http2 explained

Daniel Stenberg

目次

Introduction	1.1
背景	1.2
HTTPの現状確認	1.3
レイテンシーの闇を克服せよ	1.4
もうやめて、HTTP 1.1のライフはゼロよ	1.5
http2のコンセプト	1.6
http2プロトコル	1.7
http2は拡張の夢を見る	1.8
http2化される世界	1.9
Firefoxにおけるhttp2	1.10
Chromiumにおけるhttp2	1.11
curlにおけるhttp2	1.12
http2の次にくるもの	1.13
参考文献	1.14
謝辞	1.15

http2解説

この文書はHTTP/2（RFC 7540）、その背景、コンセプト、プロトコル、既存の実装および未来がどうなるかを記述したものです。

このプロジェクトのサイトは https://daniel.haxx.se/http2/ を見てください。

文書の全ソースコードは https://github.com/bagder/http2-explained を見てください。

貢献するには

改善案をお持ちの方はだれでも大歓迎です。私達はプルリクエストを受け付けていますが、daniel-http2@haxx.se まで改善案をメールで送って頂いても結構です。

/ Daniel Stenberg

1. 背景

この文書はhttp2を技術的にプロトコルレベルで記述したものです。始まりはDanielが2014年4月にストックホルムで行ったプレゼンテーションでした。後に拡張され、全ての詳細や丁寧な説明を含む立派な文書になりました。

RFC 7540は公式なhttp2仕様書で、2015年5月15日に発行されました: https://www.rfc-editor.org/rfc/rfc7540.txt

この文書におけるすべての誤りは私自身のものであり、私の至らなさの致すところです。指摘していただければ次のバージョンで修正します。

この文書では統一的に”http2”という単語でこの新しいプロトコルを呼称していますが、正式な名称はHTTP/2です。読みやすさと言語における収まりの良さのためにこの選択をしました。

1.1 著者

私の名前はDaniel Stenbergです。Mozillaで働いています。私はオープンソースとネットワークの分野で20年以上も様々なプロジェクトで働いています。おそらく私はcurlとlibcurlの開発リーダーとしてよく知られているかもしれません。私はIETF HTTPbisワーキンググループに数年間参加していて、最新のHTTP 1.1に追随するとともに、http2の標準化にも参加しました。

Email: daniel@haxx.se

Twitter: @bagder

Web: daniel.haxx.se

Blog: daniel.haxx.se/blog

1.2 ご協力をお願い致します

間違いや脱字、エラー、明らかな嘘をこの文書に見つけた場合、修正した段落を私に送ってください。修正を取り込みます。私はもちろん助けてくれた人全てに適切な名誉を与えるつもりです。私はこの文書が時とともに改善されていくことを望んでいます。

この文書はhttps://daniel.haxx.se/http2で利用可能です。

1.3 ライセンス

この文書はCreative Commons Attribution 4.0 license:
https://creativecommons.org/licenses/by/4.0/ でライセンスされています。

1.4 履歴

この文書の最初のバージョンは2014年4月25日に発行されました。最近の文書における主たる変更点を以下に示します。

Version 1.13

- Converted the master version of this document to Markdown syntax
- 13: mention more resources, updated links and descriptions
- 12: Updated the QUIC description with reference to draft
- 8.5: refreshed with current numbers
- 3.4: the average is now 40 TCP connections
- 6.4: Updated to reflect what the spec says

Version 1.12

- 1.1: HTTP/2 is now in an official RFC
- 6.5.1: link to the HPACK RFC
- 9.1: mention the Firefox 36+ config switch for http2
- 12.1: Added section about QUIC

Version 1.11

- Lots of language improvements mostly pointed out by friendly contributors
- 8.3.1: mention nginx and Apache httpd specific acitivities

Version 1.10

- 1: the protocol has been “okayed”
- 4.1: refreshed the wording since 2014 is last year
- front: added image and call it “http2 explained” there, fixed link
- 1.4: added document history section
- many spelling and grammar mistakes corrected
- 14: added thanks to bug reporters
- 2.4: (better) labels for the HTTP growth graph
- 6.3: corrected the wagon order in the multiplexed train
- 6.5.1: HPACK draft-12

Version 1.9

- Updated to HTTP/2 draft-17 and HPACK draft-11
- Added section ”10. http2 in Chromium” (== one page longer now)

- Lots of spell fixes
- At 30 implementations now
- 8.5: added some current usage numbers
- 8.3: mention internet explorer too
- 8.3.1 "missing implementations" added
- 8.4.3: mention that TLS also increases success rate

2. HTTPの現状確認

HTTP 1.1はインターネット上で広く何にでも使われるプロトコルになりました。このような世界で優位に立つためプロトコルやインフラストラクチャーに多大な投資がなされてきました。それにより、今日では自ら全く新しいものを作り上げるよりはHTTPの上で行うほうが簡単なのです。

2.1 HTTP 1.1は巨大

HTTPが生まれた時、それはどちらかというと簡素で直截なプロトコルであると受け取られていました。しかし歴史はそれを否定しています。1996年発行のRFC 1945ではHTTP 1.0は60ページの仕様書で定義されていました。HTTP 1.1を定義するRFC 2616は3年後の1999年に発行されましたが、176ページにまで膨れ上がりました。さらに我々がIETFにおいてRFC 2616を更新した際、6冊の文書に分割され（RFC 7230からRFC7235）、合計ではページ数は増加しています。どれをとってもHTTP 1.1は巨大であり、必須ではないたくさんのオプショナルな箇所は言うに及ばず、詳細で細かすぎてわからない規定がてんこ盛りなのです。

2.2 多すぎるオプション

HTTP 1.1の微に入り細に入る規定や後の拡張のためのオプションのお陰で、一つの実装で全てを実装する（「全て」が何を指しているかを定義することすら不可能なのですが）ことがほとんど不可能に近い状況でソフトウエアエコシステムを構築してきたのです。このため、初期にほとんど使われていない機能はほとんど実装されず、それらを実装したとしてもほとんど使われることがなかった、という状況に陥ったのです。

後になってクライアントとサーバーがそういう機能を使おうとすると相互接続性において問題が生じました。HTTPパイプライニングはそのような機能の代表例です。

2.3 TCPの使い方がいまーつ

HTTP 1.1でTCPのすべての能力を使いこなすのは困難でした。HTTPクライアントとサーバーはページロード時間を削減するための解決策を見出すためとても創造的になる必要がありました。

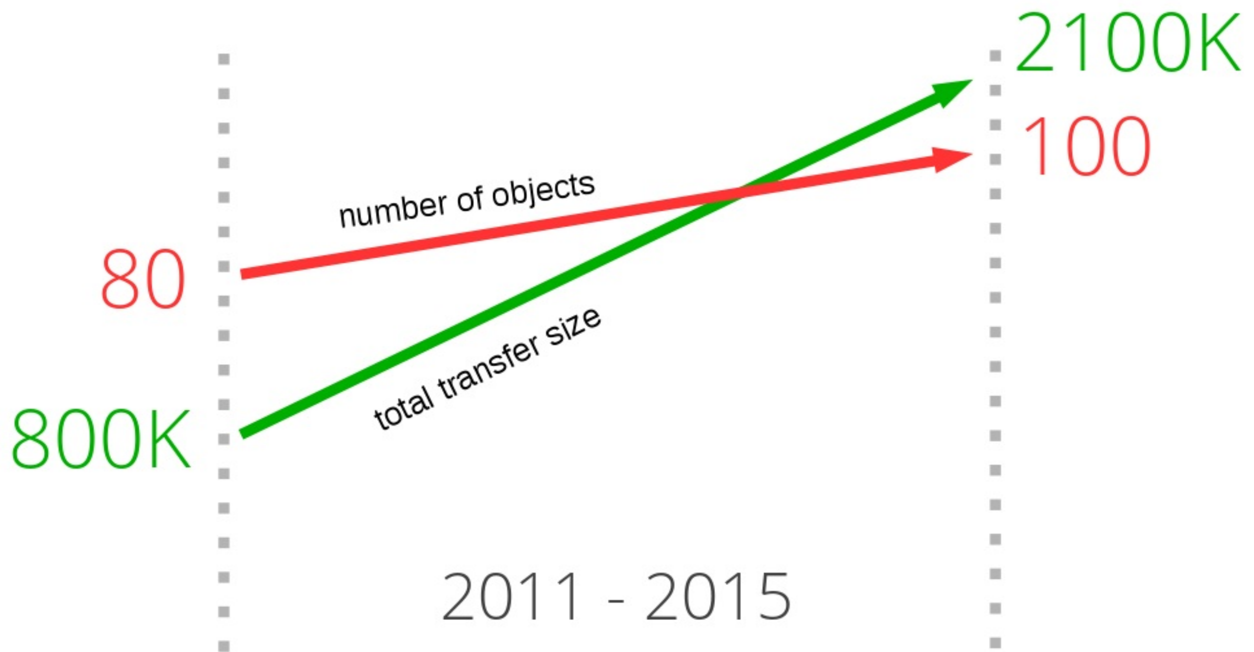
並行して行われてきた他の試みの結果、TCPを置き換えるのは容易ではないと分かったため、我々はTCPとその上のプロトコル両方を向上させることを続けたのです。

単刀直入にいって、TCPは何もしない時間を減らしてデータを送信、受信している時間を増やすほど効率よく使うことができます。続く節ではTCPの不適切な使い方の例をいくつか見ていきます。

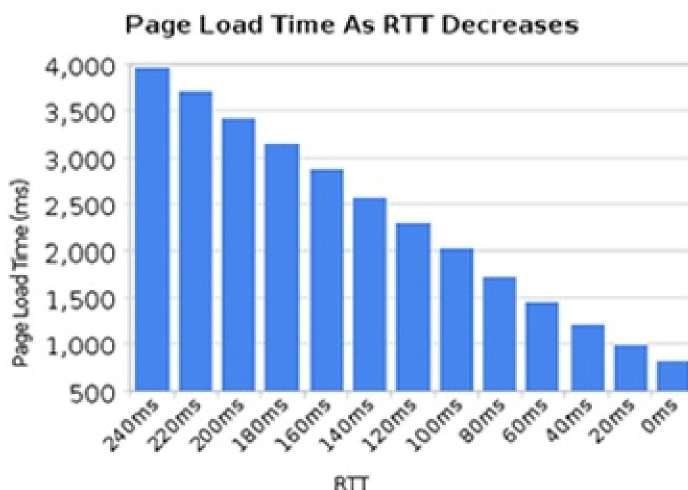
2.4 転送サイズとオブジェクトの数

今日で最も人気のあるいくつかのサイトの傾向とそれらのフロントページをダウンロードするために必要なことを注意深く見た時、明らかなパターンが浮上してきます。年を追うにつれてダウンロードしなければならないデータ量が徐々に増えていて、今日では1.9MBを超えました。この考察においてより重要なことは、平均で100個以上のリソースが1ページを表示するために必要だということなのです。

下の図が示す通り、この傾向はしばらく続いていて、近々変化するという兆しは見受けられません。下の図は世界において最も人気のあるWEBサイトをサーブするときの全体の転送サイズ（緑）、全リクエスト数の平均（赤）の増加、およびそれらが過去4年間でどのように変化したのかを示しています。



2.5 レイテンシーでwebが死ぬ



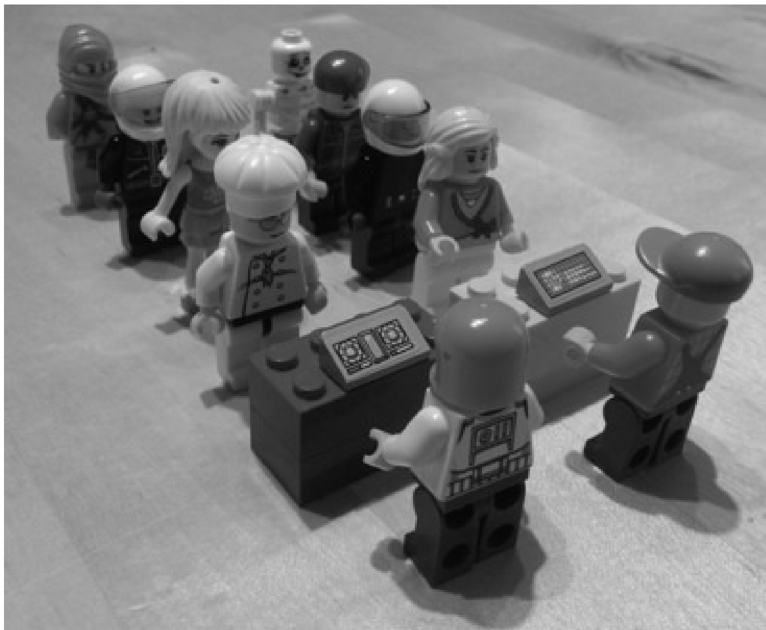
HTTP 1.1はレイテンシーにとても敏感です。HTTPパイプライニングが諸々の問題により大多数のユーザーにおいて無効化されていることもその理由の一つです。

我々は過去数年間において人々が帯域の大幅な増加を享受していることを見てきましたが、レイテンシーの削減においては同程度の向上が見られていません。レイテンシーが大きなリンク、例えば現在のモバイル技術の多くがそうです、においてはすばらしい高帯域の接続が利用できたとしても素早いwebの体験をすることはとても困難なのです。

低レイテンシーが切実に求められている他のユースケースは、映像配信、例えばビデオ会議、ゲームの類、であり、それらは単純に事前に用意したストリームを送信するだけではないのです。

2.6. ヘッドオブライン・ブロッキング

HTTPパイプライニングは前のリクエストの応答を待っている間に次のリクエストを送信するというものです。これは銀行やスーパーマーケットのカウンターで列に並ぶことに似ています。あなたのひとり前の人が素早くすましてくれる人なのか、とても際限なく時間のかかる面倒な人なのかは知るよしもありません。これがヘッドオブライン・ブロッキングです。



確かに注意深く列を選ぶことで回避できるかもしれません。また時には自分で新しい列を作ることもできるでしょう。しかしこのような決断をするということ自体を避けることは不可能であり、一旦決断した後は列を変えることはできないのです。

新しい列を作ることは性能やリソース面において不利であり、比較的少ない数の列数を超えるとスケールしません。完璧な解決策は存在しないのです。

2015年の今日においてもほとんどのデスクトップwebブラウザーはHTTPパイプライニングを既定値で無効にしています。

これに関するさらなる情報については、例えばFirefoxのbugzilla entry 264354で得ることができます。

3. レイテンシーの闇を克服せよ

問題に直面した時はいつも人々は集まって回避策を考えます。中には使いやすい案もあるし、ゴミのように役に立たない案もあります。

3.1 スプライティング

小さな画像をより集めて一つの大きな画像に結合することをスプライティングと言います。JavaScriptやCSSを使って大きな画像から個々の小さな画像を”切り取り”表示させるのです。

サイトはこのトリックを速度向上のために使います。HTTP 1.1では1つの大きな画像をダウンロードするほうが100個の小さな画像をそれぞれダウンロードするよりもはるかに高速です。

もちろんこれには不利なところがあって、それは小さな画像の1個か2個程度しか表示しないサイトの場合です。キャッシュからはすべての画像が一度に削除されてしまうことになり、よく使う画像だけをキャッシュに残すといったことができないのです。

3.2 インライニング

インライニングは個々の画像を送信することを回避する別のトリックで、CSSファイルに埋め込んだdata: URLを使います。これはスプライティングと同じ長所と短所を持っています。

```
.icon1 {
    background: url(data:image/png;base64,<data>) no-repeat;
}

.icon2 {
    background: url(data:image/png;base64,<data>) no-repeat;
}
```

3.3 コンカチネーション

大きなサイトではたくさんのJavaScriptファイルを使っています。開発者はフロントエンドのツールを使って全てのファイルを一つの大きなファイルに結合します。ブラウザーは個々のファイルをダウンロードするのではなく結合した一つのファイルだけをダウンロードするのです。ほんの一部だけをほしい場合でも巨大なファイル全体が送信されるのです。ほんの一部でも変更されると、巨大なファイル全体をリロードする必要があります。

このプラクティスはもちろん開発者にとって厄介なことなのです。

3.4 シャーディング

ここで紹介する最後の性能を向上させるトリックは”シャーディング”と呼ばれているものです。これは基本的にサービスの機能をできるだけ多くのホスト上に分散して配置することです。初見では少し奇妙に見えるかもしれません。しかしその背景には健全な理由があるのです。

初期のHTTP 1.1の仕様書はクライアントが各ホストへ確立できるTCP接続数を最大2個としていました。この仕様を破らないようにするため賢いサイトは単純に新しいホスト名を発明したのです。するとどうでしょう、サイトへの接続数は増え、ページロード時間を削減することができたのです。

時を経てこの制限は削除され、今日においてクライアントはホスト名毎に6-8接続を使っています。しかしこの接続数に制限があるのは変わりないので接続数を増やすためにサイトはこのトリックを使い続けています。先に示したようにオブジェクトの数が増えるにつれて、HTTPを効率よく機能させサイトを高速化するためだけに多くの接続が使われているのです。50以上、時には100を超える接続がひとつのサイトで使われる、ということも珍しいことではありません。httparchive.orgの最近の統計は世界トップ300K個のURLにおいて、平均40（!）個のTCP接続がサイトを表示するために使われていることを示しています。そして徐々に増加している傾向にあるのです。

別の理由は、最近のcookieのサイズはとても大きいので、画像やそれに類するリソースをcookieを使っていない別のホストに置くことです。cookieフリーな画像ホストを使うと、小さいHTTPリクエストをつかって性能を向上させることができる場合があります。

下図はスウェーデンのトップwebサイトの一つをブラウジングしたとき、パケットのトレースがどうなっているか、そしてどのようにリクエストが複数のホスト名に分散されているかを示しています。

200 GET	174.jpg	w.cdn-expressen.se	jpeg	6.14 KB	→ 105 ms
200 GET	174.jpg	y.cdn-expressen.se	jpeg	4.19 KB	→ 172 ms
200		dn-expressen.se	jpeg	4.48 KB	→ 223 ms
200		dn-expressen.se	jpeg	4.58 KB	→ 173 ms
200		dn-expressen.se	jpeg	35.18 KB	→ 56 ms
200		dn-expressen.se	jpeg	12.97 KB	→ 165 ms
200		dn-expressen.se	jpeg	4.83 KB	→ 56 ms
200		dn-expressen.se	jpeg	9.54 KB	→ 228 ms
200		dn-expressen.se	jpeg	182.50 KB	→ 285 ms
200		dn-expressen.se	jpeg	5.66 KB	→ 104 ms
200		dn-expressen.se	jpeg	12.24 KB	→ 287 ms
200		dn-expressen.se	jpeg	6.85 KB	→ 225 ms
200		dn-expressen.se	jpeg	7.50 KB	→ 173 ms
200		dn-expressen.se	gif	2.85 KB	→ 227 ms
200		dn-expressen.se	jpeg	50.87 KB	→ 188 ms
200		dn-expressen.se	jpeg	6.65 KB	→ 55 ms
200 GET	265.jpg	y.cdn-expressen.se	jpeg	6.09 KB	→ 196 ms
200 GET	540.jpg	z.cdn-expressen.se	jpeg	16.14 KB	→ 67 ms
200 GET	540.jpg	w.cdn-expressen.se	jpeg	19.89 KB	→ 112 ms
200 GET	174.jpg	z.cdn-expressen.se	jpeg	5.03 KB	→ 55 ms
200 GET	540.jpg	w.cdn-expressen.se	jpeg	21.27 KB	→ 108 ms
200 GET	540.jpg	x.cdn-expressen.se	jpeg	5.43 KB	→ 237 ms
200 GET	174.jpg	y.cdn-expressen.se	jpeg	6.08 KB	→ 169 ms
200 GET	174.jpg	w.cdn-expressen.se	jpeg	5.62 KB	→ 105 ms
200 GET	540.jpg	x.cdn-expressen.se	jpeg	20.32 KB	→ 241 ms
200 GET	174.jpg	z.cdn-expressen.se	jpeg	6.66 KB	→ 55 ms
200 GET	540.jpg	x.cdn-expressen.se	jpeg	11.13 KB	→ 237 ms
200 GET	265.jpg	w.cdn-expressen.se	jpeg	5.20 KB	→ 111 ms
200 GET	265.jpg	x.cdn-expressen.se	jpeg	6.93 KB	→ 288 ms
200 GET	265.jpg	x.cdn-expressen.se	jpeg	12.09 KB	→ 249 ms
200 GET	265.jpg	z.cdn-expressen.se	jpeg	5.92 KB	→ 167 ms
200 GET	original.jpg	y.cdn-expressen.se	jpeg	64.28 KB	→ 192 ms
200 GET	original.jpg	w.cdn-expressen.se	jpeg	21.88 KB	→ 106 ms
200 GET	540.jpg	w.cdn-expressen.se	jpeg	18.77 KB	→ 112 ms
200 GET	128.jpg	z.cdn-expressen.se	jpeg	3.34 KB	→ 55 ms
200 GET	265.jpg	x.cdn-expressen.se	jpeg	13.00 KB	→ 245 ms
200 GET	265.jpg	y.cdn-expressen.se	jpeg	9.19 KB	→ 194 ms
200 GET	540.jpg	w.cdn-expressen.se	jpeg	13.13 KB	→ 108 ms
200 GET	174.jpg	y.cdn-expressen.se	jpeg	5.66 KB	→ 197 ms
200 GET	174.jpg	z.cdn-expressen.se	jpeg	5.56 KB	→ 55 ms
200 GET	174.jpg	w.cdn-expressen.se	jpeg	5.07 KB	→ 111 ms
200 GET	174.jpg	z.cdn-expressen.se	jpeg	6.16 KB	→ 59 ms
200 GET	174.jpg	y.cdn-expressen.se	jpeg	6.57 KB	→ 210 ms
200 GET	174.jpg	y.cdn-expressen.se	jpeg	4.58 KB	→ 12 ms
200 GET	265.jpg	y.cdn-expressen.se	jpeg	11.49 KB	→ 173 ms

z.cdn-expressen.se
x.cdn-expressen.se
y.cdn-expressen.se
w.cdn-expressen.se
y.cdn-expressen.se
z.cdn-expressen.se
w.cdn-expressen.se

4. もうやめて、HTTP 1.1のライフはゼロよ

今こそ進化したプロトコルを作るべき時です。それは:

1. RTTの影響がより少ない
2. パイプライニングとヘッドオブライン・ブロッキング問題を解決する
3. ホストへの同時接続数を増やす必要性をなくす
4. 既存のインターフェースや全てのコンテンツ、URIフォーマットやスキームを変更しない
5. IETFのHTTPbisワーキンググループで策定する

4.1. IETFとHTTPbisワーキンググループ

インターネットエンジニアリングタスクフォース（IETF）はインターネットで利用する技術の標準化を推進する組織です。ほとんどの場合プロトコルのレベルで作業しています。彼らはRFCの発行で広く知られています。RFCはTCP、DNS、FTPからベストプラクティス、HTTPやその他の派生プロトコルの仕様を定めていて、決して消えることはありません。

IETFの中で、特定の目標に限定して”ワーキンググループ”が組織されます。それらはガイドラインや成果物の範囲を定めた”憲章”を樹立します。誰でもディスカッションや事の成り行きに参加することができます。参加して発言した人は誰でも平等に成果に対して影響を及ぼすことができます。参加者は皆、一人の人間として扱われ、どの会社に所属しているかはほとんど意味を持ちません。

HTTPbisワーキンググループ（名前の由来は後で説明します）はHTTP 1.1仕様書を更新することを目標とし、2007年夏ごろに組織されました。このグループで次世代HTTPの議論が本格的に始まったのは2012年も終盤になってのことでした。HTTP 1.1の更新作業は2014年序盤に完了し、RFC 7230シリーズになりました。

最後のHTTPbisワーキンググループのインターロップミーティングは2014年6月にニューヨーク市で行われました。残りの議題やRFC発行のためのIETFにおける手順は次年度に持ち越されました。

HTTPの分野でのいくつかの大手プレイヤーはワーキンググループのディスカッションやミーティングには参加していませんでした。ここでは特にどの会社だとか、どの製品だとかに言及したりはしませんが、今日のインターネットにおけるアクターの中には、これらの会社が参加しなくてもIETFは正しい仕事をするだろうと自信があるようです。

4.1.1. ”bis”の由来

グループはHTTPbisと命名されていますが、”bis”とは2を意味するラテン語の副詞です。IETFではBisを仕様の更新や第二幕の時に接尾や名前の一部として使うことが一般的です。HTTP 1.1のときもこれに当てはまります。

4.2. http2はSPDYから始まった

SPDYはGoogleによって主導され開発されたプロトコルです。彼らは確かにSPDYをオープンにし、誰でも参加できるようにしました。しかし人気のあるブラウザーとユーザーの多いサービスを稼働させている巨大なサーバー群の両方を自らの制御下に置いている、という事実から恩恵を得ていたということは間違いありません。

HTTPbisグループがhttp2について作業を開始しようと決定した時、SPDYはそれ自身のコンセプトの正しさをすでに証明していました。それはインターネットにデプロイ可能であることを示していましたし、その性能を証明する数値も公開されていました。http2は、SPDY/3ドラフトにほんの少しの文字列置換を施したものをhttp2ドラフト-00として始まったのです。

5. http2のコンセプト

http2は何を成し遂げたのでしょうか。HTTPbisはスコープの境界線をどこに引いたのでしょうか。

これらは極めて厳格であり、チームのイノベーションを厳しく制限するものでした。

- HTTPのパラダイムを保持しなければなりません。TCP上でクライアントがサーバーにリクエストを送信する形のプロトコルなのです。
- http://とhttps:// URLを変更することはできません。新しいスキームを導入することはできません。これらのURLを使うコンテンツは膨大であるため、変更することができないのです。
- HTTP 1サーバーとクライアントはこれからも数十年にわたり存在し続けます。それらをhttp2サーバーにプロキシーする必要があります。
- そして、プロキシーはhttp2の機能をHTTP 1.1クライアントへ1:1で対応させなければなりません。
- プロトコルからオプショナルな部分を削除するか削減する。これは要求ではありませんが、SPDYやGoogleチームからやってきた信条のようなものです。すべてを必須にしてしまえば、今実装できないなため後で罠にはまる、といったようなことが起こりえないのです。
- マイナーバージョンを廃止します。クライアントとサーバーはhttp2に対応するか、対応していないかのどちらかです。プロトコルを拡張あるいは変更したいという要求が出た場合は、それはHTTP 3の出番です。http2にはマイナーバージョンはありません。

5.1. http2と既存のURIスキーム

以前に述べた通り、既存のURIスキームは変更することができないので、http2は既存のものを使わなければなりません。これらはHTTP 1.xで今日使われているため、プロトコルをhttp2へアップグレードする、あるいはサーバーに古いプロトコルではなくてhttp2を使ってくださいとお願いする必要があります。

HTTP 1.1はこのためのUpgradeヘッダーという機構を備えています。古いプロトコルでこのようなリクエストを受けた場合、サーバーが新しプロトコルで応答を返すというものです。しかしラウンドトリップのペナルティを受けます。

SPDYチームはラウンドトリップのペナルティを受け入れることができませんでした。彼らはSPDYをTLS上でのみ実装していたので、ネゴシエーションを大幅に簡略化するTLS拡張を開発しました。この拡張、Next Protocol Negotiation (NPN) を使うと、サーバーは、それがサポートするプロトコルをクライアントへ通知し、クライアントがプロトコルを選択することができます。

5.2. http2とhttps://

http2ではTLS上で適切に振る舞うように随所で配慮がなされました。SPDYはTLSが必須でしたし、http2ではTLSを必須にしようという大きな後押しがありました。しかしコンセンサスが得られず、http2ではTLSは必須ではなくなりました。しかしながら今をリードする2つのwebブラウザー、Mozilla FirefoxとGoogle Chromeの開発者はhttp2をTLS上でのみ実装すると明言しました。

TLSを選択する理由には、ユーザーのプライバシーを尊重するということと、新しいプロトコルを導入する際はTLSを使うほうが高い成功率があったという実測結果の存在がありました。80番ポートを通過するものはすべてHTTP 1.1であるという広く信じられている前提があり、中間装置によっては通信を妨害したり遮断したりして、他のプロトコルが機能することを妨げるのです。

TLSを必須にするかどうかは、メーリングリストやミーティングで根強い反対意見が寄せられました。これは善なのか悪なのか、ということです。これは呪われた議題です。HTTPbis参加者に向かってこの質問をするときは注意してください。

同様にhttp2はTLSを使用する場合の必須暗号化スイートのリストを宣言すべきかどうか、あるいは、使用できないものをブラックリストにすべきか、いやいや、TLS"層"に要求することはせず、すべてTLS WGに任せようではないか、といった白熱した議論がかわされました。最終的にはTLS 1.2以上を必須とし、暗号化スイートに制限を付ける、ということになりました。

5.3. TLSにおけるhttp2ネゴシエーション

Next Protocol Negotiation (NPN) はTLSサーバーとSPDYをネゴシエートするプロトコルです。それは標準化されていなかったので、IETFで議論した結果、Application Layer Protocol Negotiation (ALPN) が生まれました。ALPNはhttp2で使われることになり、SPDYクライアントとサーバーはNPNを使い続けています。

NPNのほうが最初に登場したこと、またALPNが標準化に時間を取られたこともあって、初期のhttp2クライアントとサーバー実装はhttp2をネゴシエートする時に両方の拡張を使うようになりました。またNPNはSPDYで使用されていて多くのサーバーがSPDYとhttp2を両方サポートすることから、NPNとALPNをこれらのサーバーでサポートすることは理にかなっています。

ALPNとNPNの主たる差異はどちらがプロトコルを選択するかということです。ALPNではクライアントがサーバーに優先度の高い順にソートしたプロトコルのリストを渡し、サーバーがその中から選択しますが、NPNではクライアントが最終的な決定をします。

5.4. http2とhttp://

先に述べたとおり、平文HTTP 1.1においてhttp2をネゴシエートするにはサーバーにUpgradeヘッダーを送信します。サーバーがhttp2をサポートするなら、" 101 Switching" ステータスを返し、その接続においては以後http2を使用します。もちろんこのアップグレード手順はネットワークの完全な1ラウンドトリップを必要とします。しかし利点としてはhttp2は永続的接続であり一般的にHTTP 1接続よりも多くの部分を再利用可能です。

いくつかのブラウザーベンダーはこの方法でのhttp2の使用を実装しないと言っていますが、Internet Explorerチームは実装する意思を示していて、またcurlはすでにサポートしています。

6. http2プロトコル

我々がここに至った背景や歴史、政治的な事柄はもう十分でしょう。プロコトルの詳しい仕様について話しましょう。

6.1. バイナリー

http2はバイナリープロトコルです。

この事実を受け入れるまで少し時間を取りましょう。インターネットのプロトコルに関与してきた人なら、衝動的にこの選択について反対し、telnetなどで人間がリクエストを入力できるテキスト／アスキーベースのプロトコルがいかに優れているかを説明しだすことでしょう。

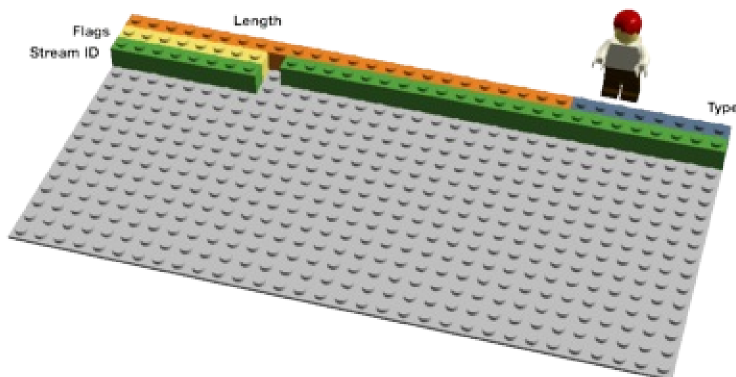
http2はフレーミングを遥かに簡単にするためにバイナリーになりました。フレームの始まりと終わりを判断することは、HTTP 1.1だけでなくテキストベースのプロトコル全般において大変複雑なのです。オプショナルな空白や、同じことを違う方法で書けるという仕様をなくすことで、実装がより簡単になるのです。

また実際のプロトコル部分とフレーミングを分離することも簡単にします。HTTP 1.1ではこれらは一体となっていたのでした。

プロトコルは圧縮をサポートし、ほとんどの場合TLS上で使われるであろうと思われるため、テキストの価値は下がりました。どうせ通信路上では読めないのです。Wiresharkやそれに似たツールを使ってhttp2プロトコルレベルで何が起こっているかを調べるようになればいいのです。

このプロトコルをデバッグには、おそらくcurlのようなツールを使うか、Wiresharkのhttp2ディセクターでネットワークを解析することになるでしょう。

6.2. バイナリーフォーマット



http2はバイナリーフレームを送信します。数種類のフレームタイプがありますがそれらは共通して以下を含んでいます:

長さ、タイプ、フラグ、ストリーム識別子（ID）、フレームペイロード

10個のフレームがhttp2仕様書に定義されていて、その中でもHTTP 1.1の機能に対応づけるための基本的なフレームはDATAとHEADERSです。いくつかのフレームについては後で詳しく述べます。

6.3. 多重化されたストリーム

先のバイナリーフレームフォーマットのセクションで述べたストリームIDはhttp2で送受信されるフレームを”ストリーム”に関連付けます。ストリームは論理的な関連付けです。http2接続では、クライアントとサーバー間で独立した、双方向のフレームの列が送受信されます。

一つのhttp2接続は複数の並行して開かれた状態のストリームを含むことができ、両エンドポイントは複数のストリームからのフレームを、フレーム単位で互い違いに送信することができます。ストリームは一方的に使うこともできるし、クライアントかサーバーで共有され、どちらかによって閉じることができます。ストリーム内でのフレームの順番は意味をもっています。受信者はフレームを受信した順に処理します。

ストリームの多重化は多くのストリームのフレームが同一接続上でフレーム単位で混合されるということを意味します。2つあるいはそれ以上の独立したデータの列車が、一つの列車に結合され、受信側でまた分離されます。ここに2編成の列車があります:

2編成の列車が同じ接続上で多重化されました:

6.4. 優先度と依存関係

各ストリームには優先度（”重み”としても知られています）があり、リソースの制約でサーバーがどのストリームを先に送るか決める時に、どのストリームが重要かをサーバーに知らせます。

クライアントはPRIORITYフレームを使ってサーバーにこのストリームが他のどのストリームに依存するのか指定することができます。これにより、クライアントは”子ストリーム”が”親ストリーム”の完了に依存するような優先度”木”を作ることができます。

優先度や依存関係は動的に変更することができるので、画像がたくさんあるページをユーザーがスクロールしたときにどの画像が最も重要であるかを伝えることや、タブを切り替えたときにフォーカスされる新しいストリームの集合の優先度を上げる、ということができます。

6.5. ヘッダー圧縮

HTTPはステートレスなプロトコルです。つまりサーバーが、以前のリクエストに含まれる情報やメタデータを保存することなく、次のリクエストを処理するためには、必要な情報を毎回送信する必要があります。 http2はこのパラダイムを変えてはいないので、同じことをする必要があります。

これによりHTTPは冗長になります。クライアントが同じサーバーに多くのリソース（webページの画像など）を要求した場合、ほとんど同じようなリクエストが大量に送信されることになります。ほとんど同じものが連続するような時は、圧縮の出番です。

私が先に言及したようにwebページ毎のオブジェクトの数が増加していますが、cookieやリクエストのサイズも同様に年々増加を続けています。Cookieは全てのリクエストに含める必要があり、リクエスト毎にほとんど違いはありません。

HTTP 1.1のリクエストのサイズはとても大きくなってきていて、初期TCPウインドウよりも大きくなる場合があり、サーバーからACKを受信するため完全なラウンドトリップを必要とすることから、リクエスト送信完了までの時間がとても長くなります。これは圧縮の必要性を示唆する理由の一つです。

6.5.1. 圧縮は注意を要する困難な問題

HTTPSとSPDYの圧縮はBREACHとCRIME攻撃に対して脆弱でした。文字列をストリームに挿入し出力がどのように変化するかを観測することで、攻撃者は何が送信されているのか知ることができます。

プロトコルの動的なコンテンツに対する圧縮を、これらの攻撃に対して脆弱ではない方法で行うには、注意深く考える必要があります。これこそHTTPbisチームが行おうとするところのものです。

そこでHPACK、HTTP/2のためのヘッダー圧縮、が誕生しました。名前が示す通り、http2ヘッダーのために生み出されたヘッダー圧縮フォーマットであり、独立したインターネットドラフトで定義されています。この新しいフォーマットは、中間装置に対しヘッダーフィールド単位に圧縮しないように指定するビットや、フレームにパッディングを付け加えるオプションもあいまって、悪用しにくくなっているはずです。

Roberto Peon（HPACKを生み出した人々の中の一人）の言葉です:

> ”HPACKは、仕様に沿う実装が情報を漏洩するのが困難であり、エンコードとデコードが高速で必要なリソースも少なく、 受信側が圧縮コンテキストのサイズを制御でき、プロキシーが再インデックス（プロキシー内部のフロントエンドとバックエンド間の共有状態）でき、ハフマンエンコードされた文字列の比較が高速である、ように設計されています。”

6.6. リセット - 考えを改めましょう

HTTP 1.1の一つの欠点は、HTTPメッセージがContent-Length付きで送信された場合、簡単に停止させることができないということです。殆どの場合（常にではありません）TCP接続を切断して実現しますが、新しいTCP接続を再度確立するという代償を払う必要があります。

よりよい解決方法はメッセージを停止させ、新しいメッセージを開始することです。http2のRST_STREAMフレームを使うとこれが実現できます。これは帯域が無駄に使われてしまうことを防ぎ、接続が切断されてしまうことを回避することに役立ちます。

6.7. サーバープッシュ

これは”キャッシュプッシュ”とも呼ばれている機能です。背後にあるアイデアはこうです。クライアントがリソースXを要求したとき、サーバーはクライアントはほとんどの場合リソースZも必要であると知っている可能性があるから、それをクライアントが要求する前に送信してしまおう。こうすることでクライアントはZをキャッシュに入れておくことができ、必要なときに使うことができます。

サーバープッシュはクライアントが明示的にサーバーに許可を与える必要がある代物であり、許可した場合でも、プッシュされたストリームが必要ないと判断した場合RST_STREAMで即座に閉じることができます。

6.8. フロー制御

http2上のストリームはそれぞれ独立にフローウインドウを持っていて、それはピアがストリームへ送信できるデータ量を制限します。SSHがどのように動いているかご存知なら、それとよく似た様式や背景を持っています。

各ストリームにおいて両エンドポイントはピアに対してどれくらいデータを受信できるか伝えなければなりません。ピアはウインドウが拡張されるまで伝えられたデータ量までしか送信することができません。DATAフレームのみがフロー制御されています。

7. http2は拡張の夢を見る

プロトコルは受信者が不明なフレームタイプを持つフレームを受信した場合、無視することを要求しています。両エンドポイントはホップバイホップで新しいフレームタイプの使用をネゴシエートすることができます。それらのフレームはセッションの状態を変えることが許されておらず、フロー制御されません。

http2に拡張を許すかどうかはプロトコル策定中に賛成と反対の意見の間で揺れ動き長い時間議論されました。ドラフト12の後、振り子は再度振れ、拡張を許すことになりました。

拡張はプロトコルの一部ではなく、基本のプロトコル仕様とは別の文書で定義されます。この時点ですでに基本プロトコルに含める意図をもって2つのフレームタイプが議論されていましたが、おそらく最初の拡張となるでしょう。ここではこの2つのフレームを紹介します。というのはこれらはよく知られているということと、もともと”標準”フレームの扱いだったからです。

7.1. オルタナティブサービス

http2の普及が進むにつれ、HTTP 1.xの時よりもTCP接続がはるかに長く持続するだろうとする理由があります。クライアントは1接続で全て間に合わせなければならないので、その接続は潜在的にかなり長い間開かれている可能性があります。

これはHTTPロードバランサーに影響を与え、サイトがクライアントに別のホストに接続してほしいという状況が生まれる可能性があります。それは性能のためだけではなく、サイトがメンテナンスや似たような理由でダウンするときもそうです。

サーバーはそのような場合Alt-Svcヘッダー（またはhttp2のALTSVCフレーム)を送信し、クライアントにオルタナティブサービスについて伝えます。別のサービス、ホスト、ポート番号を使って、同じコンテンツへ別のルートでアクセスするのです。

クライアントはオルタナティブサービスに非同期的に接続を試行し、うまくいったときだけそれを使うようにします。

7.1.1. 日和見暗号化

Alt-Svcヘッダーによりサーバーはhttp://で配信しているコンテンツをTLS接続でも配信しているとクライアントに伝えることができます。

これはそれなりに議論を生んだ機能です。このような接続は認証されたTLSではなく、”安全”だとはいえず、UIで鍵マークを使うことは出来ません。実際、ユーザーにこれは平文のHTTPではなく、日和見暗号だと伝えるすべはないため、一部の人々は強くこのアイデアに反対しています。

7.2. Blocked

このフレームは、http2のエンドポイントが送信可能なデータがあるがフロー制御によって送信できない時に一度だけ送信します。背景にあるアイデアは、あなたの実装がこのフレームを受信した場合、あなたの実装が間違っている、または帯域を使い切れていないということを知ることができるというものです。

このフレームを拡張として扱うため削除される前のドラフト12からの引用です:

> ” BLOCKEDフレームは実験のためこのドラフトに導入されました。実験結果が有意義なフィードバックを示さない場合、削除される可能性があります。”

8. http2化される世界

http2が普及すると世の中はどうなるのでしょう。そもそも普及するのでしょうか。

8.1. http2は一般の人々にどのような影響を与えるのでしょうか。

http2はまだ広くデプロイされておらず、また使われていません。我々は今後どうなっていくのか自信を持ってここで語ることはできません。我々はSPDYがどのように使われているか見てきました。その経験やその他の過去および現在の実験をもとに推測や計算をすることができます。

http2はネットワークラウンドトリップの回数を削減します。ヘッドオブライン・ブロッキングのジレンマを多重化と不必要なストリームをすぐに捨て去ることにより完全に回避します。

今日もっともシャーディングが多用されているサイトよりも多くの並行ストリームを使用することができます。

優先度をストリームに適切に適用することで、クライアントは重要なデータをあまり重要ではないデータの前に受信することができる可能性が高まります。これらを総合的にみると、高速なページロード、そしてよりレスポンシブなwebサイトを実現できる可能性が高いと言えます。単刀直入にいって、よりよいwebのエクスペリエンスにつながるのです。

どの程度速くなるのか、またどの程度改善するのか、といったことはまだわかりません。まず第一に、この技術はまだ初期の段階であり、これらプロトコルが提供する能力を余すところなく使い切るクライアントとサーバー実装はまだありません。

8.2. http2はweb開発にどのような影響を与えますか？

この何年かでweb開発者とweb開発環境はトリックや道具をたくさんかき集めてきてHTTP 1.1の問題を回避してきました。私がこの文書の最初にhttp2の正当性の証としてそれらの中のいくつかを紹介したことを思い出してください。

ツールや開発者が今日何も考えないで使っているこれらの回避策の多くはおそらくhttp2の性能に悪影響を与える、または、少なくともhttp2のすばらしい能力を余すところなく使いこなすことができない、ということになるでしょう。スプライティングとインライニングは殆どの場合http2ではすべきではありません。シャーディングもhttp2にはおそらくよくない結果となるでしょう。というのはhttp2は少ない接続数から恩恵を得るからです。

ここでの問題はもちろんwebサイトとweb開発者は、少なくとも短期間はHTTP 1.1とhttp2のクライアント両方のために開発とデプロイをする必要があることです。全ユーザーに最高の性能を提供することは2つの異なるフロントエンドの提供なくしては容易ではありません。

この理由だけをとってみても、http2の潜在能力が完全に発揮されるまでは少し時間がかかると私は考えています。

8.3. http2の実装

特定の実装についてこの文書で触れることは、もちろん理にかなったことではありませんし、失敗することは目に見えています。ほんの少し時間が経てば古くなってしまいます。そういうことはせず、私は広い視点での状況を説明することにして、読者には実装のリストを参照していただくことにします。

早くから多くの実装が存在していて、http2の作業中にも増えてきました。この文書の執筆時において40を超える実装がリストに載っており、それらのほとんどは最終版を実装しています。

8.3.1. ブラウザ

Firefoxは最新ドラフトにもっとも速く追随してきたブラウザです。Twitterは最新版に追随しサービスをhttp2で提供しています。Googleは2014年4月頃からGoogleのサービスを提供するテストサーバーでhttp2をサポートしていて、2014年5月から開発版のChromeでhttp2サポートを提供しています。マイクロソフトは次期Internet Explorerとしてhttp2をサポートしたプレビュー版を公開しました。SafariとOperaはhttp2に対応予定であると言っています。

8.3.2. サーバ

既に多くのサーバがhttp2をサポートしています。

人気のあるNginxサーバは2015年9月22日の1.9.5からhttp2を提供しています。 (SPDYモジュールの置き換えが必要で、SPDYモジュールとhttp2の共存はできません。)

Apacheのhttpdサーバは2015年10月9日にリリースされた2.4.17からhttp2モジュール mod_http2 が提供されています。

H2O、Apache Traffic Server、nghttp2、Caddy、LiteSpeedは全てhttp2が使用できます。

8.3.3. その他

curlとlibcurlは平文とTLS両方のhttp2をサポートしています。複数のTLSライブラリで対応を行っています。

Wiresharkはhttp2をサポートしています。http2のネットワークトラフィックを分析するための完璧なツールです。

8.4. http2に対するよくある批判

このプロトコルの開発中、議論は前後しました。もちろんこのプロトコルは完全に間違いであると信じている人も確かにいました。私はよくある批判のいくつかに言及し、それに対する回答を述べたいと思います。

8.4.1. ”プロトコルはGoogleによって作られた”

世界はさらにGoogleに依存または支配されていくことを示唆するような亜種の批判もあります。このプロトコルはここ30年間に開発されたプロトコルと同様の手法でIETFによって開発されました。しかしながら我々はSPDYにおけるGoogleのすばらしい仕事を認めています。それは新しいプロトコルがデプロイできるということを証明しただけでなく、それによりどのような効果が得られるのかを示す数値も提供しました。

Googleが公式に発表したところによると、SPDYとNPNをChromeから2016年に削除し、サーバーもHTTP/2に移行していくことを急ぐということです。

8.4.2. ”ブラウザーだけが得をするプロトコルだ”

これはある意味本当です。http2の開発を裏で牽引した主要なものの一つはHTTPパイプライニングを修正することです。あなたのユースケースがパイプライニングを必要としないのなら、http2はあまり大きな効果をもたらさないかもしれません。これだけがプロトコルにおける改善点ではありませんが、しかし大きな改善点の一つです。

サービスが1接続上の多重化されたストリームがもたらす真の力と能力を理解し始めるとすぐに、我々はより多くのアプリケーションがhttp2を使うことを目にすることでしょう。

小さなREST APIや簡素なHTTP 1.xのプログラムにおけるユースケースはhttp2へ移行しても大きな利点を得られません。しかし、また、ほとんどのユーザーにとってhttp2がもたらす欠点はほとんどないはずです。

8.4.3. ”プロトコルは大規模サイトでしか有用ではない”

そんなことはありません。多重化の能力は、地理的に広範囲に分散していない小さなサイトでありがちな遅延の大きい接続におけるエクスペリエンスを大きく向上させることに寄与するでしょう。大規模サイトはほとんどの場合もうすでに速くてより分散していて短いラウンドトリップ時間をユーザーに提供しています。

8.4.4. ”TLSによって遅くなる”

これはある程度真実だといえます。TLSハンドシェイクは少し余計に時間がかかります。しかしTLSにおいて必要なラウンドトリップを削減する試みが今までもありましたし、現在も進行中です。通信路上で平文ではなくTLSを使うことによるオーバーヘッドは無視できないし、より多くのCPUと電力が同じトラッフィクパターンの平文に比較して使われることになります。それがどのくらいでどの程度の影響力を持つのかについては意見や測定結果次第です。有用な情報源の例としてistlsfastyet.comを参照してください。

電話会社や他のネットワーク事業者、例えばATISオープンWebアライアンス、は、サテライトや機内のようなところでの高速なwebエクスペリエンスを提供するためにキャッシング、圧縮、その他諸々の技術が必要であり、それには平文のトラッフィクが必要だと言っています。http2はTLSを必須としているわけではありませんのでこれ以上議論を複雑にすべきではありません。

多くのインターネットユーザーはTLSが広く使われることを望んでいますし、我々はユーザーのプライバシー保護を促進すべきです。

実験ではTLSを使うと新しいプロトコルを80番ポートで実装するよりも高い成功率があることを示しています。というのは数えきれない中継器が世界に存在していて、80番ポートを通るならHTTP 1.1だと思い込んで、ときどきHTTPにみえることもあるからですが、妨害するからです。

最後に、http2の1接続上に多重化されたストリームの恩恵により、通常のブラウザーのユースケースではTLSハンドシェークの回数が削減されることになりHTTP 1.1を使うHTTPSよりも速くなる可能性もあります。

8.4.5. ”ASCIIではなくなるということは深刻な問題だ”

はい、我々は平文でプロトコルを見ることができるということを好みます。なぜならデバッギングやトレースが容易になるからです。しかしテキストベースのプロトコルはエラーを誘発しやすく、より多くのパースにおける問題を引き起こします。

あなたが本当にバイナリプロトコルを使えないというのなら、HTTP 1.xのTLSや圧縮も扱えなかったはずですが、これらは長い間我々と共にあって使われてきたのです。

8.4.6. ”HTTP 1.1よりも速くない”

これについては、もちろん速いというのが何を意味してどうやって計測するのか議論しなければなりませんが、SPDYの頃から多くのテストが行われていて速いページロードを証明しています（例えば、ワシントン大学の人々による”How Speedy is SPDY?”、Hervé Servyによる”Evaluating the Performance of SPDY-enabled Web Servers”）。このような実験はhttp2でも同様に繰り返されてきました。私はより多くのこのようなテストや実験が公開されることを楽しみにしています。httpwatch.comによる最初の基本的なテストはHTTP/2がその約束を果たしていることを示唆しています。

8.4.7. ”これは階層侵害だ”

本気でそう思っていますか？層は世界的な宗教の触ることができない聖なる柱ではありません。我々がhttp2の開発に際し、グレーゾーンに足を踏み入れた時は、与えられた制約の中で効率のいいプロトコルを作るという意思をもってのことでした。

8.4.8. ”HTTP 1.1の弱点のいくつかは修正されない”

これは本当です。HTTP 1.1のパラダイムを維持するという目的に沿い、いくつかの古いHTTPの機能は残されました。よく使われる、もう見たくもないcookieやauthorizationヘッダーなどがそうです。しかしこれらのパラダイムを維持したお陰で、基本的な部分を完全に書き換えるというアップグレード時の頭のくらくらするような膨大な作業なしにデプロイできるプロトコルを得ました。http2は基本的に新しい層の一つです。

8.5. http2は広く普及するか？

回答するには時期尚早ですが、私なりに推測と予測をしてみたいと思います。

反対論者は普及が開始するのに数十年もかかる新しいプロトコルの例としてこういうでしょう。”IPv6が何をなしてきたかみてみろ”と。しかしながらhttp2はIPv6ではありません。それはTCP上で動作し、普通のHTTPアップグレードメカニズムやポート番号そしてTLS等を使います。ほとんどのルーターやファ

イヤーウォールを変更する必要はありません。

GoogleはSPDYにより、短期間で新しいこのようなプロトコルがデプロイできて、ブラウザーやサーバーの複数の実装間で使うことができるということを証明しました。インターネットで今日SPDYをサポートするサーバーの数は1%の域ですが、これらサーバーが扱うデータの量ははるかに大きいのです。今日におけるいくつかの極めて人気のあるwebサイトはSPDYをサポートしています。

http2はSPDYと基本的に同じパラダイムを有しますが、IETFのプロトコルであることから、より広くデプロイされるだろうと私は考えています。SPDYのデプロメントは”Googleのプロトコル”という汚名により若干縮小気味です。

有名なブラウザーもリリースを控えています。Firefox、Chrome、Safari、Internet Explorer、Operaの代表者たちはhttp2をサポートするブラウザーを出荷すると表明していますし、実際に動く実装を世界に示しています。

Google、Twitter、Facebookといった巨大なサーバー事業者もhttp2を近々サポートする予定です。我々はApache HTTPサーバーやNginxといった人気のあるサーバー実装にhttp2が実装されることを望んでいます。H2Oはとてつもなく高速なHTTPサーバーでhttp2をサポートしていてその潜在能力を示しています。

HAProxy、Squid、Varnishといった巨大プロキシーベンダーはhttp2サポートの意思を示しています。

2015年全体を通して、HTTP/2のトラフィック量は増加しています。9月の初めにおいて、Firefox 40での使用率はHTTP全体の中の13%、HTTPSに限定すると27%でした。Googleは18%のトラフィックがhttp2だったと言っています。Googleは他の新しいプロトコルの実験も同時に行っている（12.1のQUICを参照してください）ので、http2の使用率は低く出てしまうことに注意してください。

9. Firefoxにおけるhttp2

Firefoxはとても緊密にドラフトに追随していて、何ヶ月もの間http2テスト実装を提供してきました。http2プロトコルの開発中、クライアントとサーバーはどのドラフトバージョンを実装しているかについて合意する必要があり、テストを行うときに若干厄介でした。クライアントとサーバーがそれらの実装するプロトコルドラフトが何なのか合意しているか気を付けてください。

9.1. まず最初にhttp2が有効になっているか確かめてください

2015年1月13日にリリースされたFirefox 35から、http2サポートがデフォルトで有効になっています。

アドレスバーに'about:config'と入力し、”network.http.spdy.enabled.http2draft”という名前のオプションを探してください。それがtrueになっているか確認してください。Firefox 36は”network.http.spdy.enabled.http2”という名前の別のオプションを導入し、デフォルトでtrueに設定されています。後者はhttp2”標準”バージョンを制御するのに対し、前者はドラフト時代のhttp2バージョンのネゴシエーションを有効／無効化します。両方ともFirefox 36からデフォルトでtrueになっています。

9.2. TLS限定

Firefoxはhttp2をTLS上でのみ実装することを忘れないでください。Firefoxではhttps://のhttp2をサポートするサイトでのみhttp2は動作します。

9.3. 透過的！

http2が使われているかどうかを示すUIはありません。簡単にはわからないようになっています。確かめる一つの方法は、”Web developer->Network”を開いてレスポンスヘッダーを見て、サーバーが何を返しているかを見ることです。上のスクリーンショットに見るとおり、レスポンスは”HTTP/2.0”であり、Firefoxが”X-Firefox-Spdy:”という独自ヘッダーを挿入します。

ネットワークツールで見ることができるヘッダーはhttp2のバイナリーフォーマットから古いHTTP 1.xスタイルのヘッダーに変換されています。

9.4. http2の使用を可視化する

サイトがhttp2を使用しているかどうかの可視化を手伝いするFirefoxプラグインがあります。それらの一つは”HTTP/2 and SPDY Indicator”です。

10. Chromiumにおけるhttp2

Chromiumチームはhttp2を実装していて、長い間dev、betaチャンネルでそのサポートを行っています。2015年1月27日にリリースされたChrome 40から、一部のユーザーにおいてhttp2がデフォルトで有効になりました。その数は最初は少なく設定されていましたが、時とともに徐々に増加しました。

SPDYサポートは削除される予定です。2015年2月のブログでの発表によると:

> ”ChromeはSPDYをChrome 6からサポートしてきました。しかしそのほとんどの恩恵はHTTP/2からも得られることから、さようならをすることに決めました。SPDYを2016年の早い時期に削除する予定です。”

10.1. まず最初にhttp2が有効になっているか確かめてください

ブラウザーのアドレスバーに”chrome://flags/#enable-spdy4”と入力し、まだ有効になっていない場合は、”enable”をクリックします。

10.2. TLS限定

Chromeはhttp2をTLS上でのみ実装することを忘れないでください。Chromeではhttps://のhttp2をサポートするサイトでのみhttp2は動作します。

10.3. HTTP/2の使用を可視化する

サイトがhttp2を使用しているかどうかの可視化を手伝いするChromeプラグインがあります。それらの一つは”HTTP/2 and SPDY Indicator”です。

10.4. QUIC

現在行われているChromeによるQUIC（12.1を参照）の試験が、HTTP/2の使用率を幾分下げています。

11. curlにおけるhttp2

curlプロジェクトは試験的にhttp2のサポートを2013年9月から行っています。

curlの精神に則り、我々はできるかぎり全てのhttp2の機能を提供する予定です。curlはしばしばテストツール、そしてwebサイトをいろいろと弄くる開発者の手段として使われるので、http2でもこの伝統を引き継ぐ予定です。

curlは第三者のライブラリnghttp2をhttp2のフレームレイヤーの実装に使用しています。curlにはnghttp2 1.0以降が必要です。

curlとlibcurlは、Linuxのディストリビューションからインストールした場合、まだ必ずしもHTTP/2プロトコルがサポートされているわけではないことに注意してください。

11.1. HTTP 1.xそっくり

curlは内部的に受信したhttp2ヘッダーをHTTP 1.xスタイルのヘッダーに変換してユーザーに提示するので、既存のHTTPと同じように見えます。これにより既存のcurlとHTTPの使用からの移行が容易になります。送信するヘッダーについても同様です。HTTP 1.xスタイルでヘッダーをcurlに伝えると、http2サーバーと通信するときには自動で変換されます。これによりユーザーは特定のHTTPバージョンが使われているのかどうかといったことに気を使わなくて済むのです。

11.2. 安全ではない平文

curlはUpgradeヘッダーでhttp2を標準のTCP上でサポートしています。あなたがHTTPリクエストでHTTP/2を要求した場合、curlはサーバーに可能なら接続をhttp2にアップグレードするように依頼します。

11.3. TLSとそのライブラリ

curlは多くのTLSライブラリをサポートしていて、これはhttp2サポートでも同様です。TLSにおけるhttp2サポートの問題点はALPNのサポート、そしてそれほど重要ではないですがNPNのサポートです。

最近のOpenSSLまたはNSSとともにcurlをビルドしてALPNとNPN両方のサポートを手に入れることができます。GnuTLSやPolarSSLの場合、ALPNが利用できますが、NPNは利用できません。

11.4. コマンドラインでの使用

curlにhttp2を使うように指示するには、平文、TLSに関係なく、 `--http2` オプションを使います（” ダッシュ ダッシュ http2”）。curlはまだデフォルトがHTTP 1.1であり、http2を使う場合は追加のオプションが必要なのです。

11.5. libcurlのオプション

11.5.1 HTTP/2を有効にする

アプリケーションではhttps://やhttp:// URLを今までどおり使いますが、http2を使うにはcurl_easy_setoptの `CURLOPT_HTTP_VERSION` オプションを `CURL_HTTP_VERSION_2` にします。こうすることで出来る限りhttp2を使うようになりますが、それができない場合はHTTP 1.1が使われます。

11.5.2 多重化

libcurlは既存の振る舞いを維持しようとするので、HTTP/2の多重化をアプリケーションで有効にするにはCURLMOPT_PIPELININGオプションを使います。このオプションを使わない場合、今まで同様に接続あたりの同時リクエスト数は1になります。

もうひとつ注意してほしいことは、multiインターフェースをつかって複数の転送を同時にlibcurlで行う場合、複数の接続が使われることになります。libcurlを少し待たせて同じ接続にすべての転送を多重化するには、待たせる転送に対してCURLOPT_PIPEWAITオプションを使います。

11.5.3 サーバープッシュ

libcurl 7.44.0以降はHTTP/2サーバープッシュをサポートしています。この機能を使うにはCURLMOPT_PUSHFUNCTIONオプションを使ってプッシュコールバックをセットします。プッシュがアプリケーションによって受け入れられた場合、新しいCURL easy handleが作成されて、他の転送と同様にコンテンツを受信します。

12. http2の次にくるもの

多くの困難な決断と妥協がhttp2ではなされました。http2がデプロイされれば、次のプロトコルリビジョンへのアップグレードのための基礎になります。また異なる複数のバージョンを同時に処理できる概念やインフラストラクチャーをもたらします。おそらく新しいものを導入するときに古いものをすべて捨て去る必要がなくなることでしょう。

http2は、HTTP 1とhttp2間の通信をプロキシーできるようにしたいという願望のためHTTP 1の”遺産”の多くを引き継いでいます。遺産の中のいくつかはさらなる発展や発明を妨げます。もしかするとHTTP 3ではそれらを捨て去ることになるかもしれませんね。

あなたがHTTPで足りないと思っていることはありますか？

12.1. QUIC

GoogleのQUIC(Quick UDP Internet Connections)は興味深い実験です。それはSPDYのときと同じようなスタイルと精神で行われています。QUICはTCP + TLS + HTTP/2の代替品でありUDPを使って実装されます。

QUICは接続の作成を遥かに少ない遅延で行えます。HTTP/2ではパケットロスにより全ストリームがブロックされましたが、QUICでは対象のストリームだけがブロックされるだけですみます。別のネットワークインターフェースをまたいだ接続の維持も可能にします。つまりMPTCPが解決しようとしている問題の領域までカバーしているのです。

QUICは現時点ではGoogleによってChromeとGoogleサーバーにだけ実装されています。コードは簡単に再利用できる形にはなっていません。libquicというプロジェクトがそれを実現しようとしています。プロトコルはドラフトとしてIETFトランスポートワーキンググループへ提出されました。

13. 参考文献

この文書の内容または技術的詳細が薄いと考えるなら、あなたの好奇心を満足させる参考文献をここに紹介します:

- HTTPbisメーリングリストとアーカイブ: https://lists.w3.org/Archives/Public/ietf-http-wg/
- HTML化されたhttp2仕様書: https://httpwg.github.io/specs/rfc7540.html
- Firefoxのhttp2ネットワーキングに関する詳細: https://wiki.mozilla.org/Networking/http2
- curlのhttp2実装に関する詳細: https://curl.haxx.se/docs/http2.html
- http2 webサイト: https://http2.github.io/ and perhaps in particular the FAQ: https://http2.github.io/faq/
- Ilya Grigorikの著書"High Performance Browser Networking"のHTTP/2の章: https://hpbn.co/http2/

14. 謝辞

発想とパッケージのフォーマットのレゴ画像は、Mark Nottingham氏から。

HTTPトレンドデータは https://httparchive.org/ から。

RTTのグラフはMike Belshe氏のプレゼンテーションから。

ヘッドオブラインの絵を作成するためにレゴのおもちゃを私に貸してくれた私の子供たちAgnesとRex
に。

レビューとフィードバックをくれた友人たちに: Kjell Ericson、Bjorn Reese、Linus Swälas and
Anthony Bryanに。あなたの助けには非常に感謝しています。お陰で文書がすばらしくよくなりました。

執筆中にいろいろな段階で、バグレポートや文書の改善に協力してくれた友人たちに: Mikael Olsson、
Remi Gacogne、Benjamin Kircher、saivlis、florin-andrei-tp、Brett Anthoine、Nick Parlante、
Matthew King、Nicolas Peels、Jon Forrest、sbrickey、Marcin Olak、Gary Rowe、Ben Frain、Mats
Linander、Raul Siles、Alex Lee、Richard Moore