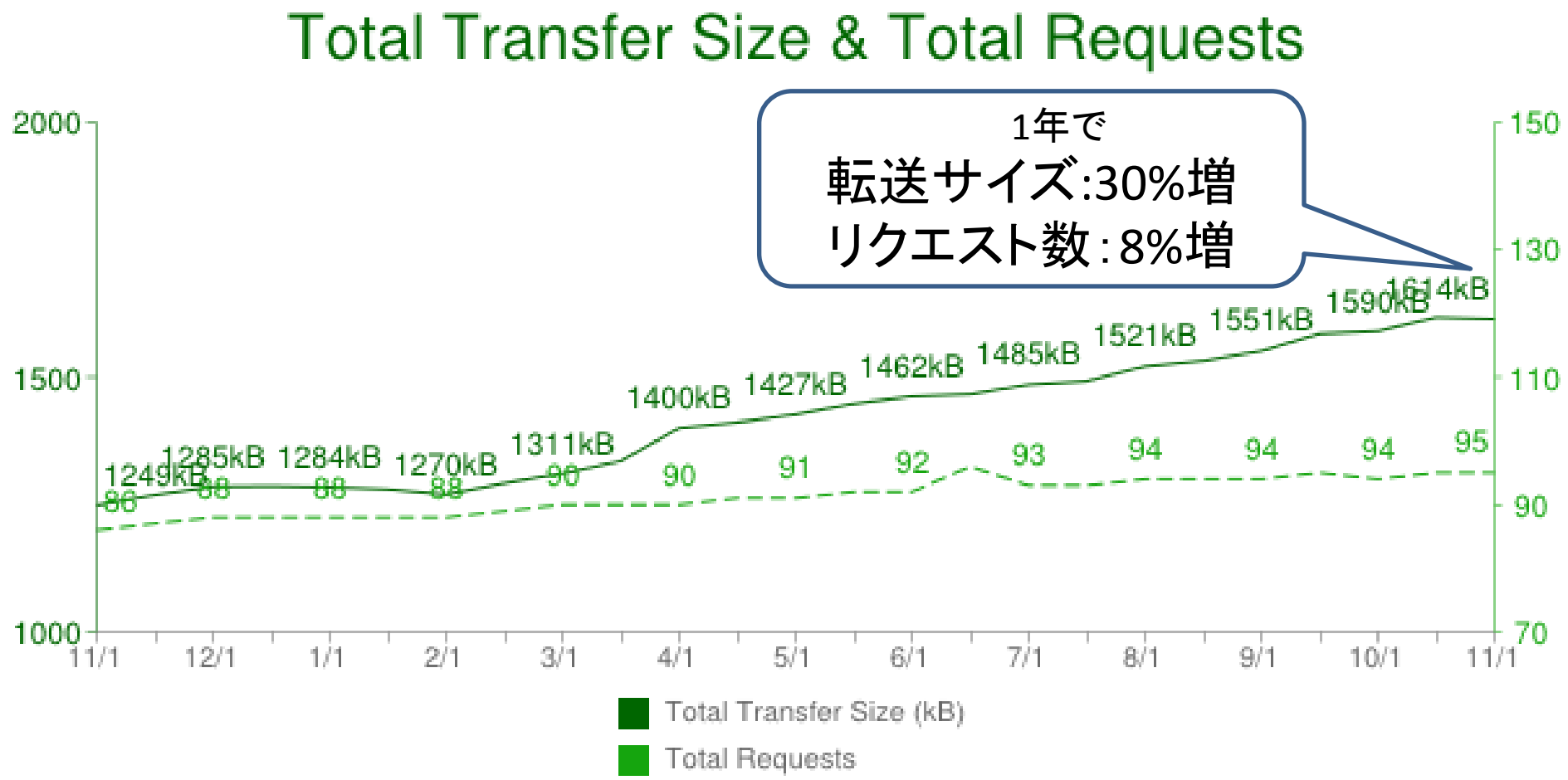
HTTP/2.0で変わるWebの世界

IIJプロダクト本部 戦略的開発部
大津繁樹

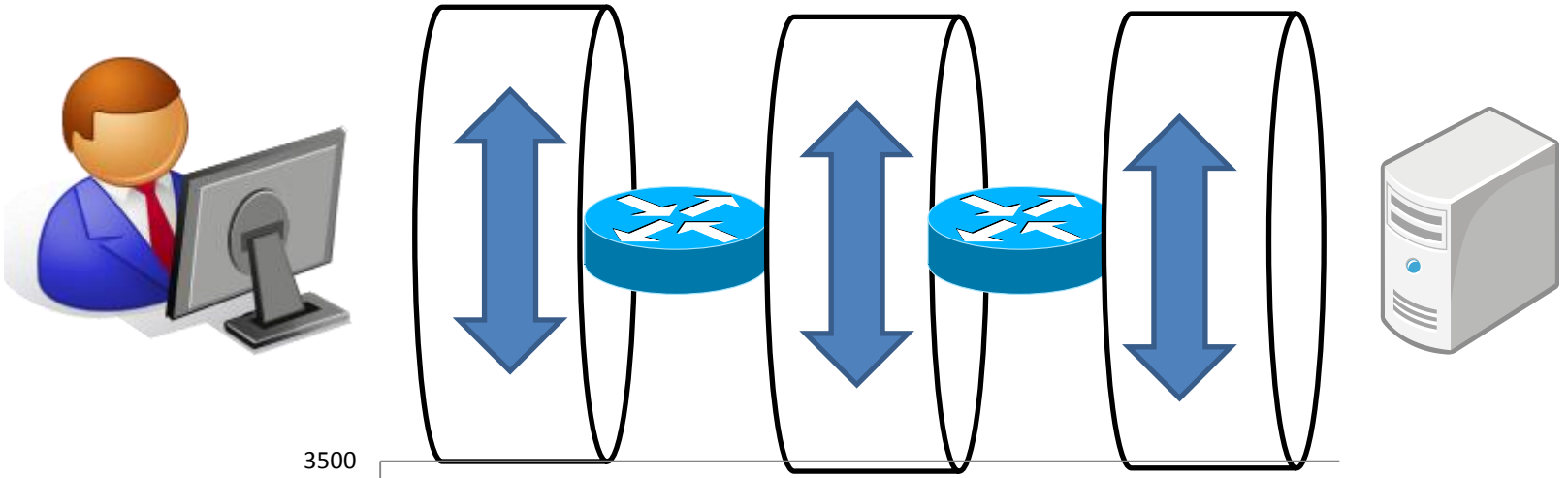
IIJ Technical Week 2013
2013年11月21日

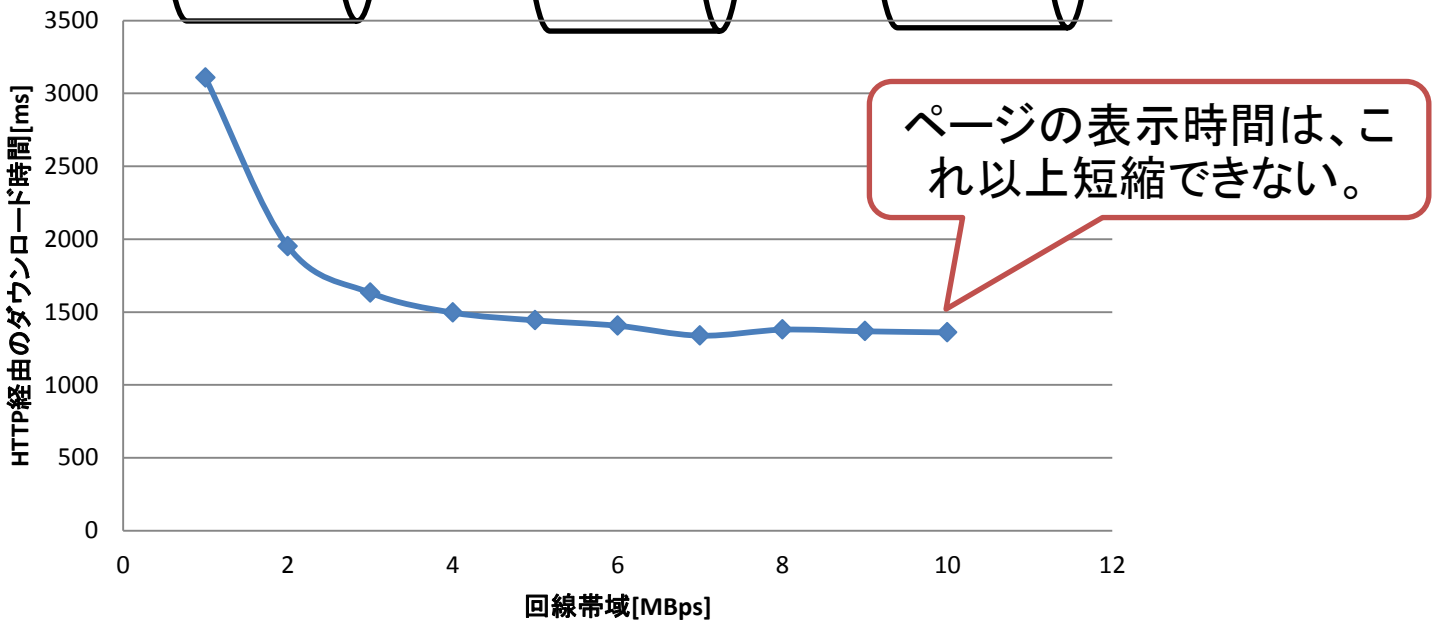
HTTP転送サイズとリクエスト数の遷移
(2012/11～2013/11)



http://httparchive.org/trends.php?s=All&minlabel=Oct+1+2012&maxlabel=Nov+1+2013

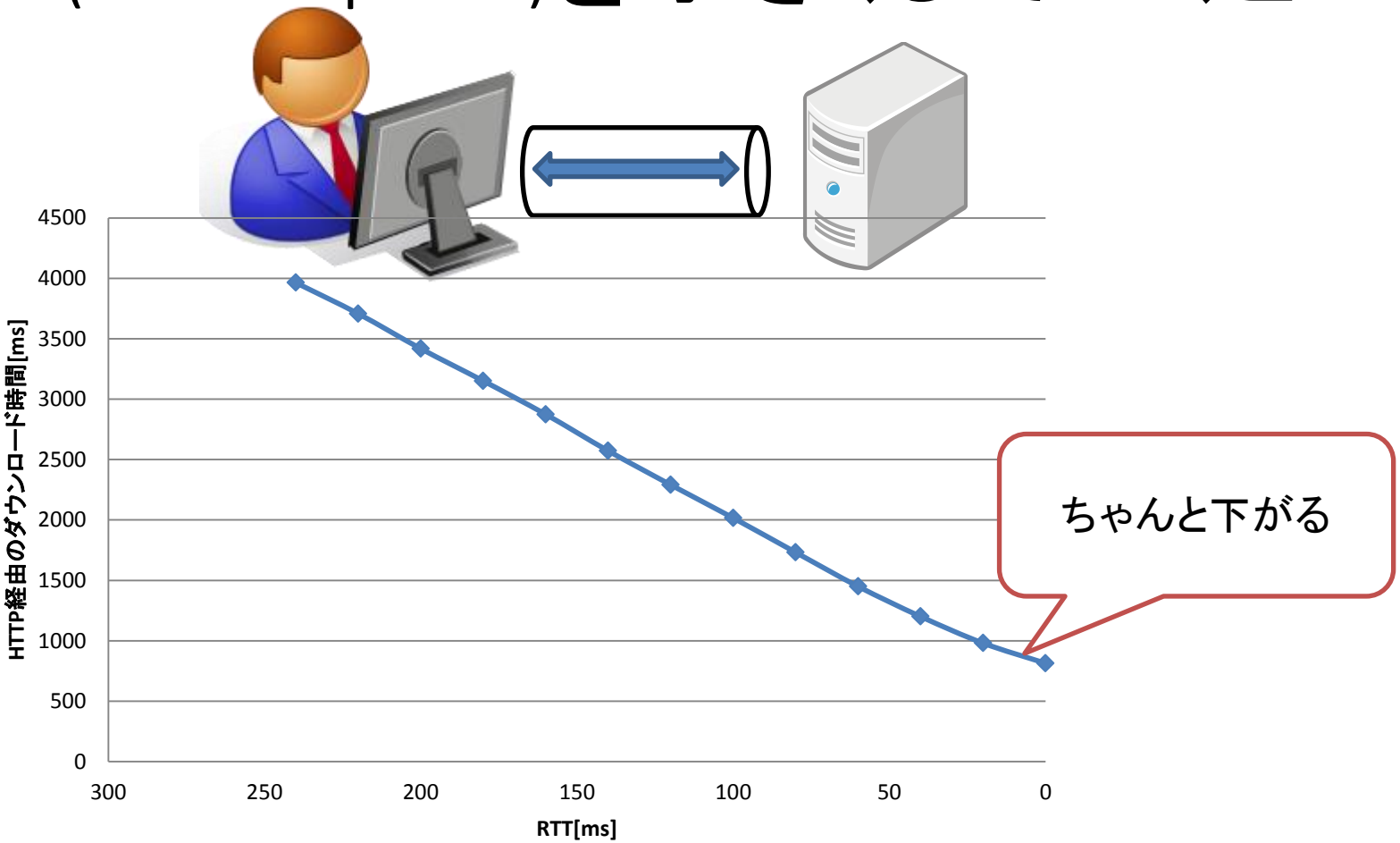
回線帯域を増速していくと





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

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

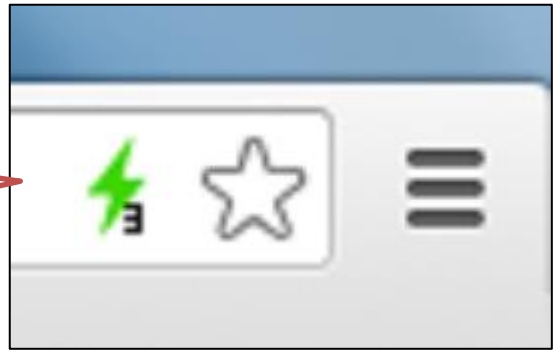
More Bandwidth does'nt matter よりデータ引用
http://docs.google.com/a/chromium.org/viewer?a=v&pid=sites&srcid=Y2hyb21pdW0ub3JnfGRldnxneDoxMzcyOWI1N2I4YzI3NzE2

SPDY(スピーディ)とは、

- SPDYは、Googleの社内プロジェクトから生まれ、Webページの表示速度を速くするためのプロトコルである。
- 既に2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へSPDYの導入が始まっている。
- SPDYは、HTTPの次期バージョン(**HTTP/2.0**)のベース仕様として採用され、現在標準化の議論が進められている。

SPDYを見るには？
SPDY Indicator

ブラウザの拡張プラグインを使うとSPDYを使っているサイトがわかる



SPDYを見るには？
chrome://net-internals/#spdy

利用しているSPDYセッション
をリアルタイムで見れる

SPDYの歩み

2009/11	spdy/1 仕様公開
2010/01	TLS NPN拡張仕様のドラフトリリース
2010/02	spdy/2 仕様公開
2010/09	Chrome6安定版リリース。SPDYがデフォルトで有効になる。
2011/01	**Google サービスの90%がSPDY化完了のアナウンス**
2011/05	spdy/3 仕様公開
2011/12	FireFox11 開発版に SPDY実装される
2012/03	**Twitter** SPDY化開始
2012/08	**wordpress.com** が SPDY対応開始
2013/01	**LINE**がSPDYを利用していることを公表 **Facebook** が SPDY対応開始
2013/06	Win8.1 の IE11 (preview)で SPDY対応していることが判明
2013/09	SPDY/3.1 仕様公開

ブラウザのSPDY対応状況

ブラウザ	デスクトップ版	モバイル版
Chrome	**対応** (Ver. 6以上)	**対応** (Ver. 18以上)
FireFox	**対応** (Ver. 13以上)	**対応** (Ver.15以上)
Opera	**対応** (Ver. 12.10以上)	**対応** (Ver. 12.10以上)
Android標準ブラウザ	----	**対応** (Ver. 3以上)
Safari	未対応	未対応
Internet Explore	**対応** (Ver. 11 on WIn8.1以上)	?

(標準でSPDYが有効になったバージョンを記載)

サーバソフトのSPDY対応状況

サーバソフト	言語
node-spdy	Node.js
spdylay	C
apache mod_spdy	C
Jetty	Java
nginx(Proxy用途)	C(現状 ver.2のみ)

（他に python, ruby 実装も)

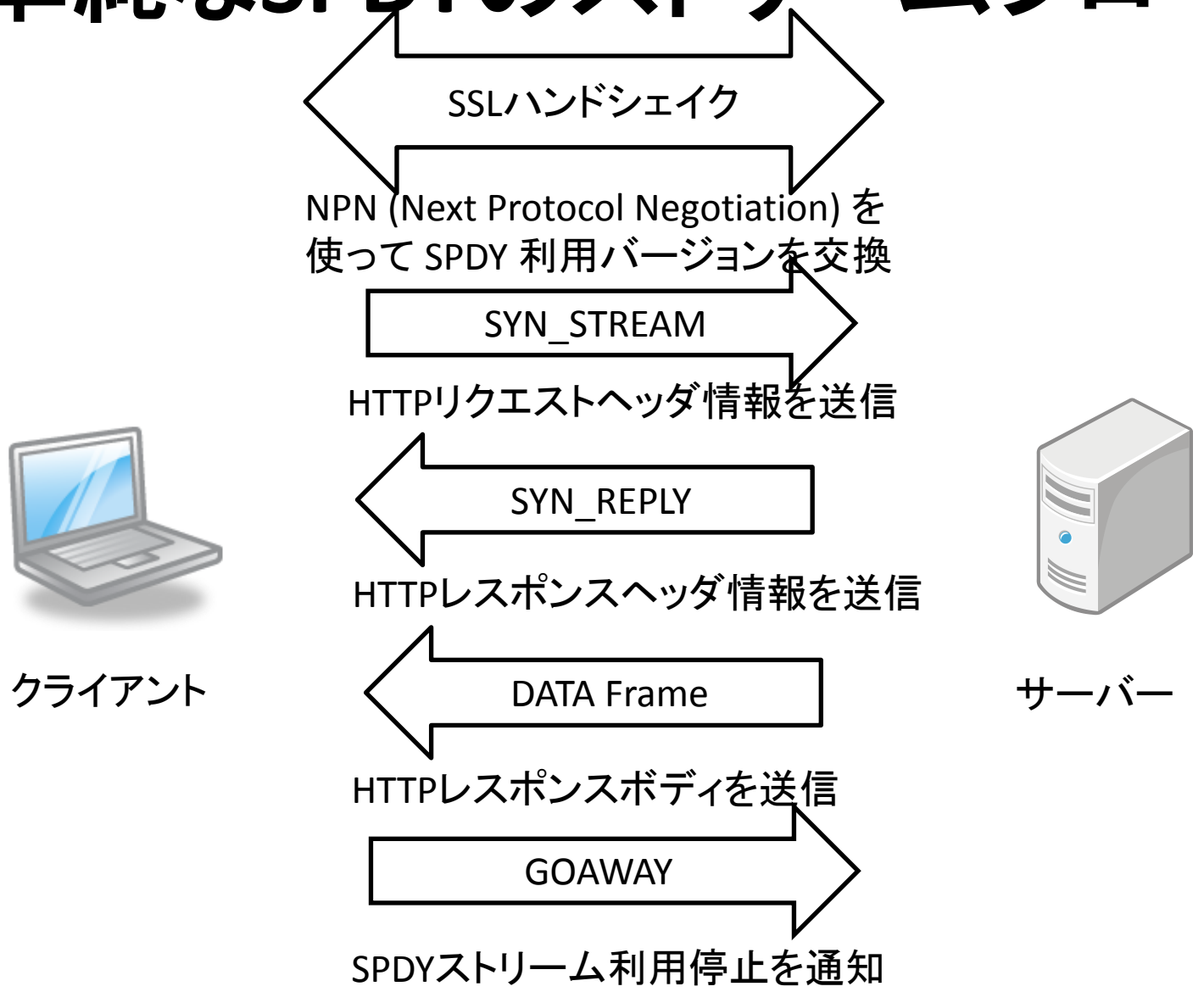
SPDYの利点

1. ブラウザへのWebサイトのページ読み込み時間を短縮する。
2. Webコンテンツの変更を必要としない
3. 既存のHTTPサーバと互換性を保ちつつ簡単に順次デプロイ(Drop-in replacement)が可能である

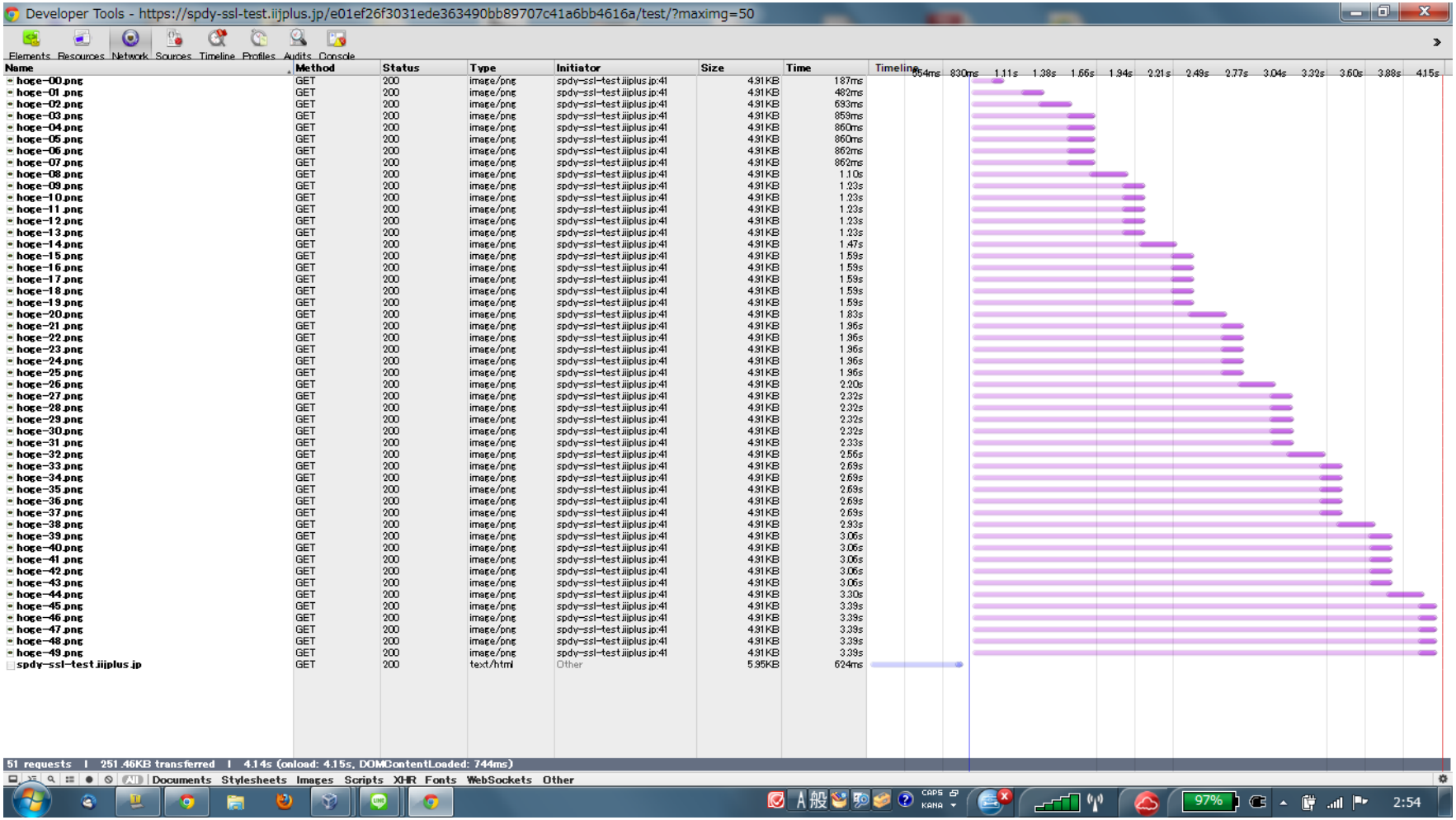
SPDYの特徴

- TLSと連携してプロトコルを自動選択
- HTTPヘッダデータの圧縮
- 優先度付全２重多重化通信
- サーバプッシュ機能

単純なSPDYのストリームフロー

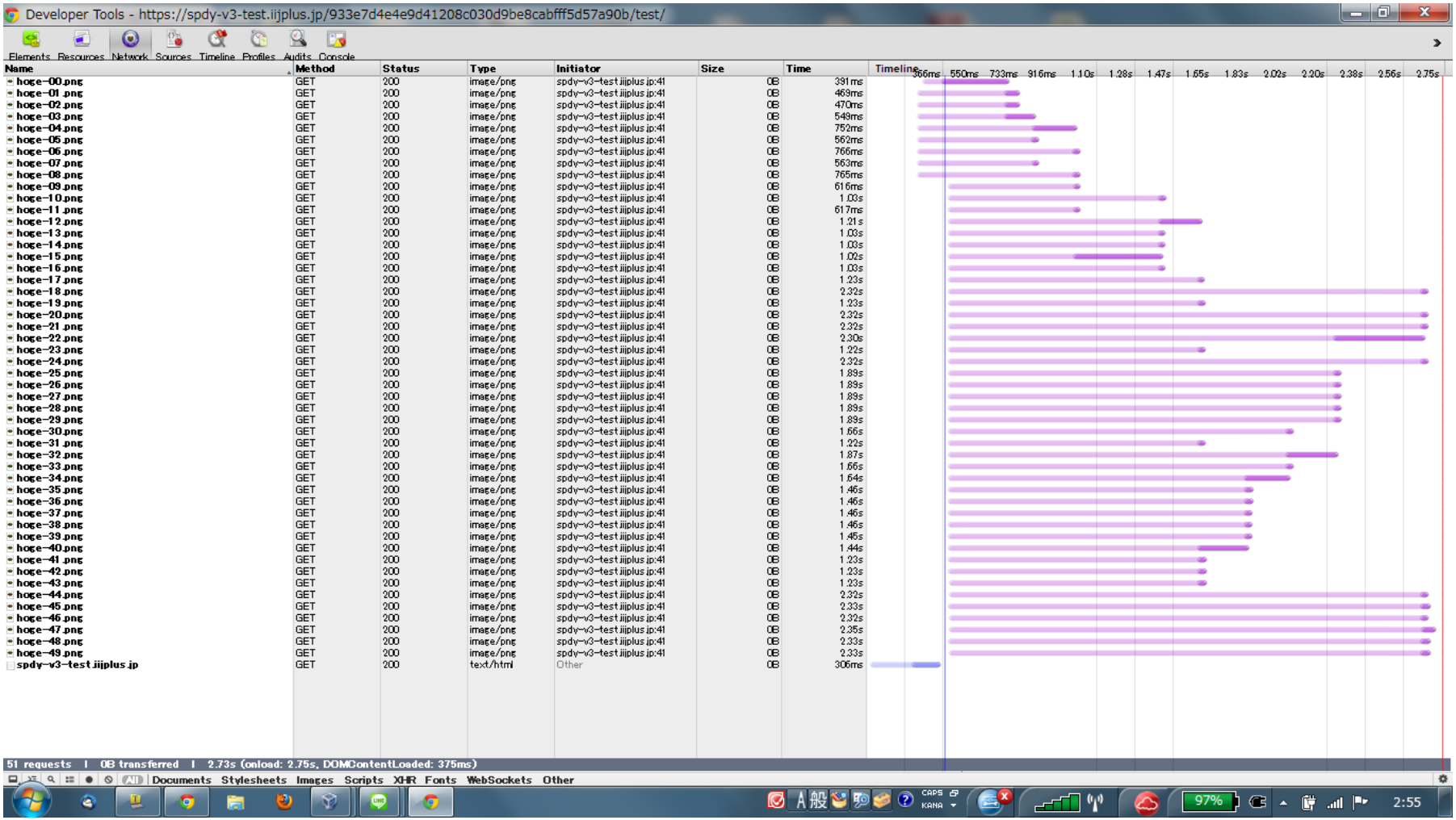


SSL(HTTP/1.1)のHTTPタイムライン



クライアントからサーバへの同時接続数が6に制限

SPDY/3のHTTPタイムライン



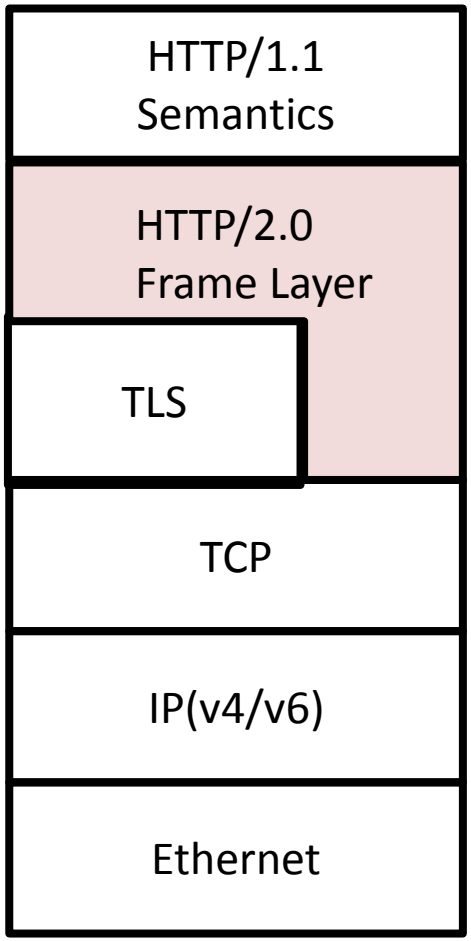
1つのTCP接続上でHTTPリクエストを多重化

SPDYに向かないサイト

1. 多数のドメインに分割してコンテンツを配置しているサイト
2. 一度にダウンロードするコンテンツの量があまり多くないサイト
3. クライアントからの通信経路の品質があまりにも悪いサイト

HTTP/2.0とは、

- HTTP/1.1 の策定(1999年)から14年。
- IETF httpbis WG で HTTP/1.1 の仕様の整理の見込みがたった。
- 新しい仕様を作る動きが開始
- 従来のHTTP/1.1のセマンティクス維持。互換性保持



HTTP/2.0 これまでの歩み

年 月	トピック
2012年1月	IETF httpbis WGでHTTP/2.0の仕様検討開始することを決定
2012年11月	3つの候補案からSPDY仕様をベースにすることを決定 **draft-00**(SPDY/3仕様をそのまま)リリース
2013年1月	**第1回中間会議(東京)** **draft-01**リリース(HTTPからのUpgrade方法を追加)
2013年4月	**draft-02**リリース(フレームフォーマット・タイプの大幅な変更)
2013年5月	**draft-03**リリース(中間会議に向けて修正点の整理・まとめ)
2013年6月	**第2回中間会議(サンフランシスコ)** 2候補案を合わせたヘッダ圧縮仕様の採用を決定
2013年7月	**draft-04**リリース(最初の実装仕様)
2013年8月	**第3回中間会議(ハンブルグ)** 最初のHTTP/2.0相互接続試験を実施 **draft-05**リリース(接続試験結果を反映) **draft-06**リリース(次の相互接続試験向け実装仕様)
2013年10月	**第4回中間会議(シアトル)** 2回目の相互接続試験を実施 draft-07リリース(中間会議の議論を反映)

HTTP-draft-06/2.0 対応相互接続試験実装リスト (2013/10時点)

	名称	実装言語	Client,Server, Intermidate	ニゴシエーション
1	nghttp2	C	S, C, I	NPN, Upgrade, Direct
2	http2-katana	C#	S, C	ALPN, Upgrade
3	node-http2	Node.js	S, C	NPN, direct
4	Mozilla Firefox	C++	C	ALPN, NPN
5	**iij-http2**	**Node.js**	**S, C**	**ALPN, NPN, Upgrade, Direct**
6	Akamai Ghost	C++	I	NPN
7	Chromium	C++	C	ALPN, NPN
8	Twitter	Java	S, C	NPN
9	Wireshark	C	other	NPN, ALPN
10	Ericcson MSP	C	proxy	

(https://github.com/http2/http2-spec/wiki/Implementations より引用)

HTTP/2.0の特徴

- HTTP/1.1のセマンティックスは変えない。
- SPDYのプロトコルアーキテクチャはそのまま利用
- 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/3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、HTTP/2.0はTLS以外の接続も利用可能にする
- CRIME脆弱性対策として新しくHTTPヘッダ送受信仕様を策定する（ヘッダ差分・変更を通知）

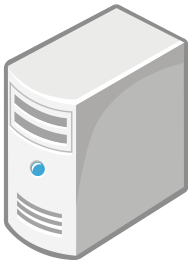
HTTP/2.0初期ニゴシエーション 3種類で2段階（その１）

(1) TLS + ALPN

TLS接続時にALPN拡張フィールドを利用して
HTTP/2.0に接続を行う。

(2) HTTP Upgrade

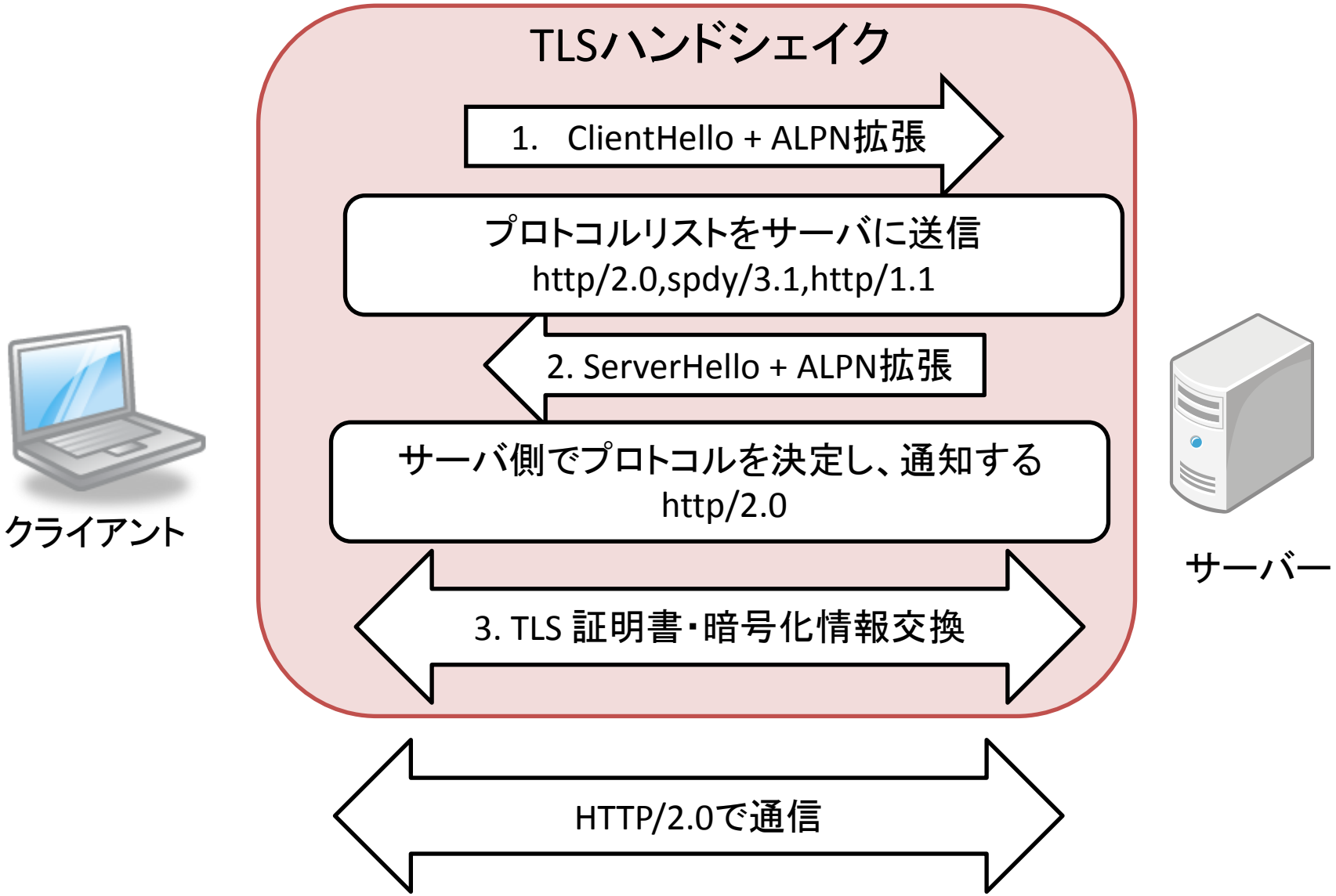
HTTP/1.1の接続後 Upgradeヘッダを使って、
HTTP/2.0 に接続をアップグレードする。



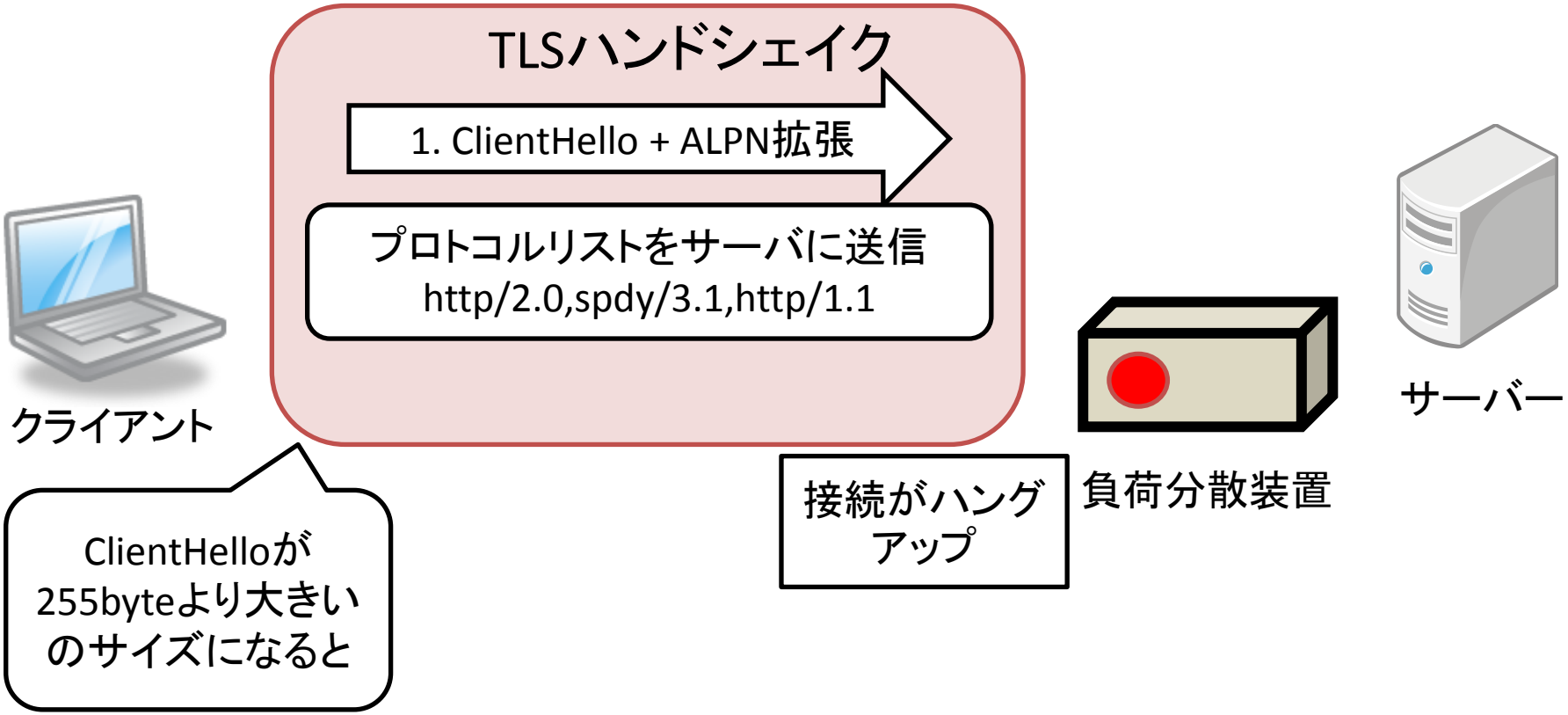
(3) Direct接続

あらかじめサーバがHTTP/2.0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)

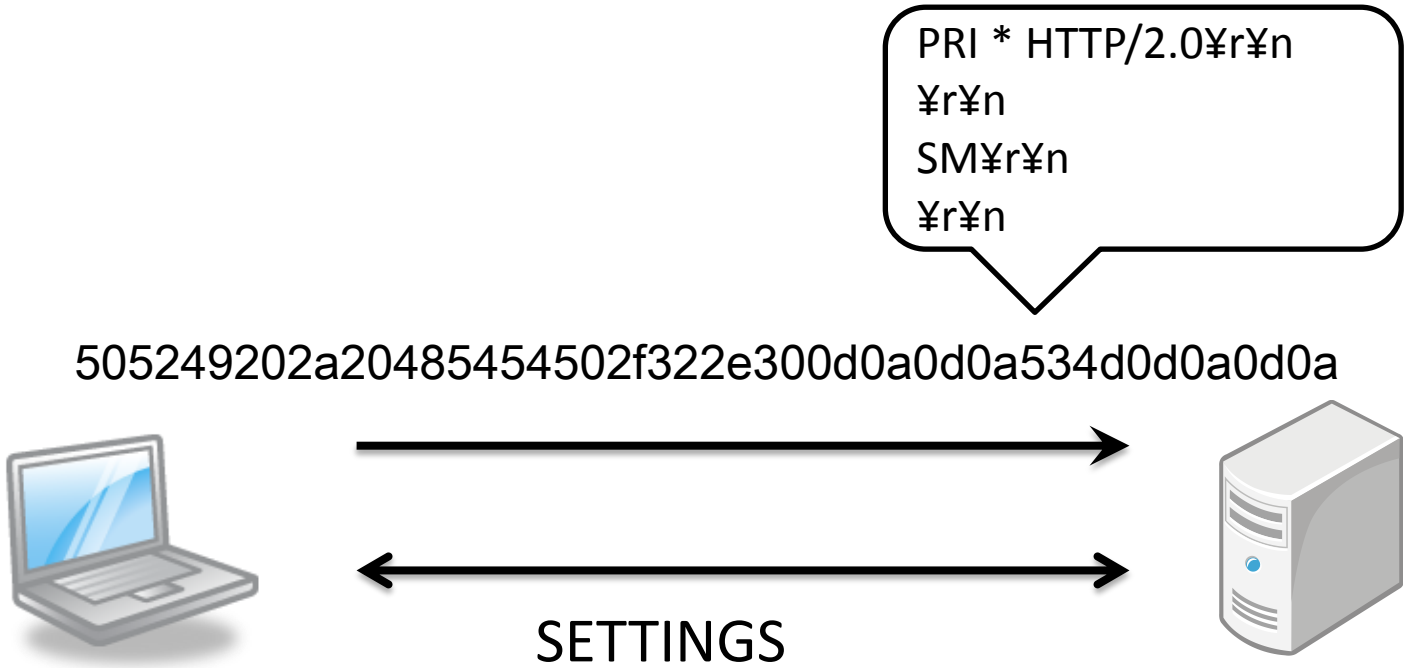
ALPN
(Application Layer Protocol Negotiation)



ご注意ください
ALPN利用時の不具合発生例



HTTP/2.0初期ニゴシエーション 3種類で2段階（その２）



(初期ウィンドウサイズ、ストリームの最大同時オープン数等の設定情報を含む）

クライアントから謎の24byteのマジックコードをサーバに送り、初期情報(SETTINGSフレームを交換する）

HPACK:新しいヘッダ圧縮仕様

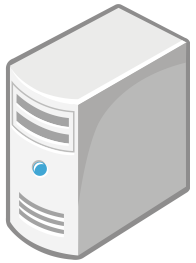
http で serverA へx-hoge: hoge 付けて GET / したいなぁ

ヘッダテーブル
0, :scheme, http
1, : scheme, https
2, :host,
3, :path, /
4, :method, GET
・・・・
38, x-hoge, hoge

1. 0番そのまま使う
2. 2番 serverAに書き換えるよ
3. 3番そのまま使う
4. 4番そのまま使う
5. x-hogeは新しく採番してhogeを入れ込む

(ヘッダ差分情報をやり取りする)

0x81
0x04 0x03 0x07 serverA
0x83
0x84
0x40 0x06 x-hoge 0x04 hoge
（実際に送信するヘッダ情報）



ヘッダテーブル
0, :scheme, http
1, : scheme, https
2, :host,
3, :path, /
4, :method, GET
・・・・
38, x-hoge, hoge

いよいよ本題

HTTP/2.0でWebはどう変わるのか？

実は何も変わらない

- 2011年1月よりGoogleサービスは全面SPDY化
- Chrome ユーザは、その違いに気づいただろうか？

- 確かに昔より表示が速く、スムーズになったような感じはある。いろんな最適化の複合要因であろう。

- 標準化で多種ブラウザーが対応するだろう。

サービスインフラ視点では？

- 大規模システムでは、＊おそらく＊変わるのではないか？（手元に定量的なデータがない）
- SPDYでは、TLS利用によるオーバヘッドより性能向上効果が上回ったとの話も。
- サーバリソース、ネットワークリソースを効率的に利用できるので大規模サービスになればなるほどその効果を享受できるのではないか？

実は単純導入だけでは無理？

- ブラウザが決めるリソース取得の優先度にどう対応する？
- プロキシ構成などで異なるトラフィックをフロー制御して最適化を図れるのか？
- 細かいチューニング手法は未知数。運用してサイトにあった構成を。日々の測定が大事。

その先の使い方

- ブラウザー以外の利用への展開
  - {key,value} + data をやり取りする独自アプリ(LINEなど)
- 原理的には接続後サーバ側からリクエストも可能(双方向化)
- いろんな付加情報をあらかじめ送りつける(DNS, Proxy, NTP 等々)

今後は、

- Internet Giants を中心にHTTP/2.0の導入が進む。（先行者利益を享受）
- 小規模サイト、一般ユーザの移行メリットは少ないだろう。HTTP/1.1はなくならない。（二極化）

- ブラウザ側の対応も進み、ますますHTTP/2.0に最適化されるであろう。
- 特にモバイル環境での性能改善に期待がかかる。