

http2

История, протокол, реализации и будущее
<http://daniel.haxx.se/http2/>

Даниэль Штенберг

Содержимое

1.История.....	4
1.1.Автор.....	4
1.2.Помогите!.....	4
1.3.Лицензия.....	4
2.HTTP сегодня.....	5
2.1.HTTP 1.1 огромен.....	5
2.2.Мир опций.....	5
2.3.Неполноценное использование TCP.....	5
2.4.Размер передачи и число объектов.....	6
2.5.Задержка убивает.....	6
2.6.Блокировка начала очереди.....	7
3.Шаги, предпринятые для преодоления задержки.....	8
3.1.Создание спрайтов.....	8
3.2.Встраивание.....	8
3.3.Объединение.....	8
3.4.Шардинг.....	9
4.Обновление HTTP.....	11
4.1.IETF и рабочая группа HTTPbis.....	11
4.1.1.Суффикс «bis».....	12
4.2.http2 начался со SPDY.....	12
5.Концепция http2.....	13
5.1.http2 для существующих URI схем.....	13
5.2.http2 для https://.....	14
5.3.http2 согласование поверх TLS.....	14
5.4.http2 для http://.....	15
6.Протокол http2.....	16
6.1.Бинарный протокол.....	16
6.2.Бинарный формат.....	16
6.3.Мультиплексирование потоков.....	17
6.4.Приоритеты и зависимости.....	18
6.5.Сжатие заголовков.....	18
6.5.1.Сжатие это непростая тема.....	19
6.6.Reset (сброс) — передумал.....	19
6.7.Server push (посылка сервера).....	19
6.8.Управление потоком.....	20
7.Расширения.....	20
7.1.Альтернативные сервисы.....	20
7.1.1.Опportunистический TLS.....	21
7.2.Блокировка.....	21
8.Мир http2.....	22
8.1.Как http2 повлияет на обычных людей?.....	22
8.2.Как http2 повлияет на веб-разработку?.....	22
8.3.http2 реализации.....	23
8.4.Типичная критика http2.....	23
8.4.1.“Протокол спроектирован и сделан в Google”.....	24

8.4.2. "Протокол полезен только для браузеров и больших сервисов"	24
8.4.3. "Использование TLS делает его медленным"	24
8.4.4. "Не ASCII-протокол всё портит"	25
8.4.5. "Он не быстрее, чем HTTP/1.1"	25
8.4.6. "Он содержит нарушения независимости слоёв"	25
8.4.7. "Не исправлены некоторые дефекты HTTP/1.1"	25
8.5. Станет ли http2 широко распространён?	26
9. http2 в Firefox	27
9.1. Сначала включите это	27
9.2. Только TLS	27
9.3. Прозрачно!	27
10. http2 в curl	29
10.1. Выглядит как HTTP 1.x	29
10.2. Простой текст	29
10.3. TLS и выбор библиотеки	29
10.4. Использование в командной строке	29
10.5. Опции libcurl	29
11. После http2	31
12. Дальнейшее чтение	32
13. Благодарности	33

1. История

Это документ, описывающий http2 с позиции технического и протокольного уровня. Первоначально он появился как презентация, которую я представлял в Стокгольме в апреле 2014 года, которая затем была конвертирована и расширена в полноценный документ с деталями и надлежащими пояснениями.

На момент написания (21 января 2015), окончательная спецификация http2 не завершена и не выпущена. Текущая версия черновика называется draft-16¹, и вероятнее всего он станет RFC – последний этап в IETF для правки документа пройден. Данный документ описывает текущую ситуацию, которая может измениться или не измениться в окончательной спецификации. Все ошибки в данном документе – мои собственные, появившиеся по моей вине. Пожалуйста сообщите мне о них и я выпущу обновление с исправлениями.

Версия этого документа – 1.9.

1.1. Автор

Меня зовут Даниэль Штенберг и я работаю в Mozilla. Открытым программным обеспечением и сетями я занимаюсь уже более двадцати лет в различных проектах. Вероятно, я наиболее известен, как основной разработчик curl и libcurl. Многие годы я был вовлечён в рабочую группу IETF HTTPbis и работал как над поддержкой HTTP 1.1, для соответствия новейшим требованиям, так и работой над стандартизацией http2.

Email: daniel@haxx.se

Twitter: [@bagder](https://twitter.com/bagder)

Web: daniel.haxx.se

Blog: daniel.haxx.se/blog

1.2. Помогите!

Если вы обнаружили опечатки, упущения, ошибки и явную ложь в этом документе, пожалуйста отправьте мне исправленную версию параграфа и я выпущу исправленную версию. Я должным образом отмечу всех, кто помог! Надеюсь, что со временем получится сделать текст лучше.

Этот документ доступен по ссылке <http://daniel.haxx.se/http2>

1.3. Лицензия

Этот документ лицензируется под лицензией Creative Commons Attribution 4.0: <http://creativecommons.org/licenses/by/4.0/>



1 <http://tools.ietf.org/html/draft-ietf-httpbis-http2-16>

2. HTTP сегодня

HTTP 1.1 стал протоколом, который используется поистине для всего в Интернете. Огромные инвестиции были вложены в протоколы и инфраструктуру, которые теперь извлекают из этого прибыль. Дошло до того, что сегодня зачастую проще запустить что-либо поверх HTTP, чем создавать что-то новое вместо него.

2.1. HTTP 1.1 огромен

Когда HTTP был создан и выпущен в мир, он, вероятно, воспринимался скорее как простой и прямолинейный протокол, но время показало, что это не так. HTTP 1.0 в RFC 1945 – это 60 страниц спецификации, выпущенной в 1996. RFC 2616, который описывал HTTP 1.1, был выпущен лишь тремя годами позднее в 1999 и значительно разросся до 176 страниц. Кроме того, когда мы в IETF работали над обновлением спецификации, она была разбита на *шесть* документов с ещё большим числом страниц в итоге (что вылилось в RFC 7230 и прочие связанные спецификации). Без сомнений, HTTP 1.1 большой и включает мириады деталей, тонкостей и в не меньшей степени опциональных разделов.

2.2. Мир опций

Природа HTTP 1.1, заключённая в наличии большого числа мелких деталей и опций, доступных для последующего изменения, вырастила экосистему программ, где нет ни одной реализации, которая бы воплотила всё – и, на самом деле, невозможно точно сказать, что представляет из себя это «всё». Что привело к ситуации, когда возможности, которые первоначально мало использовались появлялись лишь в небольшом числе реализаций, и те кто их реализовывал после наблюдали незначительное их использование.

Позже это вызывало проблемы в совместимости, когда клиенты и сервера начали активнее использовать подобные возможности. Конвейерная обработка HTTP (*HTTP pipelining*) – это один из показательных примеров подобных возможностей.

2.3. Неполноценное использование TCP

HTTP 1.1 прошёл трудный путь, чтобы по настоящему воспользоваться всей мощностью и производительностью, которую даёт TCP. HTTP-клиенты и браузеры должны быть по-настоящему изобретательными, чтобы найти способы для уменьшения времени загрузки страницы.

Прочие эксперименты, которые параллельно велись в течении многих лет, также подтверждали, что TCP не так просто заменить, и поэтому мы продолжаем работать над улучшением как TCP, так и протоколов, работающих поверх него.

TCP можно легко начать использовать полноценно, чтобы избежать пауз или периодов времени, которые могли быть использованы для отправки или приёма большого количества данных. Последующие главы осветят некоторые из этих

недостатков использования.

2.4. Размер передачи и число объектов

Когда смотришь на тенденции развития некоторых наиболее популярных на сегодня сайтов и сравниваешь сколько занимает времени загрузка их главной страницы, тенденции становятся очевидными. За последние несколько лет количество данных, которые требуется передать постепенно выросло до отметки 1,9Мб и выше, но что наиболее важно для нас в этом контексте, так это число объектов, которое теперь в среднем равняется ста. Сто объектов необходимо загрузить, чтобы отобразить всю страницу целиком.

Как показывает график, тренд был растущим, но позднее нет никаких признаков, что он будет дальше меняться.



данные httparchive.org

2.5. Задержка убивает

HTTP 1.1 очень чувствителен к задержкам, частично из-за того, что в конвейерной передаче HTTP по-прежнему хватает проблем и она отключена у подавляющего числа пользователей.

В то время, как все мы наблюдали значительное увеличение пропускной полосы у пользователей за последние несколько лет, мы не видели

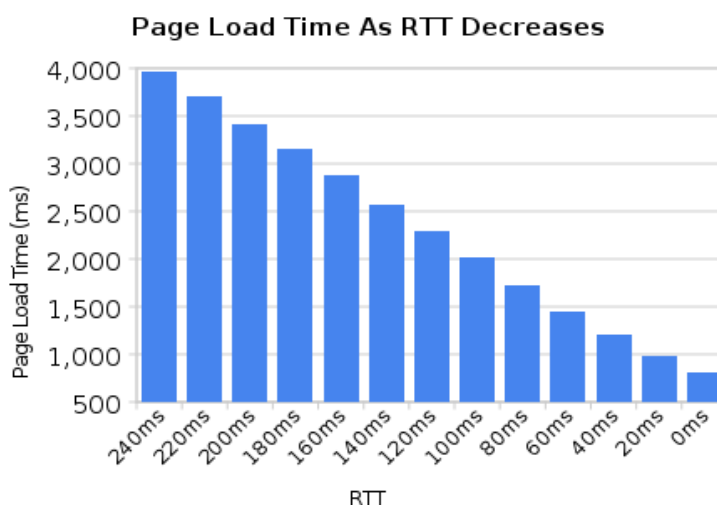


Рисунок 1: Время между отправкой запроса и получением ответа (RTT) влияет на время загрузки страницы

подобного уровня снижения задержки. Каналы с высокой задержкой, как у многих современных мобильных технологий, значительно снижают ощущение хорошей и быстрой веб-навигации, даже если у вас имеется действительно высокоскоростное подключение.

Другой пример, когда действительно требуется низкая задержка, это некоторые виды видео, такие как видео-конференция, игры и подобное, где требуется передать не только заранее созданный поток.

2.6. Блокировка начала очереди

Конвейерная передача HTTP – это способ отправки очередного запроса, уже ожидая ответ на предыдущий запрос. Это похоже на очередь к кассиру в супермаркете или банке. Вы не знаете, что за люди перед вами: быстрые клиенты или надоедливые персоны, которым потребуется бесконечное время, чтобы завершить обслуживание – блокировка начала очереди.

Безусловно вы можете тщательно выбирать очередь и в итоге выбрать ту, которую посчитаете правильной, а иногда вы можете создать свою собственную очередь, но, в конце концов, вы не сможете избежать принятия решения и однажды выбрав очередь, вы не можете её сменить.

Создание новой очереди связано с производительностью и рас платой ресурсами, и не может масштабироваться за пределы небольшого числа очередей. Для этой задачи нет идеального решения.



Даже сегодня, в 2014 году, большинство веб-браузеров на десктопах поставляются с отключенным конвейером HTTP по умолчанию.

Дополнительные сведения по этой проблеме могут быть найдены в баг-трекере Firefox под номером 264354².

2 https://bugzilla.mozilla.org/show_bug.cgi?id=264354

3. Шаги, предпринятые для преодоления задержки

Как обычно, когда люди сталкиваются с ошибками, они объединяются для поиска путей обхода. Некоторые пути обхода искусные и полезные, некоторые просто ужасающие костыли.

3.1. Создание спрайтов

Создание спрайтов – это термин, который часто используется для описания действия, когда вы собираете множество маленьких изображений в одно большое. Затем используете javascript или CSS для «нарезки» частей большого изображения для отображения маленьких картинок.

Сайт использует эту уловку для ускорения. Получение одного большого запроса значительно быстрее в HTTP 1.1, чем получение ста отдельных маленьких картинок.



Конечно, это имеет свои недостатки для тех страниц сайта, которым требуется лишь одна или две маленькие картинки. Это также выбрасывает все картинки из кэша одновременно, вместо того, чтобы, возможно, оставить часть наиболее используемых.

3.2. Встраивание

Встраивание – это ещё одна уловка для избежания отправки отдельных изображений, использование вместо этого data – URL, встроенный в CSS файл. Это имеет те же преимущества и недостатки, что и случай со спрайтами.

```
.icon1 {  
    background: url(data:image/png;base64,<data>) no-repeat;  
}  
  
.icon2 {  
    background: url(data:image/png;base64,<data>) no-repeat;  
}
```

3.3. Объединение

Также как и в двух предыдущих случаях, на сегодняшний день большие сайты могут иметь и множество javascript файлов. Утилиты разработчиков позволяют

объединить все эти файлы в один огромный ком, чтобы браузер получил один файл вместо множества маленьких. Большое число данных отправляется, тогда лишь как небольшой фрагмент реально требуется. Излишнее большое количество данных требуется перезагрузить, когда потребуются сделать изменение.

Раздражение разработчиков и принуждение к выполнению этих требований – это, конечно, «просто» боль для причастных людей и не отображено ни в каких графиках производительности.

3.4. Шардинг

Заключительный трюк, который я упомяну, применяемый владельцами сайтов для улучшения загрузки в браузерах, часто называют «шардингом». Это в основном означает рассредоточение вашего сервиса по максимально возможному числу различных хостов. На первый взгляд это звучит безумно, но на это есть простая причина!

Первоначально HTTP разрешал использовать клиенту максимум два TCP соединения на каждый хост. Таким образом, чтобы не нарушать спецификацию продвинутое сайты просто придумывали новые имена хостов и вуаля, вы можете получить большее число соединений для вашего сайта и сократить время загрузки страницы.

Со временем, это ограничение было убрано из спецификации и сегодня клиенты используют 6-8 соединений на хост, но по-прежнему имеют ограничение, поэтому сайты продолжают технику увеличения числа соединений. По мере увеличения числа объектов, как я уже показал ранее, большое число соединений стало использоваться просто чтобы убедиться, что HTTP справляется хорошо и делает сайт быстрее. Не является необычным для сайтов использование более 50 или даже 100 и больше соединений для одного сайта при помощи данной техники. Последняя статистика от httparchive.org показывает, что топ 300 000 URL'ов в мире требует в среднем 38(!) TCP-соединений для отображения сайта, а тренд говорит о том, что это число по-прежнему медленно растёт со временем.

Ещё одна причина шардинга – это размещение изображений и подобных ресурсов на отдельных хостах, которые не используют cookie, поскольку cookie на сегодняшний день могут быть значительного размера. Используя хосты изображений без cookie вы можете увеличить производительность просто за счёт значительно меньших HTTP-запросов!

Рисунок ниже показывает как выглядит запись пакетов при просмотре одного топ веб-сайта Швеции и как запросы распределяются по нескольким хостам.

GET	/	www.aftonbladet.se	html	205.71 KB	→ 32 ms
GET	general.css?57785785	www.aftonbladet.se	css	193.22 KB	→ 10 ms
GET	d-head.js?57785785	www.aftonbladet.se	js	190.40 KB	→ 20 ms
GET	aftonbladet.gif	gfx.aftonbladet-cdn.se	gif	6.33 KB	→ 44 ms
GET	sanna_lundell_white.jpg	gfx.aftonbladet-cdn.se	jpeg	3.01 KB	→ 44 ms
GET	pappa01.jpg	gfx2.aftonbladet-cdn.se	jpeg	18.66 KB	→ 228 ms
GET	forskare.jpg	gfx.aftonbladet-cdn.se	jpeg	6.88 KB	→ 44 ms
GET	story26jan.jpg	gfx.aftonbladet-cdn.se	jpeg	2.66 KB	→ 45 ms
GET	osbalk.jpg	gfx2.aftonbladet-cdn.se	jpeg	4.11 KB	→ 238 ms
GET	Bank.jpg	gfx2.aftonbladet-cdn.se	jpeg	16.09 KB	→ 247 ms
GET	Tatu.JPG	gfx.aftonbladet-cdn.se	jpeg	7.38 KB	→ 44 ms
GET	sve8.jpg	gfx.aftonbladet-cdn.se	jpeg	7.39 KB	→ 45 ms
GET	otto-tvaa.jpg	gfx.aftonbladet-cdn.se	jpeg	6.71 KB	→ 46 ms
GET	ASTridTT.JPG	gfx.aftonbladet-cdn.se	jpeg	6.52 KB	→ 66 ms
GET	nimoy.jpg	gfx.aftonbladet-cdn.se	jpeg	7.87 KB	→ 67 ms
GET	PSYK0\$.jpg	gfx2.aftonbladet-cdn.se	jpeg	18.28 KB	→ 257 ms
GET	parislgh.jpg	gfx2.aftonbladet-cdn.se	jpeg	21.89 KB	→ 267 ms
GET	ProjectRunwayvinnare.jpg	gfx2.aftonbladet-cdn.se	jpeg	7.83 KB	→ 276 ms
GET	par_strand-opt.jpg	gfx2.aftonbladet-cdn.se	jpeg	5.72 KB	→ 286 ms
GET	mack.jpg	gfx2.aftonbladet-cdn.se	jpeg	16.27 KB	→ 299 ms
GET	skildaVpuffmitt.jpg	gfx1.aftonbladet-cdn.se	jpeg	4.49 KB	→ 144 ms
GET	socker_ny.jpg	gfx1.aftonbladet-cdn.se	jpeg	3.79 KB	→ 153 ms
GET	andreetta.jpg	gfx1.aftonbladet-cdn.se	jpeg	5.53 KB	→ 163 ms
GET	skidor_50.gif	gfx.aftonbladet-cdn.se	gif	3.20 KB	→ 70 ms

4. Обновление HTTP

А не было бы лучше сделать усовершенствованный протокол? Который бы включал в себя следующее...

1. Создать протокол, который был бы менее чувствителен к RTT
2. Исправить конвейерную обработку и проблему блокировки начала очереди
3. Остановить необходимость и желание в увеличении числа соединений к каждому хосту
4. Сохранить существующие интерфейсы, всё содержимое, формат URI и схемы
5. Сделать это внутри рабочей группы IETF HTTPbis

4.1. IETF и рабочая группа HTTPbis

Инженерный совет Интернета (IETF) – это организация, которая разрабатывает и продвигает интернет стандарты. Большею частью на протокольном уровне. Они хорошо известны по серии RFC-документов, документирующих всё: от TCP, DNS, FTP до лучших практик, HTTP и множества вариантов протокола, которые нигде не были применены.

Внутри IETF есть выделенные «рабочие группы», которые сформированы вокруг небольшого круга задач для достижения цели. Они составляют «устав» из набора принципов и ограничений для достижения поставленной цели. Любой и каждый может присоединиться к дискуссии и разработке. Все, кто участвует и что-либо высказывает, имеют равные возможности и шансы для влияния на результат и все учитываются как люди и личности, без оглядки на то, в какой компании работает человек.

Рабочая группа HTTPbis была сформирована в течении лета 2007 года и должна была обновить спецификацию HTTP 1.1 — отсюда и суффикс «bis». Обсуждение в группе новой версии HTTP протокола по-настоящему началось в конце 2012 года. Работа над обновлением HTTP 1.1 была завершена в начале 2014 года и привела к появлению серии документов RFC 7320.

Предположительно заключительное совещание для рабочей группы HTTPbis прошло в Нью-Йорке в начале июня 2014 года. Процедуры IETF до выхода официального RFC займут весь оставшийся год и возможно больше.

Некоторых больших игроков на поле HTTP не хватало в обсуждениях и встречах рабочей группы. Я не хочу называть какую-либо конкретную компанию или имя продукта здесь, но ясно, что на сегодняшний день некоторые действующие лица в Интернете, по всей видимости, уверены, что IETF сделает всё хорошо без привлечения этих компаний...

4.1.1. Суффикс «bis»

Группа названа HTTPbis, где суффикс «bis» происходит от латинского наречия, которое означает «два». Бис часто используют как суффикс или часть имени внутри IETF для обновления или второй попыткой работы над спецификацией. Также, как в случае HTTP 1.1.

4.2. http2 начался со SPDY

SPDY³ – это протокол, который был разработан и инициирован в Google. Они определённо разрабатывали его открыто и приглашали всех участвовать, но было очевидно, что они получают огромные преимущества имея контроль над двумя реализациями: популярный веб-браузер и значительная популяция серверов с активно используемыми сервисами.

Когда группа HTTPbis решила начать работать над http2, SPDY уже был проверен как рабочая концепция. Он показал, что его возможно развернуть в Интернете, и были опубликованные цифры, которые показывали насколько он справлялся. Работа над http2 впоследствии началась с черновика SPDY/3, который по большому счёту стал черновиком http2 draft-00 после пары операций поиска с заменой.

3 <http://en.wikipedia.org/wiki/SPDY>

5. Концепция http2

Так для чего создан http2? Где границы, которые ограничивают область работы группы HTTPbis?

Они, на самом деле, достаточно чёткие и накладывают заметные ограничения на способность команды к инновациям.

- ✓ Она должна поддерживать парадигмы HTTP. Это по-прежнему протокол, где клиенты отправляют запросы на сервер поверх TCP.
- ✓ Ссылки `http://` и `https://` не могут быть изменены. Нельзя добавить новую схему или сделать что-нибудь подобное. Количество контента, которое использует подобную адресацию слишком велико, чтобы когда-либо ожидать подобного изменения.
- ✓ HTTP1 серверы и клиенты будут существовать ещё десятилетия, мы должны иметь возможность проксировать их к http2-серверам.
- ✓ Следовательно, прокси должны быть способны конвертировать один в один возможности http2 в HTTP 1.1 для клиентов.
- ✓ Удалить или уменьшить число опциональных частей в протоколе. Это даже не столько требование, сколько мантра, пришедшая от SPDY и команды Google. Настаивая на том, что все требования обязательны, у вас не будет возможности не делать чего-то сейчас, а потом попасть в ловушку.
- ✓ Больше нет минорных версий. Было решено, что клиенты и серверы могут быть либо совместимы с http2, либо нет. Если окажется, что необходимо расширить протокол или изменить его, тогда появится http3. В http2 больше не будет минорных версий.



5.1. http2 для существующих URI схем

Как было отмечено ранее, уже существующие схемы URI не могут быть изменены, поэтому http2 должен использовать только их. Так как сегодня они используются для HTTP 1.x нам нужен явный способ для обновления протокола до http2 или как-то иначе попросить сервер использовать http2 вместо старых протоколов.

HTTP 1.1 уже имеет предопределённый способ для этого, так называемый Upgrade – заголовок, который позволяет серверу отправить ответ, используя новый протокол, при получении подобного запроса по старому протоколу. Ценой времени одной итерации запрос-ответ.

Расплата временем запроса-ответа не являлась тем, с чем команда SPDY могла согласиться, и, поскольку они также разрабатывали SPDY поверх TLS, они создали новое TLS-расширение, которое применялось для существенного сокращения согласования. Используя это расширение, названное NPN от Next Protocol Negotiation (согласование следующего протокола), сервер сообщает клиенту какие протоколы он знает, а клиент может выбрать и использовать наиболее предпочтительный.

5.2. http2 для https://

Большое внимание в http2 было уделено тому, чтобы он правильно работал поверх TLS. SPDY работал только поверх TLS и было сильное желание сделать TLS обязательным и для http2, но консенсус не был достигнут и http2 будет выпущен с опциональным TLS. Однако, два известных разработчика спецификации чётко заявили, что они будут реализовывать только http2 поверх TLS: руководитель Mozilla Firefox и руководитель Google Chrome. Это два лидирующих браузера на сегодня.

Причины выбора режима только с TLS заключаются в заботе о неприкосновенности частной жизни пользователя, а ранние исследования показали высокий уровень успеха у новых протоколов при использовании TLS. Это связано с широко-распространённым допущением, что всё, что приходит на 80 порт – это HTTP 1.1, и некоторые промежуточные сетевые устройства вмешиваются и уничтожают трафик других протоколов, которые работают на этом порту.

Тема обязательного TLS вызывает много размахов руками и агитационных призывов в списках рассылки и собраниях – добро это или зло? Это болезненная тема – помните об этом, если вы решите задать этот вопрос прямо в лицо участнику HTTPbis!



Также проходили длительные и ожесточённые дебаты о том, должен ли http2 определять список обязательных шифров при использовании TLS, или возможно он должен задавать чёрный список шифров, или вообще не должен требовать от TLS-слю, но в итоге оставили этот вопрос для рабочей группы TLS.

5.3. http2 согласование поверх TLS

Согласование следующего протокола (NPN) – это протокол, который использовался SPDY для согласования с TLS-серверами. Так как это не был настоящий стандарт, он был переработан в IETF и вместо него появился ALPN: *Application Level Protocol Negotiation* (согласование протокола на уровне приложения). ALPN продвигается для использования в http2, в то время как клиенты и серверы SPDY по-прежнему используют NPN.

То факт, что NPN появился первым, а для ALPN потребовалось время для прохождения стандартизации, привело к тому, что ранние реализации http2-клиентов и http2-серверов использовали оба этих расширения при согласовании http2. Также, поскольку NPN используется для SPDY и множество серверов поддерживают оба протокола SPDY и http2, то поддержка и NPN, и ALPN на таких серверах имеет смысл.

Основное отличие ALPN от NPN заключается в том, кто определяет какой использовать протокол. В ALPN клиент указывает серверу список протоколов в порядке предпочтения и сервер выбирает то, что ему удобнее, в то время как в NPN клиент делает финальный выбор.

5.4. http2 для http://

Как кратко отмечалось ранее, для текстового HTTP 1.1 для согласования http2 требуется отправить запрос серверу с заголовком Upgrade. Если сервер понимает http2, он ответит статусом «101 Switching» и затем начнёт использовать http2 в соединении. Вы конечно понимаете, что эта процедура обновления стоит времени одного полного сетевого запроса-ответа, но с другой стороны http2-соединение можно сохранять работоспособным и повторно использовать значительно дольше чем обычно используется HTTP1-соединение.

Несмотря на то, что некоторые представители браузеров настаивают, что они не будут реализовывать такой способ согласования http2, команда Internet Explorer выразила готовность его реализовать, а curl его уже поддерживает.

6. Протокол http2

Достаточно сказано о предпосылках, истории и политике, и теперь мы здесь. Давайте погрузимся в специфику протокола. Те части и концепции, которые слагают http2.

6.1. Бинарный протокол

http2 – это бинарный протокол.

Давайте попробуем осознать это на минутку. Если вы были знакомы с интернет-протоколами до этого, то велика вероятность, что инстинктивно вы сильно воспротивитесь этому факту и приготовите аргументы о том, как полезно иметь протоколы, использующие текст/ascii, и, что вы сотни раз писали HTTP 1.1 запросы руками просто подключившись телнетом к серверу.

http2 – бинарный для того, чтобы сделать формирование пакетов проще. Определение начала и конца пакета – одна из самых сложных задач в HTTP 1.1 и во всех текстовых протоколах в принципе. Уходя от опциональных пробелов и всевозможных способов записи одних и тех же вещей, мы делаем реализацию проще.

Кроме того, это позволяет гораздо проще разделять части связанные с самим протоколом и пакетом данных, что в HTTP1 беспорядочно перемешано.

Тот факт, что протокол позволяет использовать сжатие и часто работает поверх TLS также снижает ценность текста, так как вы в любом случае больше не увидите открытого текста в проводах. Мы просто должны прийти к пониманию, что надо использовать анализатор Wireshark или что-то похожее, чтобы выяснить что происходит на уровне протокола в http2.

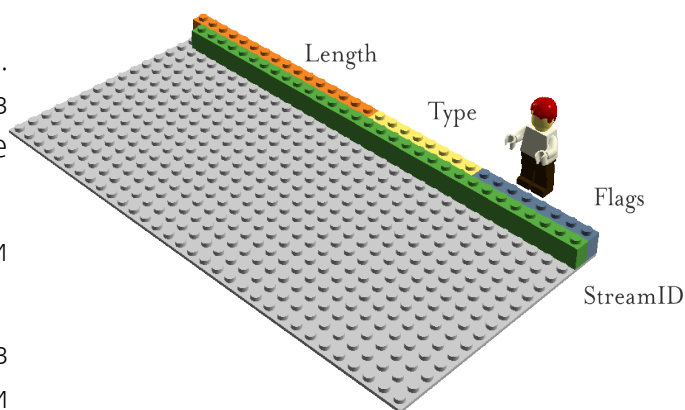
Отладка этого протокола, скорее будет выполняться такими утилитами, как curl, или путём анализа сетевого потока http2-диссектором Wireshark или чего-то подобного.

6.2. Бинарный формат

http2 отправляет бинарные фреймы. Существует несколько различных типов фреймов, но все они имеют одинаковое строение:

Тип, длина, флаги, идентификатор потока и полезная нагрузка фрейма.

Существует десять различных типов фреймов определённых в спецификации http2 и, в том числе, два, возможно наиболее важных, которые связывают с HTTP 1.1: DATA (данные) и HEADERS (заголовки). Я опишу некоторые из фреймов



более подробно дальше.

6.3. Мультиплексирование потоков

Идентификатор потока, упомянутый в предыдущей секции, описывающей формат фреймов, привязывает каждый фрейм, передаваемый поверх http2, к так называемому «потoku». Поток – это логическая ассоциация. Независимая двухсторонняя последовательность фреймов, которыми обмениваются клиент с сервером внутри http2-соединения.

Одно http2-соединение может содержать множество одновременных открытых потоков от любой из сторон, обменивающихся фреймами множества потоков. Потоки могут быть установлены и использованы в одностороннем порядке или совместно использованы как клиентом, так и сервером, и могут быть закрыты любой из сторон. Важен порядок, в котором отправляются фреймы. Получатель обрабатывает фреймы в порядке их получения.

Мультиплексирование потоков означает, что пакеты множества потоков смешаны в рамках одного соединения. Два (или больше) отдельных поезда данных собираются в один состав, а затем разделяются на другой стороне. Здесь два поезда:



Они собираются вместе по одному соединению в смешанном режиме:



В http2 мы увидим десятки и сотни одновременных потоков. Цена создания потока очень низка.

6.4. Приоритеты и зависимости

Каждый поток имеет приоритет, используемый для того, чтобы показать другому участнику обмена, какие потоки считать более важными.

Точные детали работы приоритетов в протоколе менялись несколько раз и до сих пор обсуждаются. Смысл в том, что клиент может указать какой из потоков наиболее важный и есть параметр зависимости, который позволяет сделать один поток зависимым от другого.

Приоритеты могут динамически меняться при обмене, что позволит браузеру быть уверенным, что когда пользователь проматывает страницу заполненную картинками, он сможет указать какие изображения являются наиболее важными, или когда вы переключаете вкладки, он может повысить приоритет потокам, которые неожиданно попали в фокус.

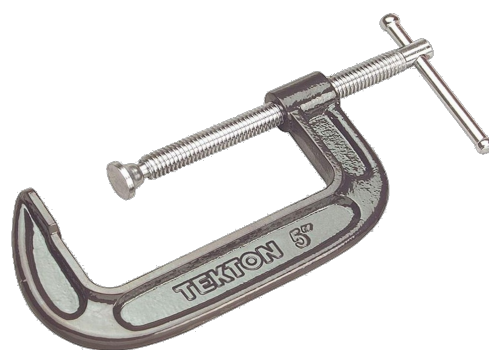
6.5. Сжатие заголовков

HTTP – это протокол без состояния. Вкратце, это означает, что каждый запрос должен содержать максимальное число деталей, которые требуются серверу, чтобы выполнить запрос без необходимости сохранять множества метаданных от предыдущего запроса. Так как http2 не меняет ни одну из подобных парадигм, ему приходится делать также.

Это делает HTTP повторяющимся. Когда клиент запрашивает множество ресурсов с одного сервера, например, изображения веб-страницы, это превращается в большую серию запросов, выглядящих почти одинаково. Для серии чего-то почти одинакового само собой напрашивается сжатие.

Как я уже упоминал, в то время как число объектов на странице увеличивается, использование cookie и размер запросов также продолжают расти. Cookie также должны быть включены во все запросы, практически всегда одинаковые на протяжении множества запросов.

Размер HTTP 1.1 запроса стал настолько велик со временем, что иногда он становился больше чем первоначальный размер TCP-окна, что делало его чрезвычайно медленным в отправке, требуя полного цикла отправки-приёма для получения подтверждения ACK от сервера перед тем, как полный запрос будет отправлен. Ещё один аргумент для сжатия.



6.5.1. Сжатие это непростая тема

Сжатие HTTPS и SPDY оказались уязвимыми к атакам BREACH⁴ и CRIME⁵. Путём вставки известного текста в поток и наблюдения за тем, как меняется зашифрованный вывод, атакующий мог выяснить, что было отправлено.

Выполнение сжатия для динамического контента в протоколе без риска быть подверженным одной из известных атак, требует серьёзного обдумывания и внимательности. То, что команда HTTPbis и пытается делать.

Так появился HPACK⁶, *Сжатие заголовков для HTTP/2*, который, как и подсказывает название, формат сжатия, предназначенный специально для http2-заголовков и, строго говоря, это отдельный интернет черновик спецификации. Новый формат совместно с другими контр-мерами, такими как специальные флаги, которые просят посредников не сжимать определённые заголовки и опционально добавлять в фреймы лишние пустые данные, чтобы усложнить атаку на сжатие.

Со слов Роберто Пиона (один из создателей HPACK) «HPACK был разработан так, чтобы для соответствующей реализации была затруднена утечка информации, чтобы сделать процесс кодирования и декодирования быстрым и дешёвым, предоставить получателю контроль над размером контекста сжатия, позволить прокси реиндексацию (т. е. Разделяемое состояние между фронтендом и бекендом внутри прокси) и для быстрого сравнения huffman-кодированных строк».

6.6. Reset (сброс) — передумал

Один из недостатков HTTP 1.1, когда HTTP-сообщение отправлено с заголовком Content-Length определённой длины, вы не можете так просто его остановить. Конечно, зачастую вы можете (но не всегда – я пропущу здесь длинное объяснение почему это так) разорвать TCP-соединение, но ценой повторного согласования нового TCP-соединения.

Теперь вы можете просто отменить отправку и начать новое сообщение. Это может быть достигнуто отправкой http2-фрейма RST_STREAM, который таким образом предотвратит растрату полосы пропускания и необходимости разрыва соединения.

6.7. Server push (посылка сервера)

Эта возможность также известна как «посылка в кэш». Идея в том, что если клиент запрашивает ресурс X, а сервер предполагает, что клиент наверняка затем попросит ресурс Z, отправляет этот ресурс клиенту без просьбы с его стороны. Это помогает клиенту поместить Z в свой кэш, и он будет на месте, когда потребуется.

Посылка сервера – это то, что клиент явно должен разрешить серверу, и даже если он разрешил, он может по своему выбору быстро отменить посланный поток с

4 http://en.wikipedia.org/wiki/BREACH_%28security_exploit%29

5 <http://en.wikipedia.org/wiki/CRIME>

6 <http://tools.ietf.org/html/draft-ietf-httpbis-header-compression-07>

помощью RST_STREAM, если он ему оказался не нужен.

6.8. Управление потоком

Каждый индивидуальный поток в http2 имеет своё объявленное окно потока, которое другая сторона разрешила для передачи данных. Если вы представляете как работает SSH, то это очень похоже и выполнено в том же духе и стиле.

Для каждого потока оба конца сообщают друг другу, что у них есть ещё место для принятия входных данных, и противоположному концу дозволено отправить только указанное количество данных до тех пор, пока окно не будет расширено.

7. Расширения

Протокол требует, что получатель должен считывать и игнорировать все неизвестные ему типы фреймов. Таким образом, две стороны могут согласовать использование нового типа фрейма на уровне от одного промежуточного хоста к другому, этим фреймам запрещено изменять состояние и их передача не управляется.

Дискуссия по поддержке расширений в http2 длилась на протяжении всего времени разработки протокола. После draft-12 маятник качнулся последний раз и расширения были разрешены.

Расширения не являются частью текущего протокола и будут задокументированы вне основной спецификации. На данный момент уже есть два типа фрейма, которые обсуждались для принятия в протокол и станут первыми фреймами отправляемыми как расширения. Я опишу их здесь, поскольку они популярны и изначально являлись «нативными» фреймами:

7.1. Альтернативные сервисы

После адаптации http2 есть причины ожидать, что TCP-соединения будут более продолжительными и дольше сохраняться в рабочем состоянии, чем это было с HTTP 1.x соединениями. Клиент сможет делать многое, что он захочет в рамках одного соединения к каждому хосту/сайту и это соединение вероятно будет открыто очень долго.

Это повлияет на работу HTTP-балансировщиков, и могут возникнуть ситуации, когда сайт захочет предложить клиенту подключиться к другому хосту. Это может быть как по причинам производительности, но также и необходимости отключения для обслуживания или подобных целей.

Сервер отправляет заголовок Alt-Svc⁷ (или ALTSVC-фрейм в http2), сообщая клиенту о наличии альтернативного сервиса. Дополнительный маршрут к такому же контенту, используя другой сервис, хост и номер порта.

7 <http://tools.ietf.org/html/draft-ietf-httpbis-alt-svc-01>

Ожидается, что клиент попытается асинхронно подключиться к сервису и начать использовать альтернативный сервис, если он нормально работает.

7.1.1. Опportunистический TLS

Заголовок Alt-Svc позволяет серверу, который предоставляет контент по http://, информировать клиента о наличии такого же контента доступного поверх TLS-соединения.

Это отчасти спорная возможность. Такое соединение выполняет неаутентифицированный TLS и не будет помечаться «безопасным» где бы то ни было: не будет показывать замочек в интерфейсе программы и никак не сообщать пользователю, что это не обычный старый открытый HTTP. Но это будет опportunистический TLS и некоторые люди очень уверенно выступают против этой концепции.

7.2. Блокировка

Фрейм данного типа может быть отправлен участником http2 только один раз, когда он имеет данные для отправки, но контроль потока запрещает ему отправку каких-либо данных. Смысл идеи в том, что если ваша реализация получает такой фрейм, вы должны понять, что ваша реализация что-то упустила и/или вы не можете добиться высоких скоростей передачи данных из-за этого.

Цитата из черновика draft-12, до того как фрейм стал расширением:

“Фрейм BLOCKED включён в эту версию черновика для удобства экспериментирования. Если результат эксперимента не даст положительных результатов он будет удалён.”

8. Мир http2

Как всё будет выглядеть, когда http2 будет принят? Будет ли он принят?

8.1. Как http2 повлияет на обычных людей?

http2 ещё широко не представлен и не используется. Мы не можем сказать точно как всё сложится. Мы видели как использовался SPDY и мы можем сделать некоторые предположения и вычисления, основанные на этом и на других прошлых и текущих экспериментах.

http2 уменьшает количество необходимых сетевых приёмов-передач, полностью избегает дилеммы блокировки начала очереди за счёт мультиплексирования и быстрого отклонения нежелательных потоков.

Он позволяет работать множеству параллельных потоков, число которых может превышать число соединений даже у наиболее активно использующих шардинг современных сайтов..



С приоритетами, корректно используемыми на потоках, шансы получить важные данные раньше менее важных значительно выше.

Собрав всё это вместе, я скажу, что очень высоки шансы, что это приведёт к ускорению загрузки страниц и повысит отзывчивость веб-сайтов. Коротко: лучше ощущения от веб-серфинга.

Насколько быстрее и насколько лучше мы увидим. Я не думаю, что мы пока готовы сказать. Во-первых технология по-прежнему ещё молода, а во-вторых, мы ещё не видели аккуратных реализаций клиентов и серверов, которые по-настоящему используют всю мощь, которую предоставляет новый протокол.

8.2. Как http2 повлияет на веб-разработку?

Годами веб-разработчики и среды веб-разработки собирали полный набор приёмов и утилит для обхода проблем HTTP 1.1, некоторые из которых отмечены в начале этого документа.

Большинство этих обходных путей, которые инструменты и разработчики теперь используют не задумываясь по умолчанию, вероятно ударят по производительности http2 или, по крайней мере, не воспользуются всеми преимуществами новой супер-силы http2. Спрайты и встраивание не должны

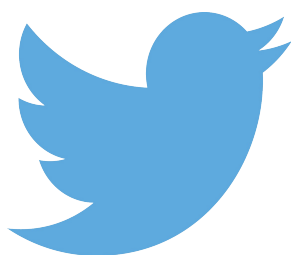
использоваться совместно с http2. Шардинг вероятно будет вреден для http2, так как http2 выигрывает от использования меньшего числа соединений.

Проблема здесь конечно в том, что веб-разработчики должны будут разрабатывать и внедрять веб-сайты в мире, где в лучшем случае на небольшой период времени, будут как HTTP 1.1, так и http2-клиенты, и, для получения максимальной производительности всеми пользователями, будет затратно предлагать два различных варианта сайта.

Только по этой причине, я подозреваю, пройдёт какое-то время, прежде чем мы увидим полное раскрытие потенциала http2.

8.3. http2 реализации

Конечно пытаться задокументировать специфические реализации в подобном этому документах – полностью бесполезный труд, который обречён на провал и устареет за весьма короткий период времени. Вместо этого я объясню ситуацию в широком смысле и направлю читателей к списку реализаций⁸ на веб-сайте http2.



Уже сейчас существует большое количество реализаций и их число растёт день ото дня по мере работы над http2. В то же время, на момент написания, существует 27 реализаций и многие из них реализуют самый последний черновик спецификации.



Firefox всегда был браузером в авангарде самых новейших версий черновика, Twitter продолжает предоставлять сервисы по http2. Google в апреле 2014 запустил поддержку http2 на нескольких тестовых серверах своих сервисов, а начиная с мая 2014 они поддерживают http2 в разрабатываемой версии браузера Chrome.



curl и libcurl поддерживают как простой http2, так и поверх TLS, используя одну из нескольких TLS-библиотек.

В то время как draft-16 является текущей версией, он бинарно совместимым с draft-14, и 14 версия является последней, помеченной как черновик для реализации и взаимодействия. Например, Firefox в ноябре 2014 стал объявлять поддержку обеих версий. Но хотя они бинарно совместимы в проводах, язык в спецификации изменился так, что их поведение, на самом деле, не идентично.

8.4. Типичная критика http2

В процессе разработки протокола дебаты возникали снова и снова, и, конечно, есть некоторое число людей, который верят, что протокол получился совершенно неправильным. Я хотел бы отметить некоторые из наиболее типичных жалоб и

8 <https://github.com/http2/http2-spec/wiki/Implementations>

аргументов против него:

8.4.1. “Протокол спроектирован и сделан в Google”

Есть также вариации подразумевающие, что мир ещё больше зависит и контролируется Google. Это не правда. Протокол разработан внутри IETF тем же самым способом, как протоколы разрабатывались за последние 30 лет. Однако, все мы признательны Google за бесподобную работу над SPDY, который не только доказал, что возможно внедрять новый протокол таким способом, но и помог получить оценки того, что мы можем получить.

8.4.2. “Протокол полезен только для браузеров и больших сервисов”

В какой-то мере это так. Одна из главных причин, кроющейся за разработкой http2 – это исправление конвейерной обработки HTTP. Если в вашем случае не требовалась конвейерная обработка, то http2 не будет вам особо полезен. Это конечно не единственное достижение протокола, но самое значительное.

Как только сервисы начнут понимать всю мощь и возможности мультимплексированных потоков в одном соединении, я ожидаю, что мы увидим увеличение числа приложений, использующих http2.

Небольшие REST API и простые программные применения HTTP 1.x не получат больших преимуществ от перехода на http2. Но, тем не менее, будет совсем немного минусов для большинства пользователей.

8.4.3. “Использование TLS делает его медленным”

В некоторой степени это верно. Согласование TLS даёт небольшие накладные расходы, но уже прилагаются усилия для ещё большего уменьшения числа запросов-ответов для TLS. Расходы на выполнение TLS- шифрования, по сравнению с передачей открытым текстом, не так незначительны и явно заметны, поэтому больше процессорного и времени и электричества будет потрачено на том же самом трафике, как в небезопасном протоколе. Сколько и какие последствия это будет иметь – тема для высказываний и измерений. Смотрите, например, istlsfastyet.com как один из источников по теме.

Телекомы и другие сетевые операторы, например *ATIS Open Web Alliance*, утверждают, что им нужен нешифрованный трафик⁹, чтобы осуществлять кэширование, сжатие и другие технологии, требуемые для быстрой работы через спутники, в самолётах и подобных системах.

http2 не обязывает использовать TLS, поэтому мы не должны смешивать термины.

Большинство пользователей в Интернете сегодня желают, чтобы TLS использовался более широко, и мы должны помогать защитить неприкосновенность частной жизни пользователей.

9 <http://www.atis.org/openweballiance/docs/OWAKickoffSlides051414.pdf> (страница 26)

Наконец, благодаря мультиплексированным потокам http2 по одному соединению, в привычной практике браузеров будет выполняться значительно меньше TLS-согласований и они будут работать быстрее, чем с HTTPS при использовании HTTP 1.1.

8.4.4. “Не ASCII-протокол всё портит”

Да, нам нравится идея возможности видеть протокол открыто, так как это упрощает отладку. Но текстовые протоколы гораздо более склонны к появлению ошибок и подвержены проблемам правильного синтаксического разбора.

Если вы действительно не можете принять бинарный протокол, тогда вы также не сможете принять и TLS, и сжатие в HTTP 1.x, которые существуют уже довольно длительное время.

8.4.5. “Он не быстрее, чем HTTP/1.1”

Это спорный вопрос для дебатов и дискуссий о том, что значит быстрее и как это измерить, но уже сейчас, во времена SPDY, множество проведённых тестов доказали увеличившуюся скорость загрузки страниц (например «Насколько быстр SPDY?»¹⁰ людьми из университета Вашингтона и «Оценка производительности веб-серверов с поддержкой SPDY»¹¹ Херва Серви) и подобные эксперименты были повторены также и для http2. Ждём дальнейших публикаций подобных тестов и экспериментов. Первый базовый тест, проведённый в httpwatch.com¹², обнадеживает, что HTTP/2 сдерживает свои обещания.

8.4.6. “Он содержит нарушения независимости слоёв”

Серьёзно, это является аргументом? Слои не являются неприкосновенными религиозными столпами и мы пересекли черты в нескольких серых областях при создании http2 в интересах создания хорошего и эффективного протокола в заданных условиях.

8.4.7. “Не исправлены некоторые дефекты HTTP/1.1”

Это правда. Из-за специфической цели поддержки HTTP/1.1 есть несколько старых HTTP особенностей, которые остались на месте. Такие как заголовки, включая ужасные Cookie, заголовки авторизации и другие. Но ценой поддержки такой парадигмы мы получаем протокол, который возможно внедрять без невероятного объёма работ, который бы потребовал переписывания фундаментальных частей. Http2, на самом деле, просто новый слой фреймов.

10 https://www.usenix.org/system/files/conference/nsdi14/nsdi14-paper-wang_xiao_sophia.pdf and https://www.usenix.org/sites/default/files/conference/protected-files/nsdi14_slides_wang.pdf

11 <http://www.neotys.com/blog/performance-of-spdy-enabled-web-servers/>

12 <http://blog.httpwatch.com/2015/01/16/a-simple-performance-comparison-of-https-spdy-and-http2/>

8.5. Станет ли http2 широко распространён?

Ещё довольно рано говорить наверняка, но я могу предположить и оценить, и это я и собираюсь сделать здесь.

Скептики скажут «смотрите как хорошо был сделан IPv6», как пример нового протокола, который потребовал десятки лет просто, чтобы хотя бы начать широко применяться. http2 – это вовсе не IPv6. Это протокол, работающий поверх TCP, использующий обычный HTTP механизм обновления, номер порта, TLS и т. д. Он вообще не потребует замены большинства маршрутизаторов и брендмауеров.

Google доказал миру с помощью своей работы над SPDY, что такой новый протокол может быть внедрён и использован браузерами и сервисами с несколькими реализациями за довольно небольшой период времени. Несмотря на то, что количество серверов в Интернете, которые сегодня предлагают SPDY в районе 1%, но количество данных, с которыми они работают значительно больше. Некоторые из наиболее популярных веб-сайтов сегодня предлагают SPDY.

http2, основанный на тех же базовых парадигмах, что и SPDY, я уверен, вероятно будет внедрён ещё активнее, так как это официальный протокол IETF. Внедрение SPDY всегда сдерживалось клеймом «это протокол Google».

За выпуском стоят несколько известных браузеров. Представители Firefox, Chrome и Internet Explorer выразили готовность выпускать браузер с поддержкой http2 и показали рабочие реализации.

Существуют и несколько серверных провайдеров, которые вероятно вскоре предложат http2, включая Google, Twitter и Facebook, и мы надеемся вскоре увидеть поддержку http2 добавленную в популярные реализации веб-серверов, таких как Apache HTTP Server и nginx.

H2o¹³ – это новый невероятно быстрый HTTP-сервер с поддержкой http2 демонстрирует явный потенциал.

Некоторые крупные вендоры прокси-серверов, включая HAProxy, Squid и Varnish выразили намерение поддерживать http2.

Думаю есть гораздо больше реализаций, которые появятся после ратификации спецификации как RFC.

13 <https://github.com/h2o/h2o>

9. http2 в Firefox

Firefox очень плотно отслеживает черновик спецификации и предоставляет поддержку тестовой http2 реализации уже многие месяцы. За время разработки протокола http2 клиенты и серверы должны договориться о том, какую версию черновика протокола они реализовали, что делает несколько раздражительным запуск тестов. Просто проверьте, что клиент и сервер согласовали какую версию черновика они реализовали.

9.1. Сначала включите это

Введите «about:config» в адресной строке и ищите опцию, которая называется «network.http.spdy.enabled.http2draft». Убедитесь, что она установлена в **true**.

В версиях Firefox Nightly, начиная с 8 августа 2014, http2 включён по умолчанию.

9.2. Только TLS

Помните, что Firefox реализовывает только http2 поверх TLS. Вы увидите работу http2 в Firefox, только когда перейдёте на https:// сайты, которые поддерживают http2.

9.3. Прозрачно!

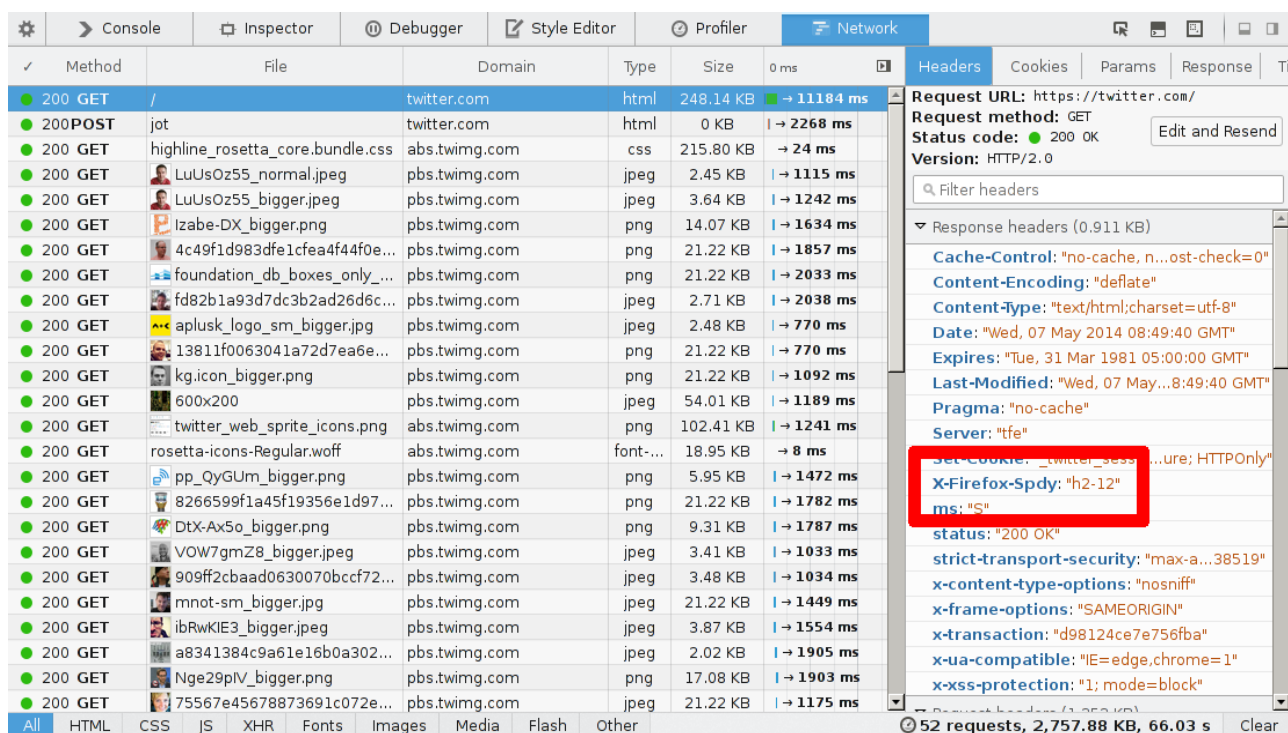


Рисунок 2: скриншот Firefox в дни, когда draft-09 был текущим

Ни один элемент нигде в интерфейсе не скажет, что вы работаете по http2. Вы не сможете это так просто понять. Есть лишь один способ узнать это, включив «Веб-разработка->Сеть», проверить заголовки ответа и увидеть, что вы получили от сервера.. Отклик содержит что-то о «HTTP/2.0» и Firefox вставляет свой заголовок с

названием «X-Firefox-Spdy», как показано на этом, уже устаревшем скриншоте.

Заголовки, которые вы увидите в сетевых инструментах, когда общаетесь по http2, конвертируются из бинарного формата http2 в похожие на старые HTTP1.x заголовки.

10. http2 в curl

Проект curl начал эксперименты с поддержкой http2, начиная с сентября 2013 года.

В духе curl, мы намереваемся поддерживать все аспекты http2, которые сможем. curl часто используется как тестовая утилита и простой способ проверить веб-сайт, и мы намерены сохранить эту возможность также и для http2.

Curl использует отдельную библиотеку `nghttp2`¹⁴ для работы со слоем http2 фреймов.

10.1. Выглядит как HTTP 1.x

Внутренне curl будет конвертировать входящие http2-заголовки в заголовки в стиле HTTP 1.x и передавать их пользователю так, что они будут казаться очень похожими на существующий HTTP. Это упростит переход для чего бы не использовался curl и HTTP сегодня. Точно также curl конвертирует выходные заголовки. Передайте curl заголовки в стиле HTTP 1.x и он сконвертирует их на лету, когда будет передавать для http2-сервера. Это также позволяет пользователям особо не волноваться или заботиться о том, какая конкретно версия HTTP используется в сетевом обмене.

10.2. Простой текст

curl поддерживает нешифрованный http2 с помощью заголовка Upgrade. Если вы выполняете HTTP-запрос и запрашиваете HTTP 2, curl попросит сервер обновить соединение до http2, если это возможно.

10.3. TLS и выбор библиотеки

curl поддерживает широкий спектр различных TLS-библиотек в качестве своего TLS-бэкенда, и это по-прежнему верно и для поддержки http2. Проблема с TLS для http2 – это наличие поддержки ALPN и, в некоторой мере, поддержки NPN.

Собирайте curl с современными версиями OpenSSL или NSS чтобы получить поддержку ALPN и NPN. При использовании GnuTLS или PolarSSL вы получите поддержку ALPN, но не NPN.

10.4. Использование в командной строке

Чтобы объяснить curl, что надо начать использовать http2, как открытым текстом, так и через TLS, вы можете использовать опцию `--http2` («дефис дефис http2»). Curl по прежнему по умолчанию использует HTTP/1.1, поэтому эта опция необходима, если вы хотите http2.

10.5. Опции libcurl

Ваше приложение как обычно использует `https://` или `http://` URL'ы, но вы можете

14 <https://nghttp2.org/>

задать опцию из `curl_easy_setopt` `CURLOPT_HTTP_VERSION` в значение `CURL_HTTP_VERSION_2`, чтобы `libcurl` попытался использовать `http2`. Он попытается сделать всё возможное для работы с `http2`, если сможет, но по-прежнему будет оперировать с `HTTP 1.1`.

11. После http2

Большое число трудных решений и компромиссов было сделано в http2. После развёртывания http2 существует путь для обновления до других рабочих версий протокола, что открывает возможность для создания большего числа ревизий протокола впоследствии. Это также приносит представление и инфраструктуру, которая сможет поддерживать множество различных версий протокола одновременно. Возможно нам не нужно полностью удалять старое, когда создаём новое?

http2 по-прежнему имеет многое из устаревшего HTTP 1, перенесённого в него из-за желания сохранить возможность проксирования трафика во все направления между HTTP 1 и http2. Кое что из этого наследия затрудняет дальнейшую разработку и нововведения. Возможно http3 сможет отбросить часть из них?

Как вы думаете, что по-прежнему не хватает в http?

12. Дальнейшее чтение

Если вы считаете, что этот документ немного простоват по содержанию или техническим деталям, далее даны дополнительные ресурсы, которые удовлетворят ваше любопытство:

- Список рассылки HTTPbis и его архивы: <http://lists.w3.org/Archives/Public/ietf-http-wg/>
- Актуальная спецификация черновика http2 и связанных документов группы HTTPbis: <http://datatracker.ietf.org/wg/httpbis/>
- Детали сетевых решений в http2 Firefox: <https://wiki.mozilla.org/Networking/http2>
- Детали реализации http2 в curl: <http://curl.haxx.se/dev/readme-http2.html>
- Веб-сайт http2: <http://http2.github.io/> и возможно особенно FAQ: <http://http2.github.io/faq/>

13. Благодарности

Вдохновение в работе, а также лего-изображение получено от Марка Ноттингема (Mark Nottingham) <http://www.mnot.net/talks/http2-n-you/>

Данные HTTP-тренда взяты с <http://httparchive.org>.

График RTT взят из презентации, выполненной Майком Белше (Mike Belshe).

Моим детям Агнес и Рекс, позволившим взять из Лего фигурки для создания изображения блокировки начала очереди.

Спасибо моим друзьям за рецензирование и отзывы: Kjell Ericson, Bjorn Reese, Linus Swälas и Anthony Bryan. Я высоко ценю Вашу помощь, которая действительно улучшила документ!

Перевод на русский язык выполнил Владимир Леттиев ([@truecrux](#))