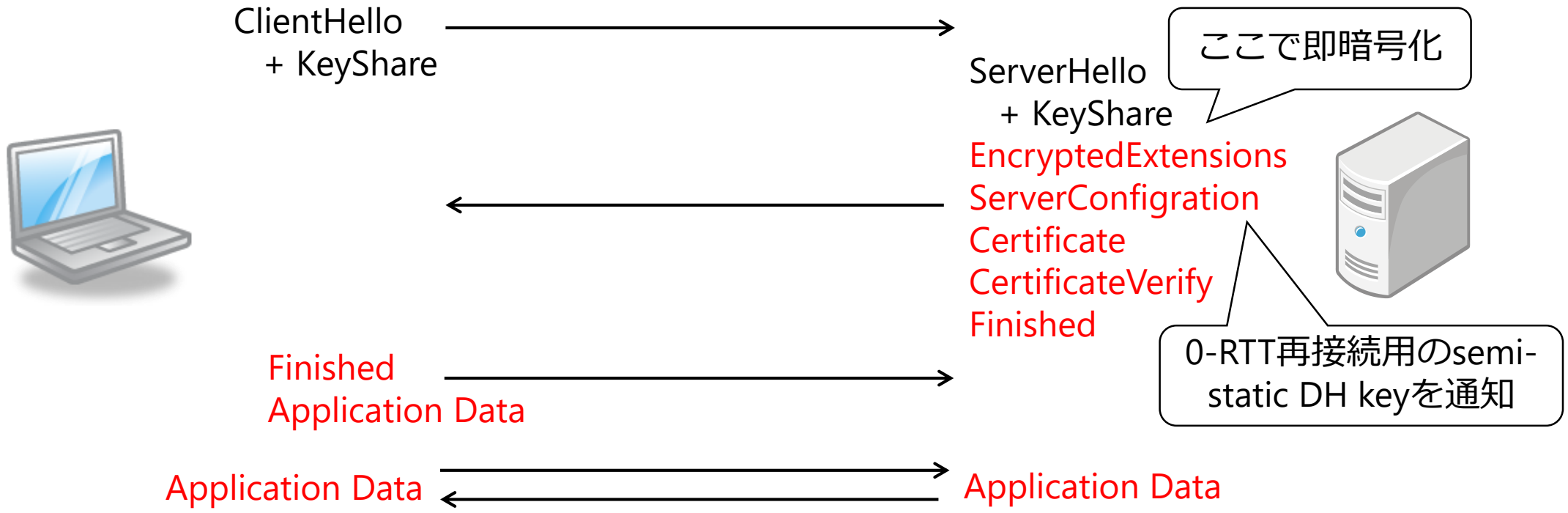
これまでの機能の必要性を全面的に見直し

機能	Compression	CRIME/TIME等の圧縮タイミング攻撃対策
	Renegotiation	Triple Handshake等Renegotiation前後の状態引継ぎの不備を対策 廃止に伴いクライアント認証とRekeyを行う方式を変更
	SessionID resumption	PSKと同一方法で resumptionが可能になったため
暗号署名方式	MD5, SHA-1, SHA-224	危殆化、低強度対策
	DSA	利用用途の減少
	non-AEAD(CBC, RC4)	RC4危殆化、MtEを狙ったパディングオラクル攻撃の対策
	IVフィールド	AEADのみになったため nonce は seq no から生成
	AEADのadditional data	要検証なレコードヘッダ部はnonce/暗号データ内で改ざん検知可能
	custom DHE	Cross-Protocol対策で named groupに統一
	ECDHE compress format	uncompress書式だけで利用用途が十分
	anonymous DH	raw public key (RFC7250)を使うか、証明書の検証をしない

TLS1.3での廃止項目一覧とその理由 cont'd

ハンドシェイク書式	record layerのバージョン	実質的に利用用途がないため。後方互換だけのために残されている。
	random中のgmt unix time	乱数生成が高度化され利用用途がなくなったため
	PFS/PSK以外の鍵交換	秘密鍵の危殆化対策
	SSL2.0/3.0のサポート	危殆化プロトコルのサポート廃止
	不要タイプの削除	ChangeCipherSpec,HelloRequest,Sever/ClientKeyExchange等(後述)
拡張	max_fragment_length	最大値は2^14 に固定
	truncated_hmac	AEADに利用できないし、安全でないため。
	srp	PSKに置き換え？
	encrypt_then_mac	AEAD利用のため
	extended_master_secret	TLS1.3仕様に取り込み
	SessionTicket	Resumption/PSKで利用するTicketに置き換えるため
	renegotiation_info	Renegotiationを禁止

TLS1.3 ハンドシェイク (full handshake)



1-RTT

注意： 内容は2015年11月2日時点での
TLS1.3 draftを元にしています。

(赤文字は暗号化されているデータ)

TLS1.3 ハンドシェイク (0-RTT再接続)

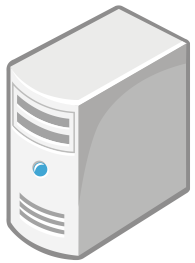
フライイングで
アプリデータ暗
号化して送る

semi-static
server DH key

以前の接続で保持

ClientHello
+ KeyShare
+ EarlyDataIndication
EncryptedExtensions
ApplicationData

ServerHello
+ KeyShare
+ EarlyDataIndication
EncryptedExtensions
ServerConfigration
Certificate
CertificateVerify
Finished



本来必要ないが口ジック
の簡略化のため常時署名

Finished
Application Data

Application Data

0-RTT

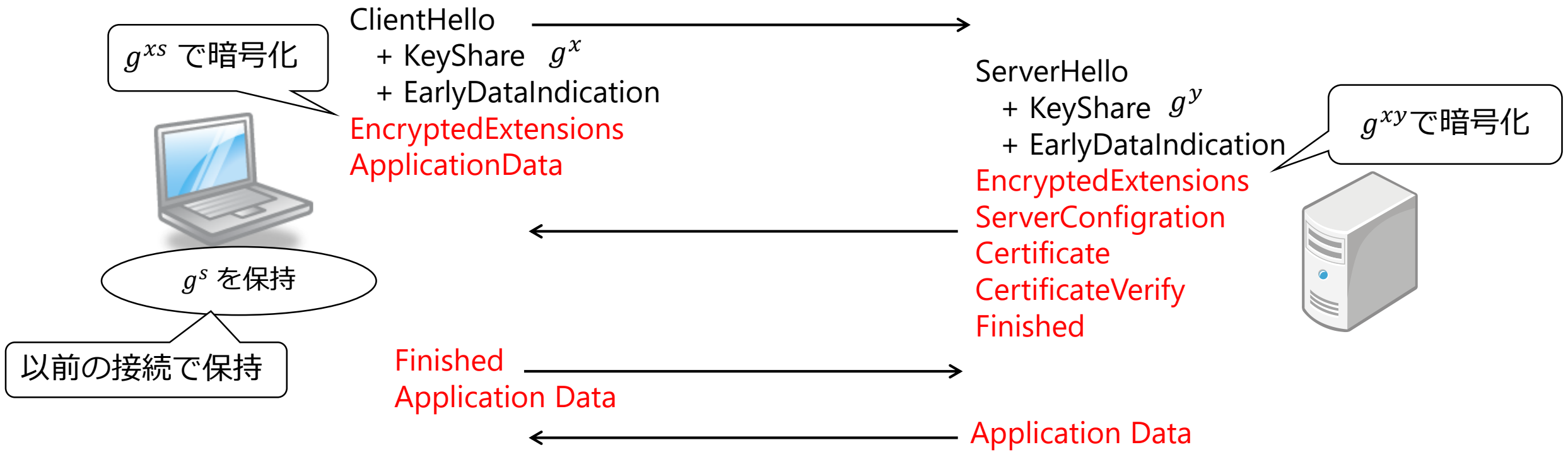
注意： 内容は2015年11月2日時点での
TLS1.3 draftを元にしています。

(赤文字は暗号化されているデータ)

TLS1.3の新たな鍵交換方式と鍵スケジュール (OPTLSの採用)

- signature based handshake → DH based handshake にする提案
- 当初 DH証明書と公開鍵の offline signature(後述)の利用を想定していたため反対意見が多かった。
- プロトコルロジックや鍵生成が簡略化され、PSKの導入もしやすくなるのでTLS1.3に採用する方針となった。
- しかし offline signature機能を外し、online署名された semi-static DH鍵を利用することにした。(offline署名については後述)

TLS1.3の鍵交換(0-RTT)

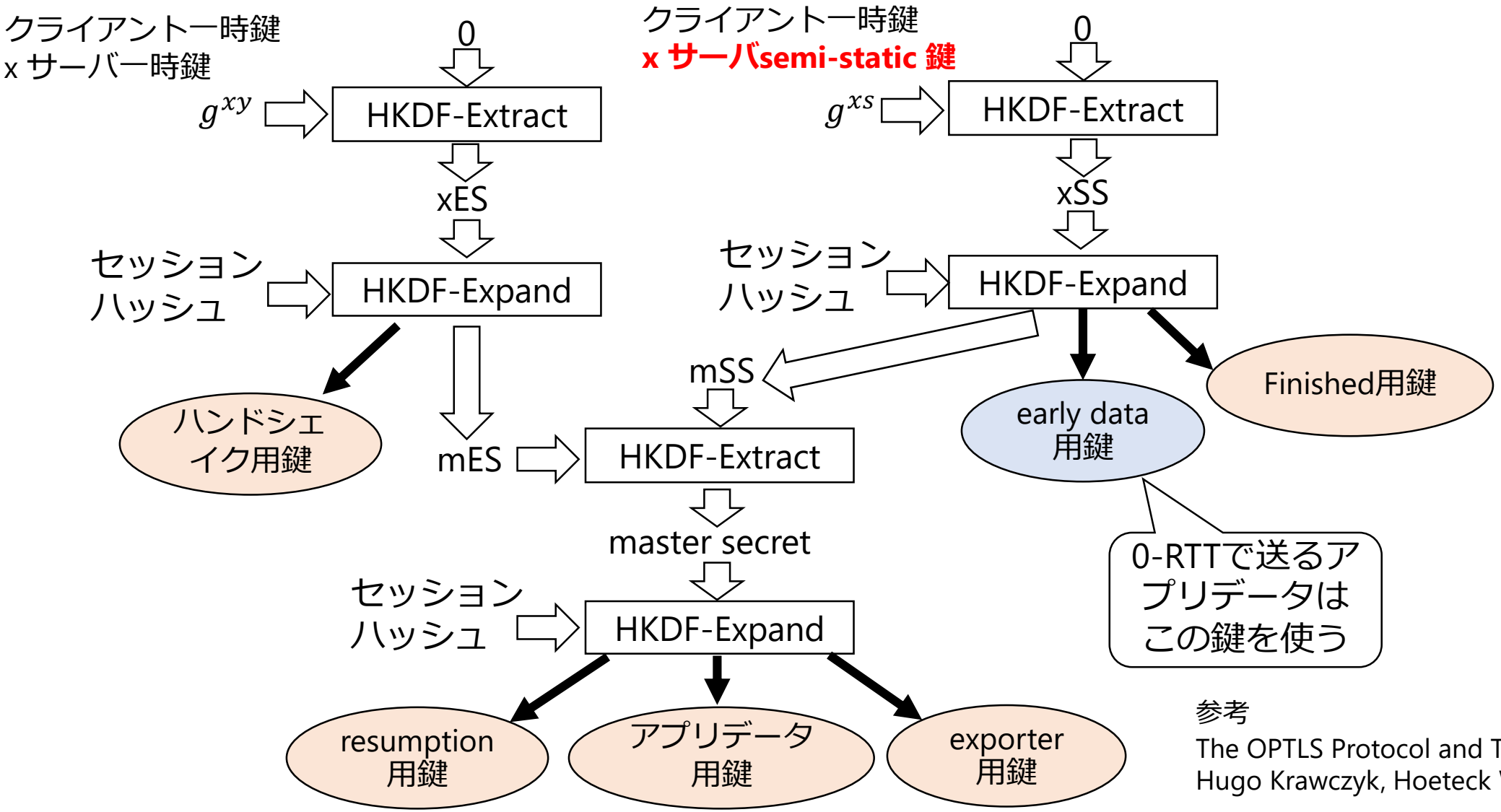


(赤文字は暗号化されているデータ)

g^x: クライアント一時鍵
g^y: サーバ一時鍵
g^s: サーバ semi-static 鍵

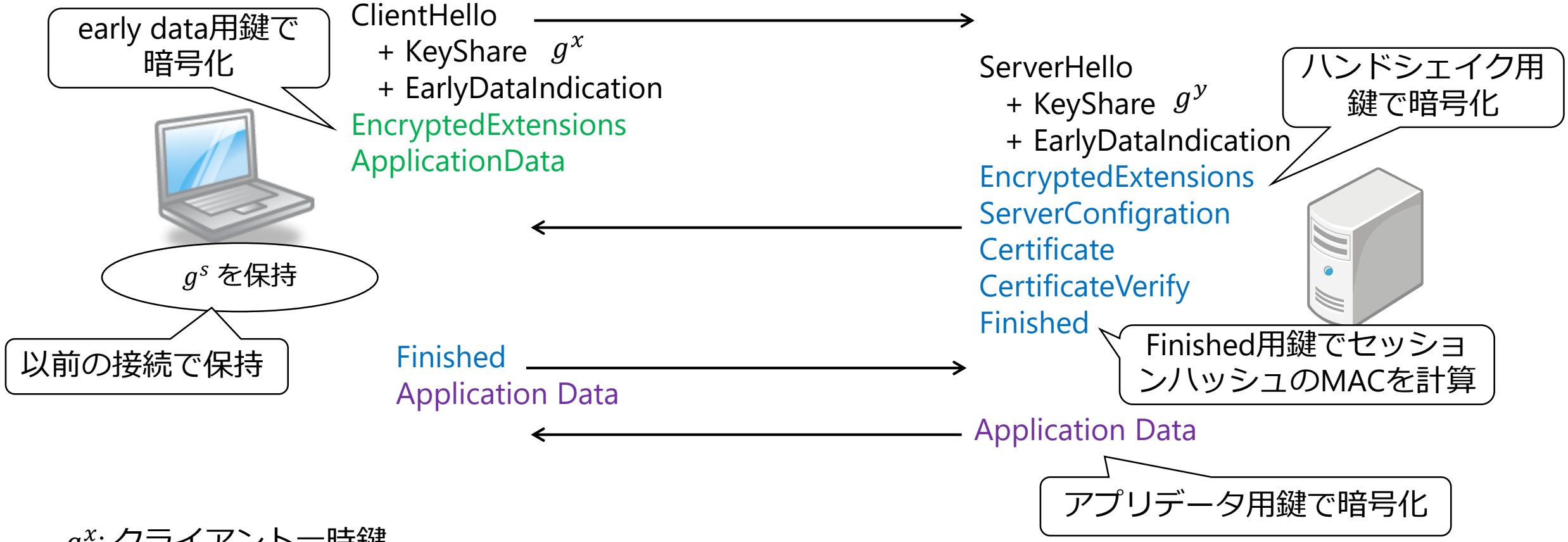
注意： 内容は2015年11月2日時点での
TLS1.3 draftを元にしています。

TLS1.3鍵スケジュール(0-RTT)



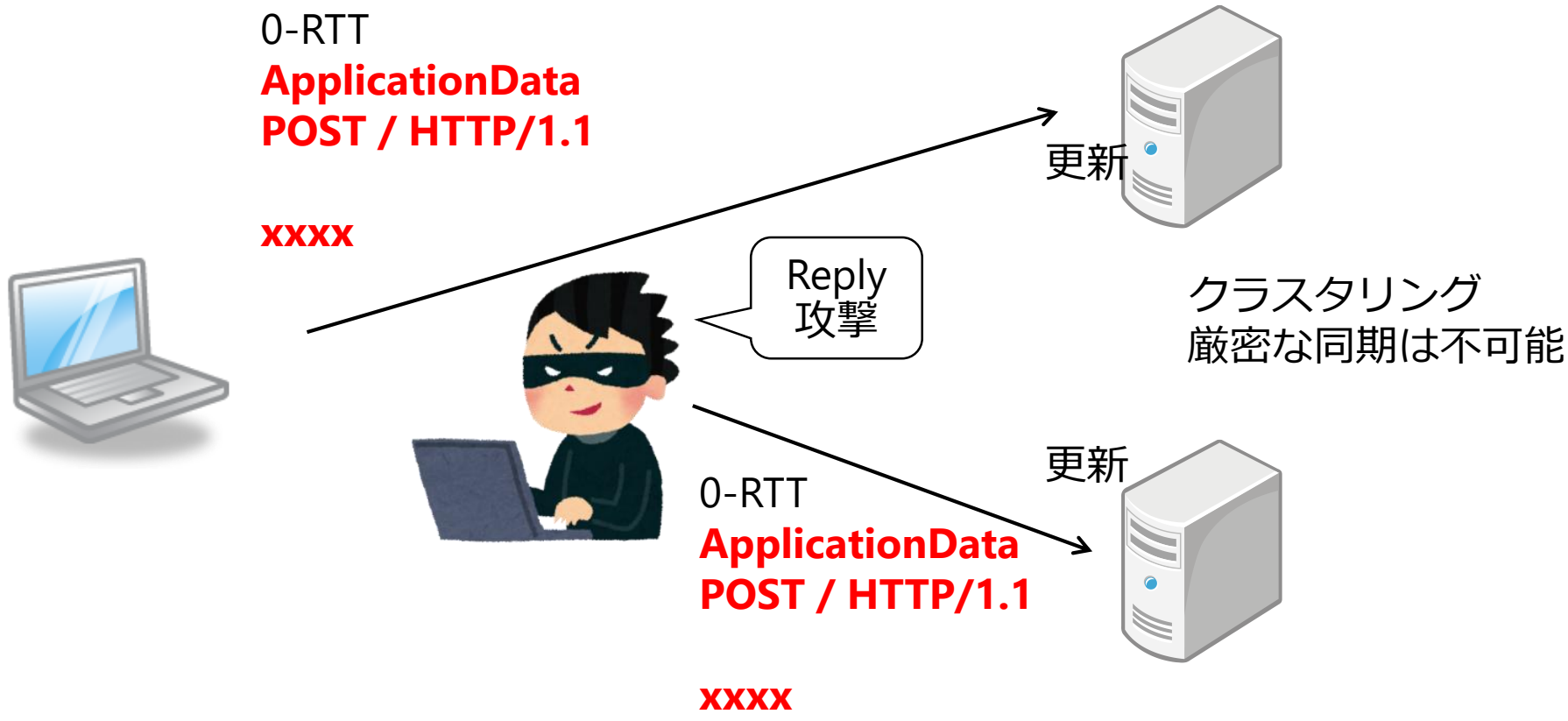
参考
The OPTLS Protocol and TLS 1.3
Hugo Krawczyk, Hoeteck Wee 2015

0-RTTでは4種類の鍵を利用



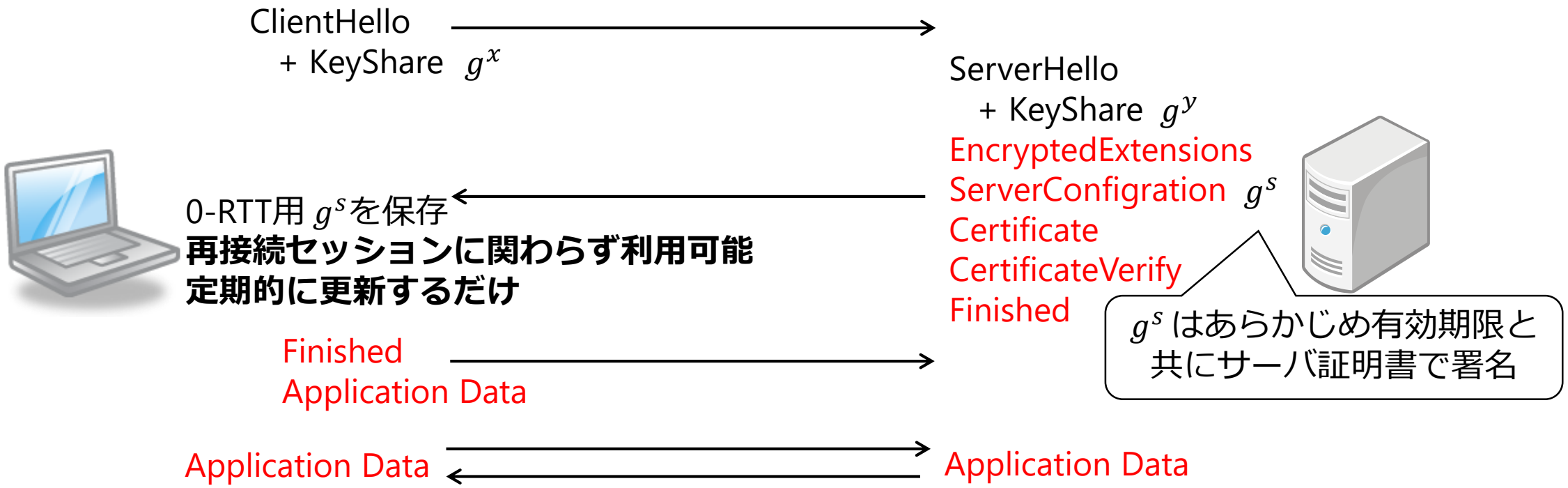
g^x: クライアント一時鍵
g^y: サーバ一時鍵
g^s: サーバ semi-static 鍵

TLS1.3 0-RTTの問題点



Reply攻撃に弱いため、最初のデータは冪等性のあるリクエストのみに制限しなければならない

TLS1.3のその先(offline signature)



TLS1.3の採用は見送られたが並行して拡張仕様として検討を進める方向に

まとめ

- HTTP/2, QUIC は Web閲覧の高速化を図る新しいプロトコルです
- HTTP/2は HTTP/1.1の HoLブロックの解消を実現しました。
- QUICは、TCPの HoLブロックの解消し、暗号化ハンドシェイクの0-RTTを実現しました。
- TLS1.3は、古くて無駄な項目を数多く廃止し、よりセキュアな暗号通信を実現します。また0-RTTで接続レイテンシーの向上を目指します。
- 常時TLS接続の世界が予想される中、将来的にTLS1.3の元でHTTP/2や次のバージョン、QUIC等のWebプロトコルが実現されるでしょう。