HTTP/2.0 Tutorial

(株)レピダム
清水 一貴
(@kazubu)

HTTP/2.0って?

HTTP/2.0って?

HTTP/1.1の高効率版

これまでの経緯

2012/03 IETF83(Paris, FR)

- 年頭ごろから次世代HTTPの話題がちらほらと
- IETF83 httpbis WGのセッションにて、HTTP/2.0への取り組みを開始することについてコンセンサス
- この時点での提案は以下の4つ
  - SPDY by Google
  - HTTP Speed and Mobility(S+M) by Microsoft
  - Network-Friendly HTTP Upgrade by W.Terreau, et al.
  - Waka by Roy Fielding
    - HTTP/2.0の方向性と異なることもあり、正式に提案はされていない

2012/07 IETF84(Vancouver, CA)

- SPDYをベースに採用することで合意
  - SPDYを参考にして新たに作っていく方向
- SSL/TLSの必須化は行わない方向に
  - HTTPSに限定はしない
- アップグレード方法について議論が開始
  - ひとまず今までにあった提案の整理程度

2012/09 CRIME Attackの発表

- 圧縮率の変化を元に、SSL/TLS上で行われる通信の一部を推測する攻撃手法
- 特定の条件下において、非常に効率的にCookie等を推測することが可能
- SPDYがそのターゲットとなり、SPDYでは圧縮機能の無効化等のデグレを余儀なくされた

これにより、圧縮に対する見方が少なからず変化し、セキュアな圧縮手法が求められるようになった

2012/11 IETF85(Atlanta, US)

- アップグレードメカニズムとヘッダー圧縮についての議論が主
- アップグレードメカニズム
  - HTTPSのときはNPNが最有力
  - HTTPのときにどうするか
- 圧縮
  - 様々なサイトのヘッダー等を収集
  - SPDYの圧縮はそのまま利用できないため、新たな方法の検討

2013/01 httpbis interim(Tokyo, JP)

- 全体について広く方針決め
- アップグレード方法に関する新しい提案
  - 新しいDNSリソースレコード?
  - 非対応クライアントを素早く諦めさせる方法
- 安全なヘッダー圧縮方式
  - 差分のみやり取りする方式等
- フレームの取り扱い
- フロー制御・優先度制御, etc…

2013/03 IETF86(Orlando, US)

- Interim Meetingで決まったことの合意形成がメイン
- TLS WGにおいて、TLS層におけるアプリケーション層プロトコルの交渉方式として、ALPNをベースに進めていくことに合意

なぜHTTP/2.0が必要なのか

Webの台頭

- インターネットでHTTPが占めるトラフィック量の増加
- なんでもWebの上で行う風潮
  - 静的コンテンツ
  - 動的コンテンツ(リッチコンテンツ)
    - 地図
    - ゲーム
    - 動画
    - 音声
  - リアルタイムコミュニケーション
    - (WebRTC)
    - WebSocket
  - (API)

モバイルデバイスの増加・性能向上

- 2011年初頭から終わりにかけて、モバイルトラフィック量は2.21倍に(*)

→今後さらなる増加が予想される

- モバイルデバイスの性能は日々高まる一方
  - 例: ARM 1.9Ghz 4-Core CPU, 2GB RAM

→コンテンツの重量化

→トラフィック・コネクション数増加

→ユーザ体験のボトルネックは

端末の速度ではなくネットワークへ

*: インターネット白書2012 P.193より

モバイルネットワークとHTTP

- HTTP/1.1で通信する際、1つのWebサイトにアクセスする際のオーバーヘッド
  - DNSリクエスト+レスポンス x ページ内で参照されるURLのFQDNの数
  - TCP 3-way Handshake x 1～HTTP最大同時接続数 xページ内で参照されるURLのFQDNの数
- モバイルネットワーク等の高レイテンシ環境で、複数TCPコネクションを張るのはコスト高

利用可能なTCPポート数の減少

- IPv4アドレスの枯渇

→CGNの導入

→IPアドレスの共有

→一人当たりの利用可能TCPポート数が減少

HTTP/2.0はどうなるのか

HTTP/2.0が目指す効果

パフォーマンスの改善とUX向上

- トラフィック量の削減
- レイテンシ・Round-Trip数の削減
- ネットワーク資源の有効利用
- その他

HTTP/2.0のデザイン

パフォーマンス改善を目的とし、
HTTP/1.1とセマンティクスは変更しない

なので

- HTTP/1.1とHTTP/2.0は相互に変換できる
- GET, POST, PUT, … 等のメソッドに変更はない

具体的に何が変わるのか？

- ヘッダーのバイナリ化
- ヘッダーの圧縮
- 多重化(Multiplexing)
- 優先制御(Prioritizing)
- TCPコネクションの利用方針
- HTTP通信の開始方法
- など

具体的に何が変わるのか？

→ヘッダーのバイナリ化

- ヘッダーの圧縮
- 多重化(Multiplexing)
- 優先制御(Prioritizing)
- TCPコネクションの利用方針
- HTTP通信の開始方法
- など

ヘッダーのバイナリ化

- 今までのヘッダーの例

GET / HTTP/1.1¥r¥n
Host: example.com¥r¥n
Accept-Encoding: gzip, deflate¥r¥n
User-Agent: Mozilla/4.0 (.....) ¥r¥n
Cookie: hogehogehogehoge¥r¥n
Connection: Keep-Alive¥r¥n

HTTP/1.1 200 OK¥r¥n
Date: Wed, 3, Jul 2013 12:34:56 GMT¥r¥n
Server: Hogehoge Web Server/1.0¥r¥n
Content-Length: 12345¥r¥n
……

本当に必要なデータ(request)

{:method=> :get,

:path => “/”,

:scheme => “http”,

:host => “example.com”,

:accept-encoding => [:gzip, :deflate],

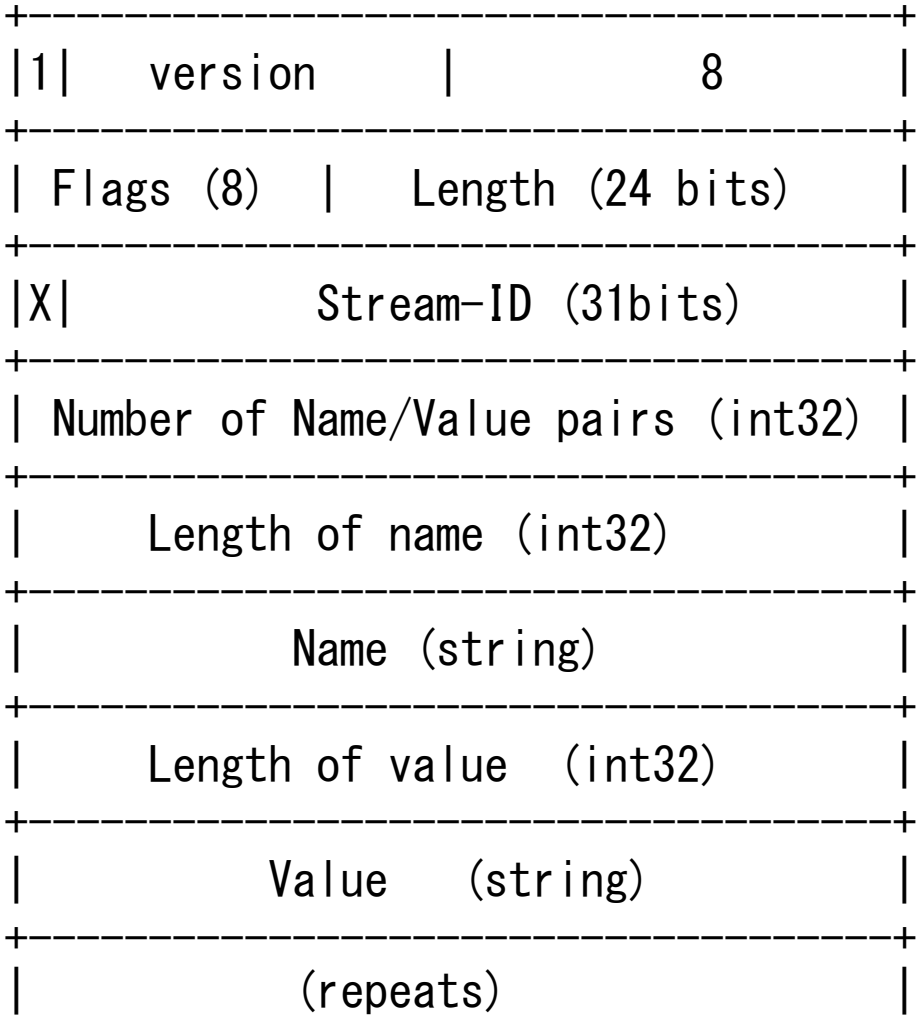
:user-agent => “Mozilla/4.0 (.....)”,

:cookie => “hogehogehogehoge”,

:connection => :keep-alive}

実際HTTP/2.0では？

- まだ未定だったり…
- (参考)SPDYでは →



具体的に何が変わるのか？

- ヘッダーのバイナリ化

→ヘッダーの圧縮

- 多重化(Multiplexing)
- 優先制御(Prioritizing)
- TCPコネクションの利用方針
- HTTP通信の開始方法
- など

ヘッダー圧縮

- HTTP/1.1で圧縮はサポートされているものの、対象はコンテンツのみ
- HTTPS通信時はTLSの圧縮機能によって圧縮されていたケースも存在
  - CRIME攻撃の対象となりうる
- SPDY/3まではZlibを用いていたが、CRIME攻撃によって事実上無効化を余儀なくされた
- 安全で高効率な方式を検討中

考えられている圧縮手法

- よく使われる文字列の符号化
- 前回のリクエストからの差分のみ送信
  - 例: :pathだけ書き換えて再度実行
  - ストリーム毎にテーブルを保持

圧縮における問題点

- 差分のみ送信する場合
  - プロクシサーバ等に対する負担が大きい
  - テーブルの保持が必要になるため、メモリ等の資源を多く消費する
    - DoS攻撃などに弱い

具体的に何が変わるのか？

- ヘッダーのバイナリ化
- ヘッダーの圧縮

→多重化(Multiplexing)

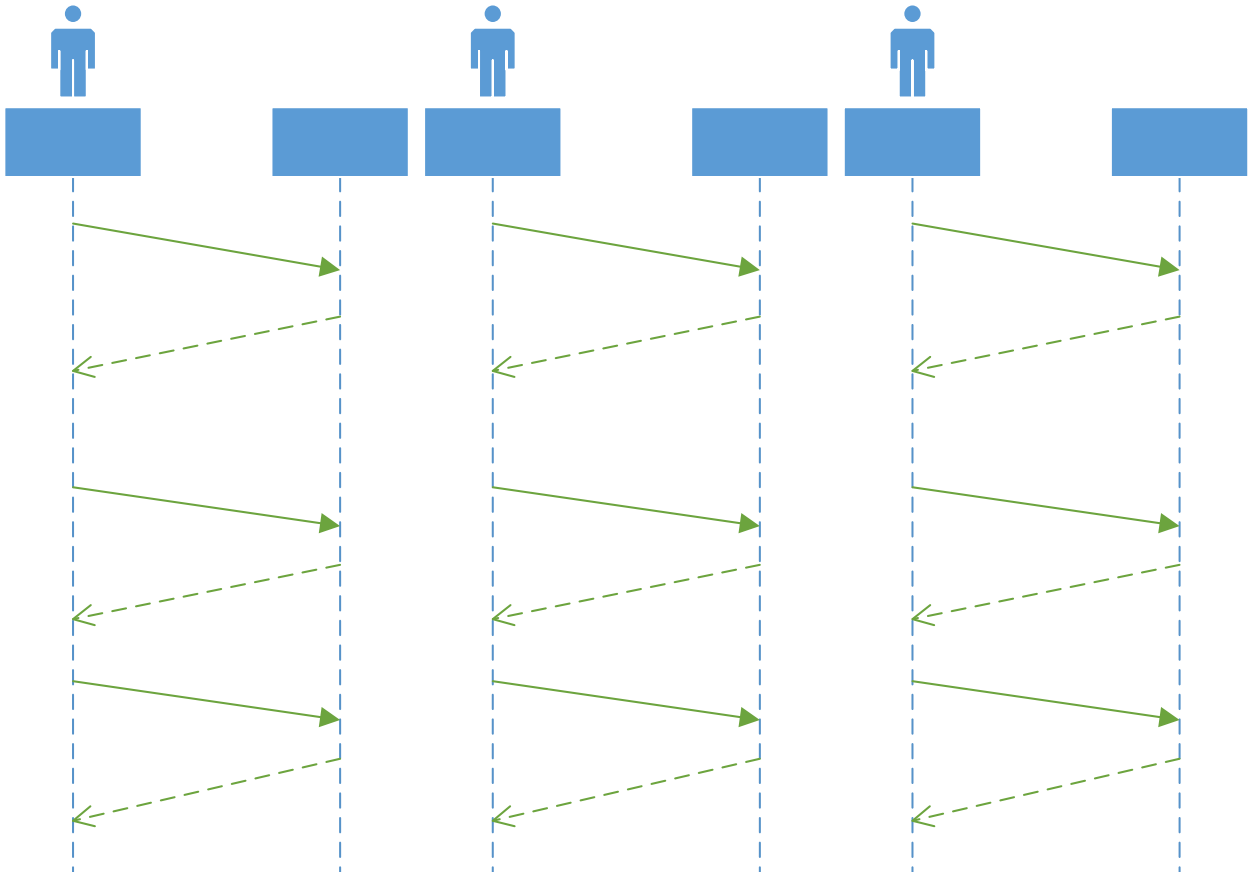
- 優先制御(Prioritizing)
- TCPコネクションの利用方針
- HTTP通信の開始方法
- など

多重化

- HTTP/1.1
  - 1つのTCPコネクション内では同時に複数のリクエストを行えない
    - HTTP Pipeliningにはいくつかの問題がある(行頭ブロッキング問題、一部のサーバやProxyにおける対応等)ため、多くのブラウザでデフォルト無効
  - そのため、同時に複数のTCPコネクションを利用することがある程度許容されている

HTTP/1.1

- 1接続につき1リクエスト、帰ってくるまで待つ

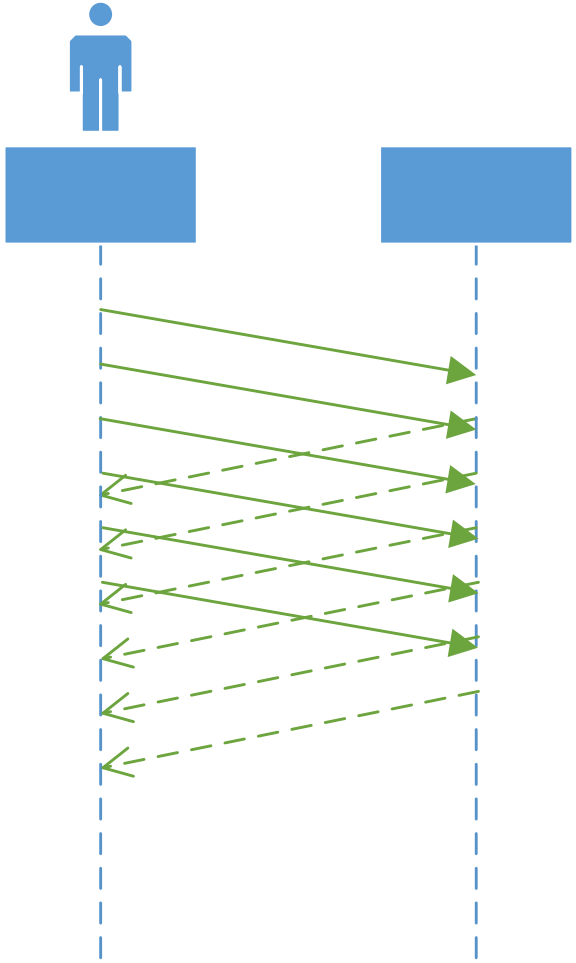


多重化

- HTTP/2.0
  - ストリームという単位でTCPコネクションの内部にチャネルを作成
  - それにより、1つのTCPコネクション内で同時に複数のリクエストが可能
  - レスポンスはリクエストの順序に依存しない

HTTP/2.0

- 順番は入れ替わっても問題ない



具体的に何が変わるのか？

- ヘッダーのバイナリ化
- ヘッダーの圧縮
- 多重化(Multiplexing)

→優先制御(Prioritizing)

- TCPコネクションの利用方針
- HTTP通信の開始方法
- など

優先制御

- ストリームごとに8段階の優先度を設定
- それをもとに、どのストリームを優先するかを決定
- たとえば、画像等のリソースは優先度を低く、CSSやJS等は少し高くする等(あくまで例)

具体的に何が変わるのか？

- ヘッダーのバイナリ化
- ヘッダーの圧縮
- 多重化(Multiplexing)
- 優先制御(Prioritizing)

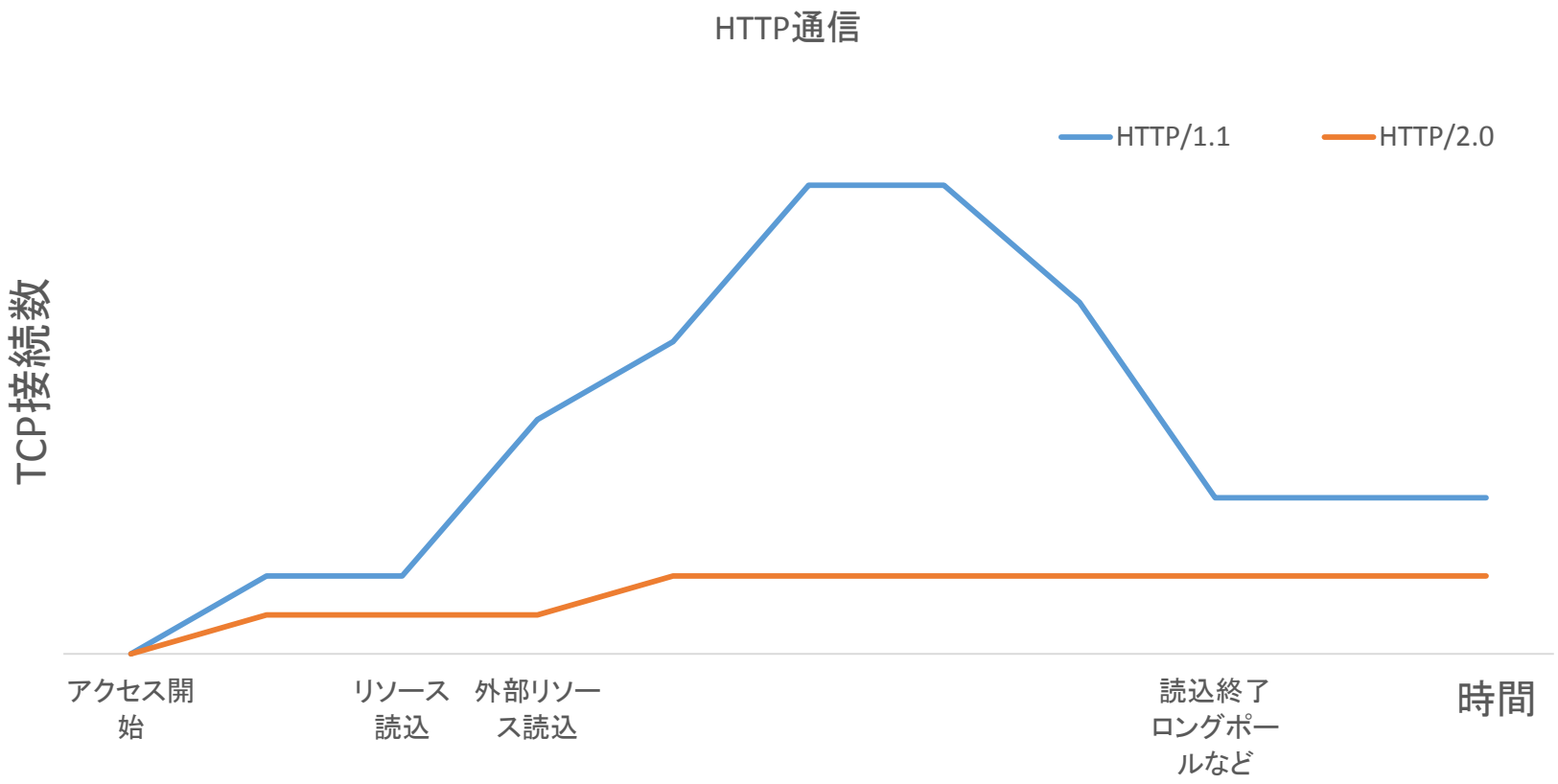
→TCPコネクションの利用方針

- HTTP通信の開始方法

TCPコネクションの利用方針

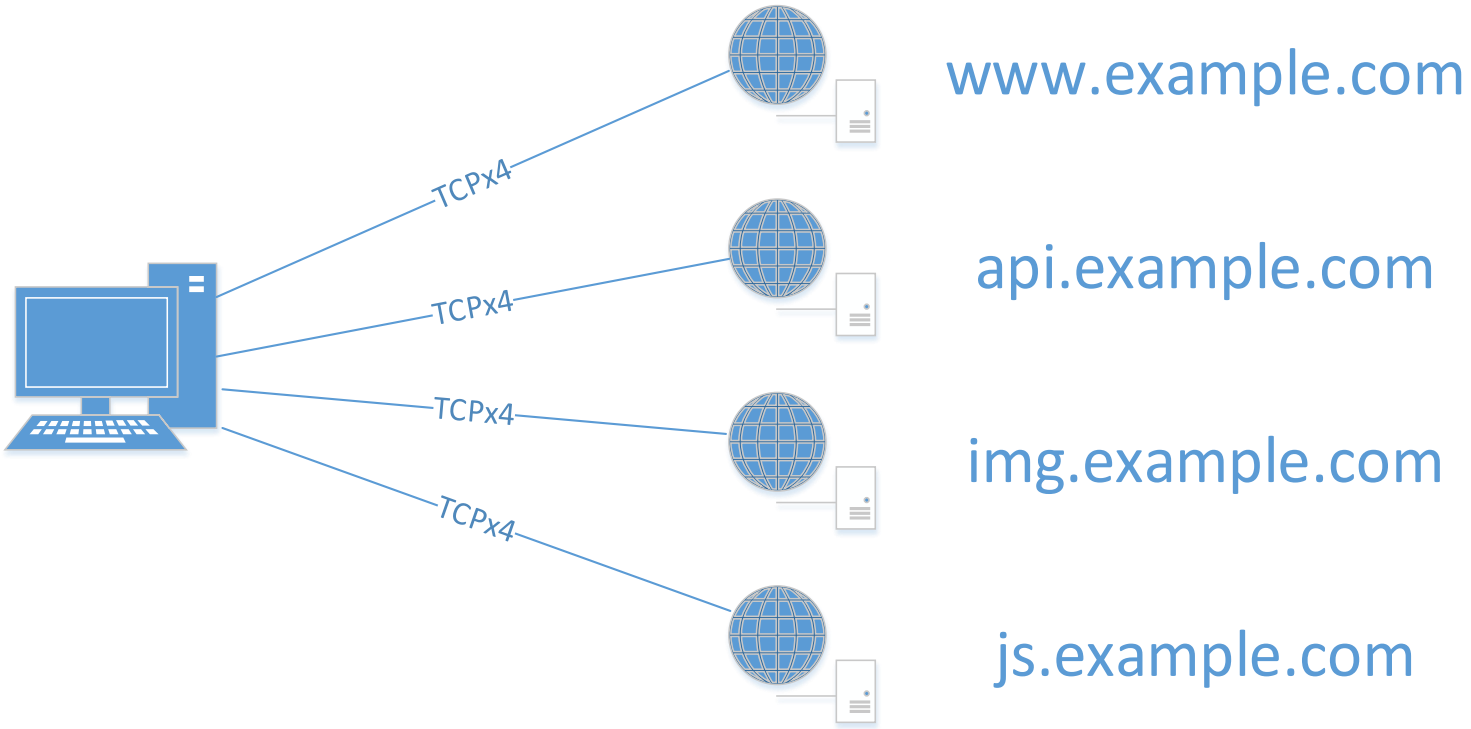
- HTTP/1.1
  - FQDNごとに2～6のTCP接続
  - 全コンテンツの受信が完了すると切断
    - 一定時間HTTP Keep-aliveで維持(サーバの設定等にもよる)
    - 必要に応じて、JavaScriptによるLong-polling, WebSocket等
- HTTP/2.0
  - 多重化により、FQDNごとに1つ程度のTCP接続
  - コンテンツ受信が完了しても切断しない(推奨)
    - タブのクローズ、ページ遷移等によって切断

TCPコネクション数予想(理想?)



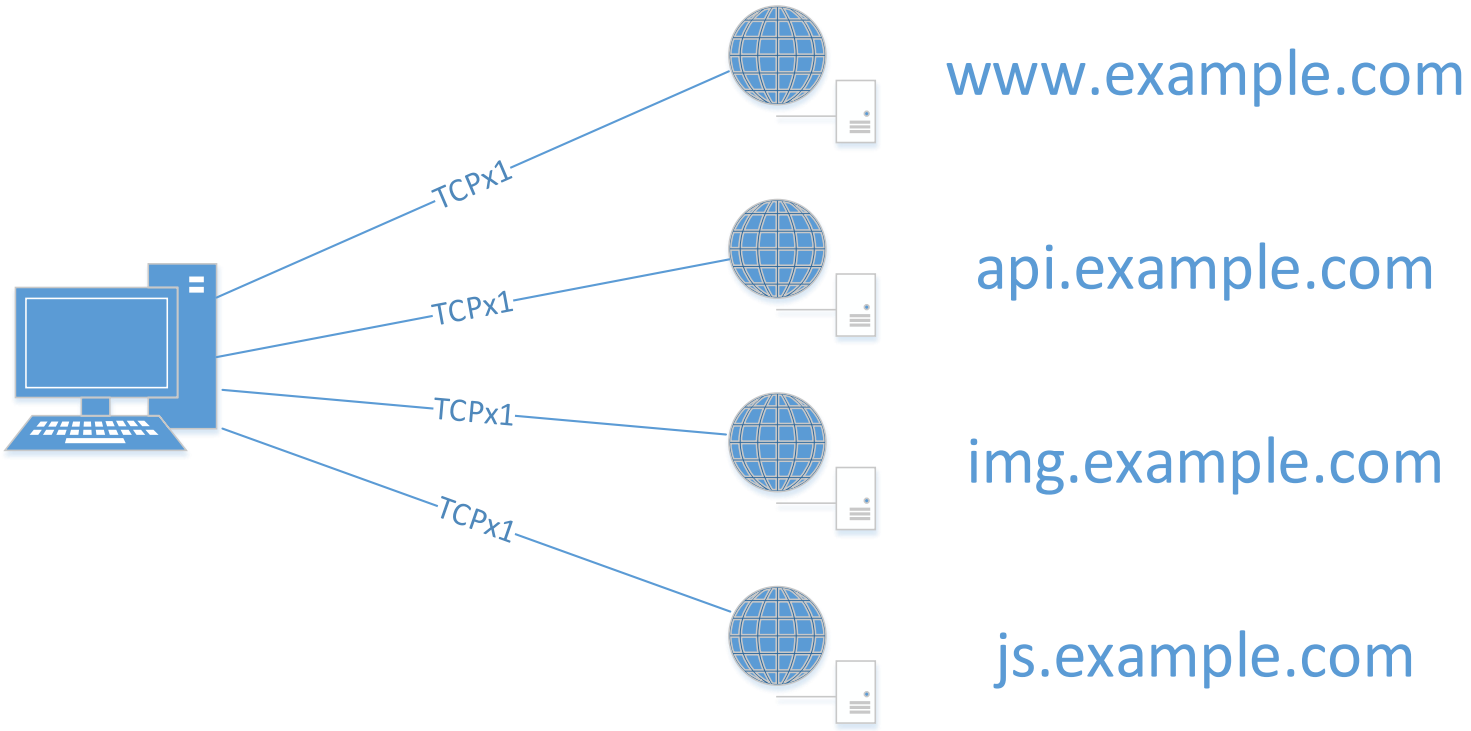
現状(HTTP/1.1)

- 「Web高速化」を行うために、1つのWebサイトのリソースを複数FQDNに分散し、同時接続数を稼ぐ手法が広く用いられている



HTTP/2.0にそのまま移行

- FQDN当たりのTCP接続数は減少する(だけ)

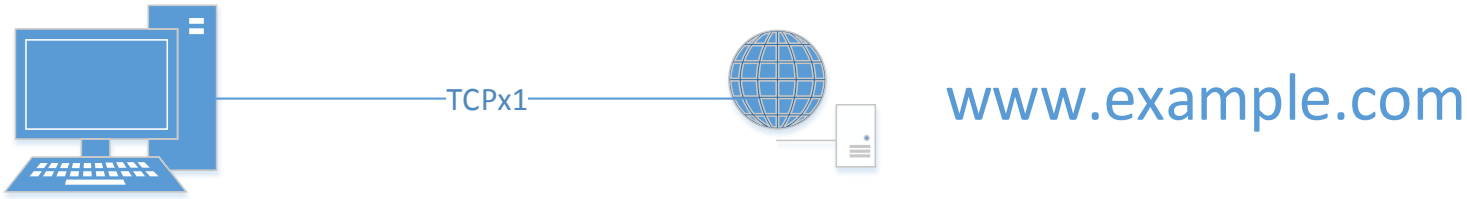


HTTP/2.0を効果的に使うには

- ドメイン分割をやめる

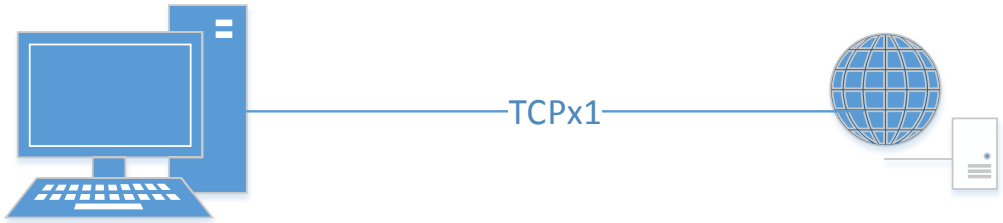
→1つのWebサイトにつき1つのTCPコネクション

→TCPコネクション数削減、新規接続コスト削減



(参考)SPDYの試験実装では

- 以下の条件の時、複数FQDNに分割されている場合でも1つのコネクションにまとめられる
  - 複数FQDNが同じIPアドレスでホストされている
  - ワイルドカード証明書等、全てのFQDNで同一の証明書が利用可能
- HTTP/2.0で可能になるかはまだ不明瞭



www.example.com
api.example.com
img.example.com
js.example.com

具体的に何が変わるのか？

- ヘッダーのバイナリ化
- ヘッダーの圧縮
- 多重化(Multiplexing)
- 優先制御(Prioritizing)
- TCPコネクションの利用方針

→HTTP通信の開始方法

HTTP通信の開始方法

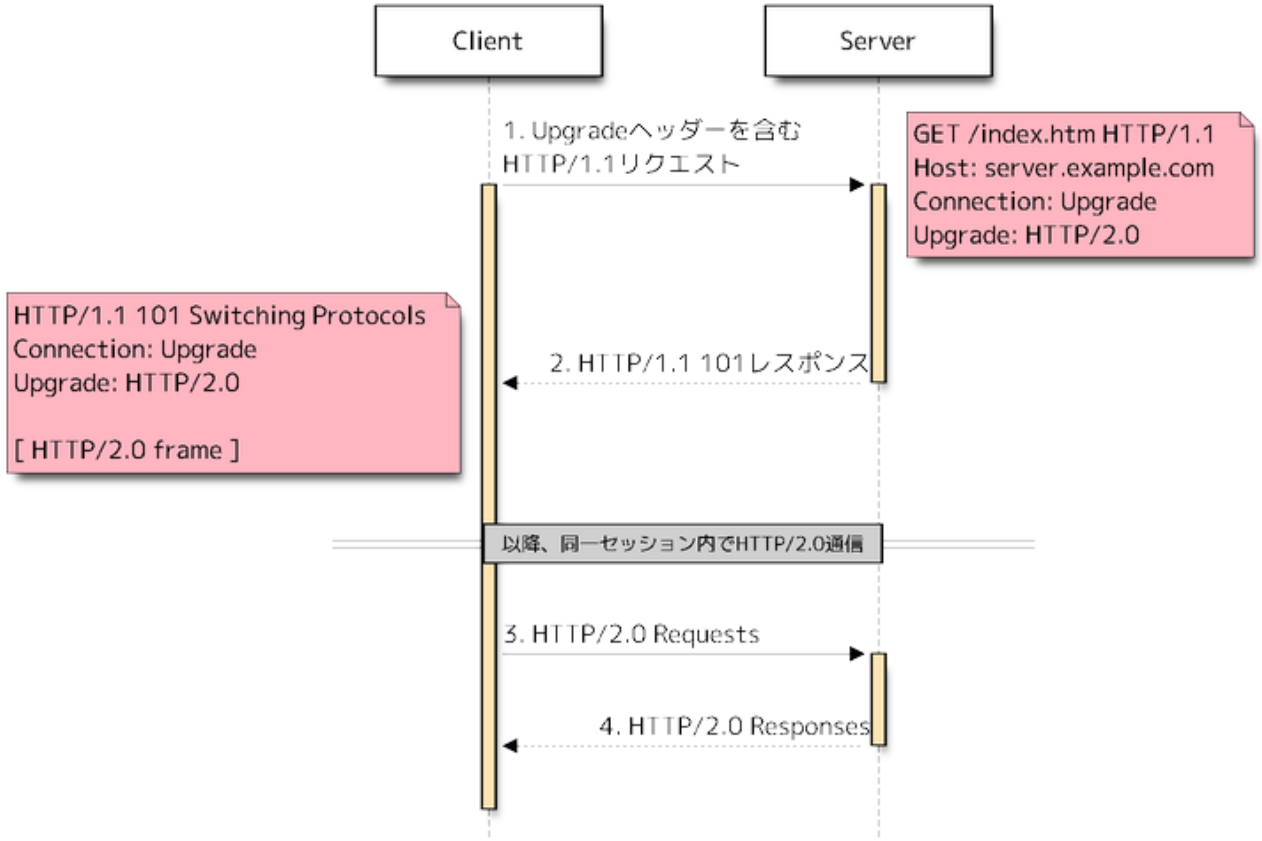
- HTTP/1.1とHTTP/2.0は、そのままでは互換性がない
- でも、80/TCP, 443/TCPは使いたい
  - http2://example.com/ ?…論外
  - 企業内ファイアウォールのポリシー等
- 通信先がHTTP/2.0に対応しているかを識別し、適切に選択する必要がある

HTTP通信の開始方法(HTTP)

- SPDYはHTTPSとセットでの利用を前提としていたため、非HTTPS通信で利用するための方法を考える必要があった
- 様々な手法が提案された
  - HTTP Upgrade Mechanism
  - DNS SRV Recordの利用
  - DNS SVCINFO Recordというものを作成
  - HTTP/1.1のレスポンスに2.0のポート番号を含める
  - など
- 現状、HTTP Upgrade Mechanismを利用することに

HTTP Upgrade Mechanism

- HTTP/1.1の仕様に存在する、Upgrade Mechanism

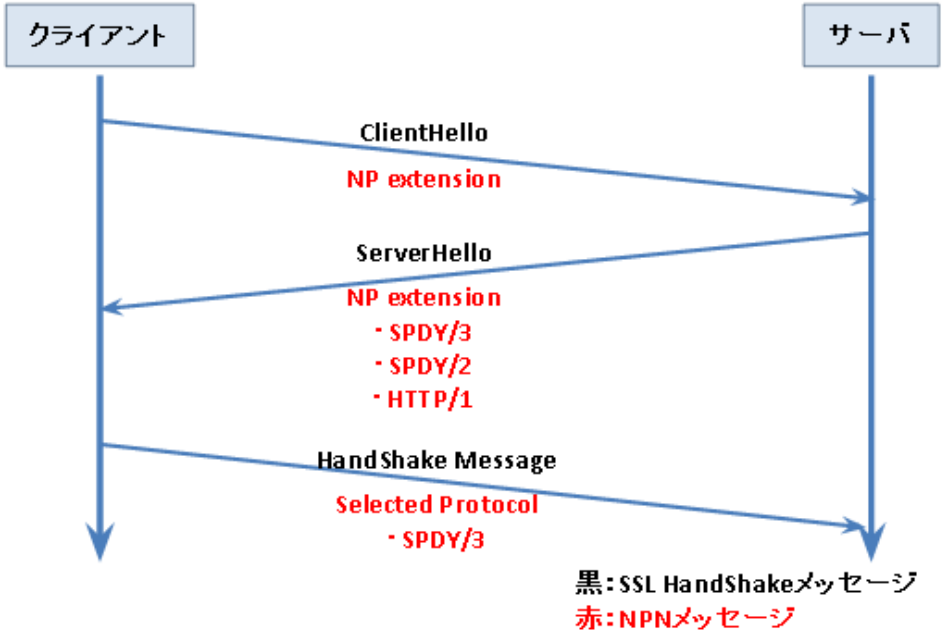


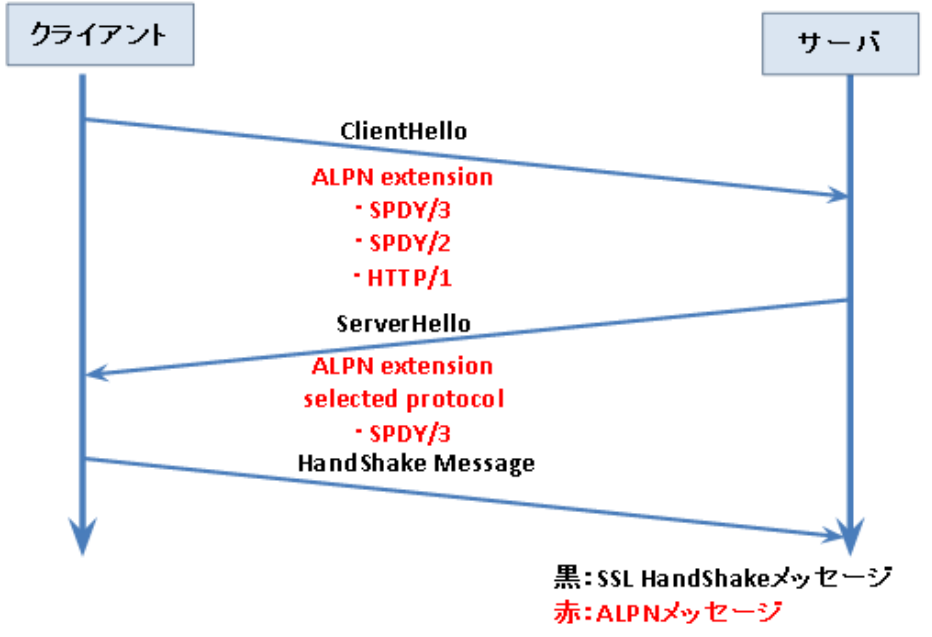
HTTP通信の開始方法(HTTPS)

- HTTPS通信時については、GoogleがSPDYを利用する際に作成した、NPN(Next Protocol Negotiation) TLS拡張が存在
- HTTPbis WGからTLS WGに対し、TLSでアプリケーション層のプロトコルを識別する方式の標準化を要請
- NPNに加えて、ALPN(Application-Layer Protocol Negotiation)という手法が提案され、ALPNを基に進んで行く方向に

NPNとALPN

- NPN:サーバが使えるプロトコルを返し、クライアントが選択
- ALPN:クライアントが使えるプロトコルを送り、サーバが選択





シーケンス図提供元: http://d.hatena.ne.jp/ASnoKaze/20130207/1360249692

ALPNの特徴

- クライアントが使えるプロトコルを送り、サーバが選択
  - サーバが使えるプロトコルをクライアントが送信できる必要がある
- ネゴシエーション結果は暗号化されない
  - 暗号セッション確立後にTLS RenegotiationによってALPNを行えば暗号化された状態でネゴシエーション可能
  - 暗号化されないため、TLS通信上のプロトコルがHTTP/2.0か否か、FWやNAT機器でも判別可

おまけ

JANOG本編で話す予定のことを簡単に

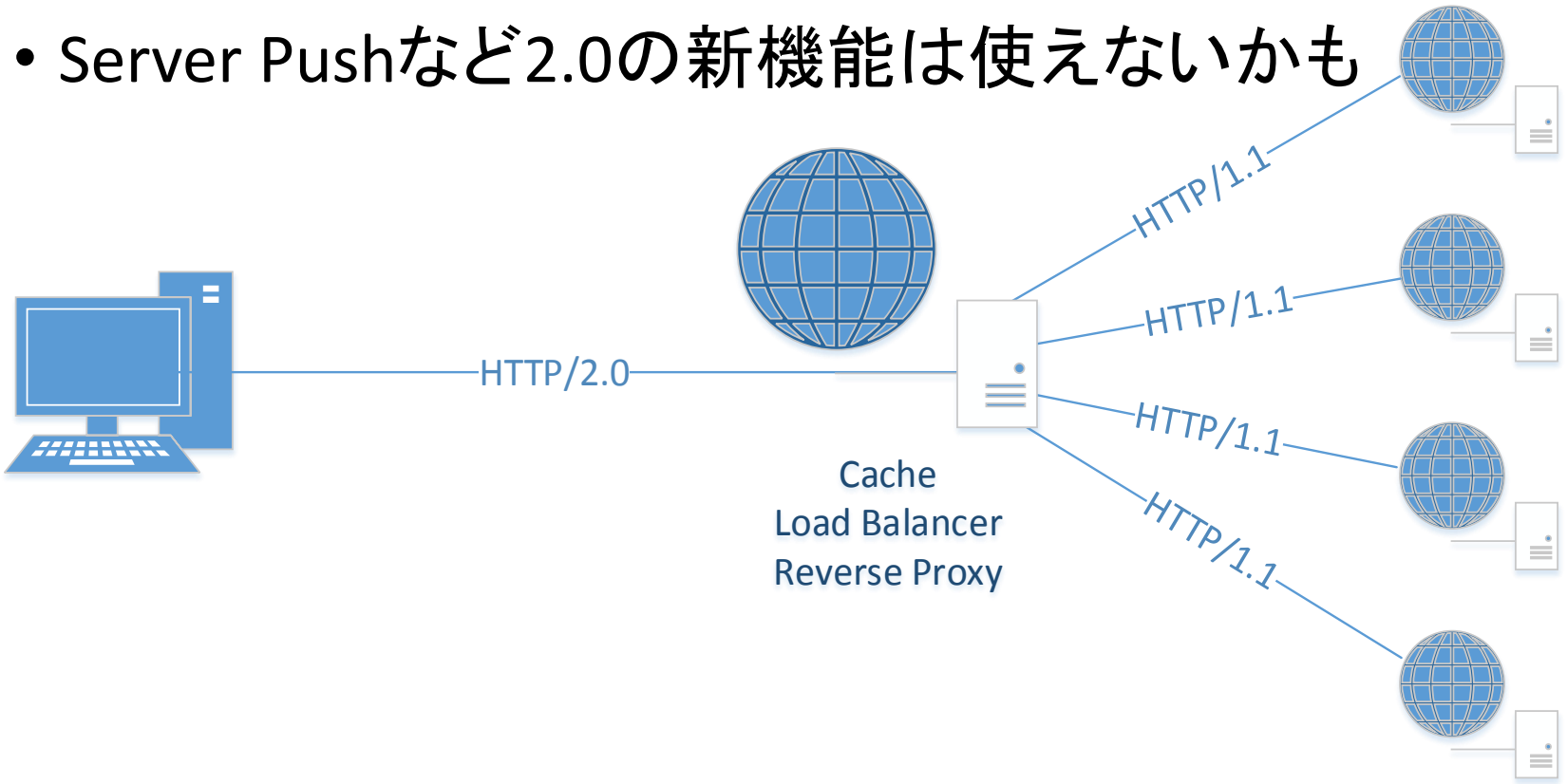
低レイヤの視点から

- バイナリ化・多重化等による可読性低下
  - WiresharkのプラグインやNetcatのようなツール類は適宜作成していく予定 (*)
- CGN等との相性？
  - HTTP/1.1と比べてTCPコネクションの寿命が長いため、NATタイマ等による問題が出るかも？
  - 概ね問題無いと思われる
- 暗号化？
  - 今までのHTTP/HTTPSの関係と同等

*: https://github.com/http2/wg_materials/blob/master/interim-13-01/minutes.md

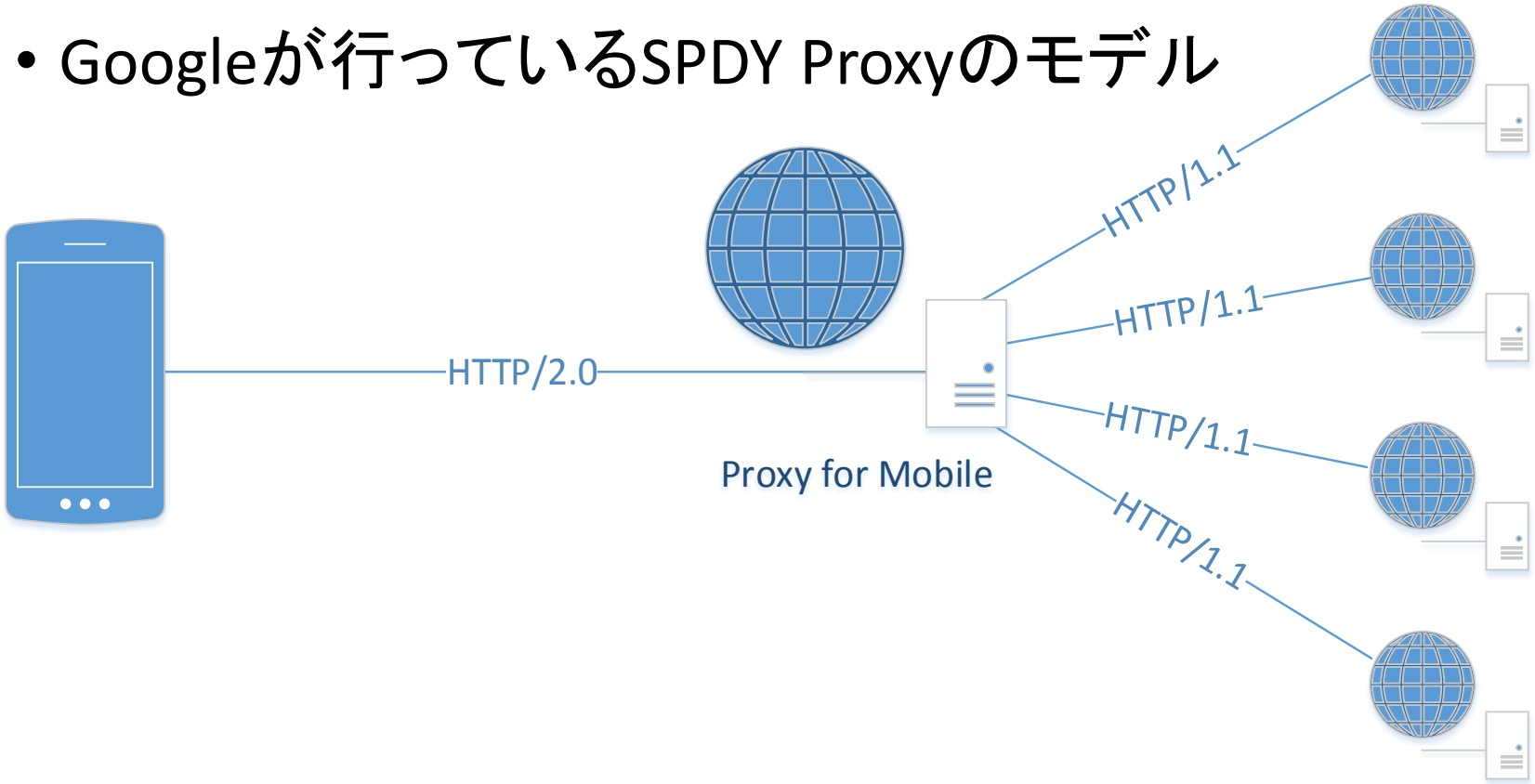
ユーザ・フロントエンド間のみ2.0

- バックエンドのコストはユーザに直接影響しにくい?
- 通信の解析が容易
- Server Pushなど2.0の新機能は使えないかも



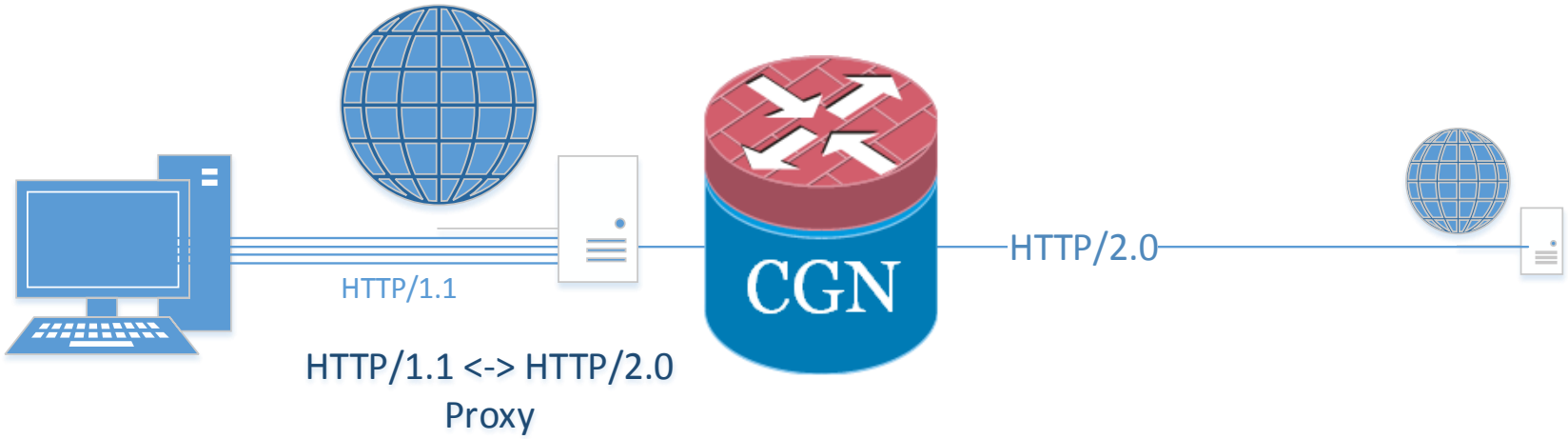
モバイル向け2.0プロクシ

- 高レイテンシなモバイル環境向け
- TCPセッション開始やDNSルックアップのコスト削減
- Googleが行っているSPDY Proxyのモデル



2.0非対応ブラウザ向けプロクシ

- クライアントからHTTP/1.1で受けてHTTP/2.0でリクエスト
- CGNの内側に置けばクライアント・プロクシ間のポート数は問題なし
- NATを通るTCPセッション数削減？



まとめ+α

- HTTP/2.0は昨今のWeb普及に対応するために、効率を上げていく取り組み
- 普及すればTCPポート数やトラフィック、レイテンシ等の削減が期待できる
- 普及するかはまた別の問題
- バイナリ化や多重化等のため、可読性は落ちる
- 一般的なWebサイトで導入するには敷居が高い