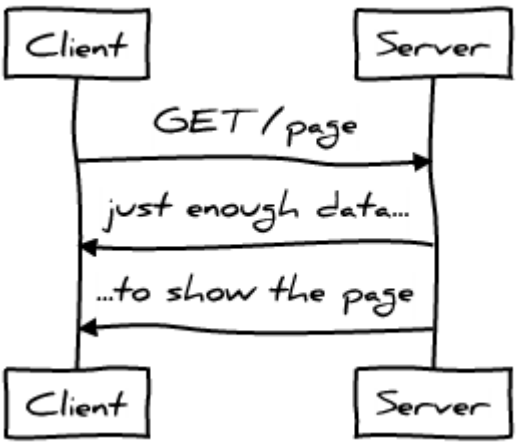
Увеличение производительности HTTP

Егоров Андрей Алексеевич
Бакалавр НИУ МИЭТ, Россия, г. Москва
E-mail: Egorov_andrey_1994@mail.ru
Россия, г. Москва

Один из аспектов понятия «производительность Web» заключается в том, чтобы уменьшить наблюдаемые пользователем задержки; получить готовую к работе страницу как можно быстрее. В отношении протокола HTTP это подразумевает, что идеальный протокол связи выглядит примерно так:

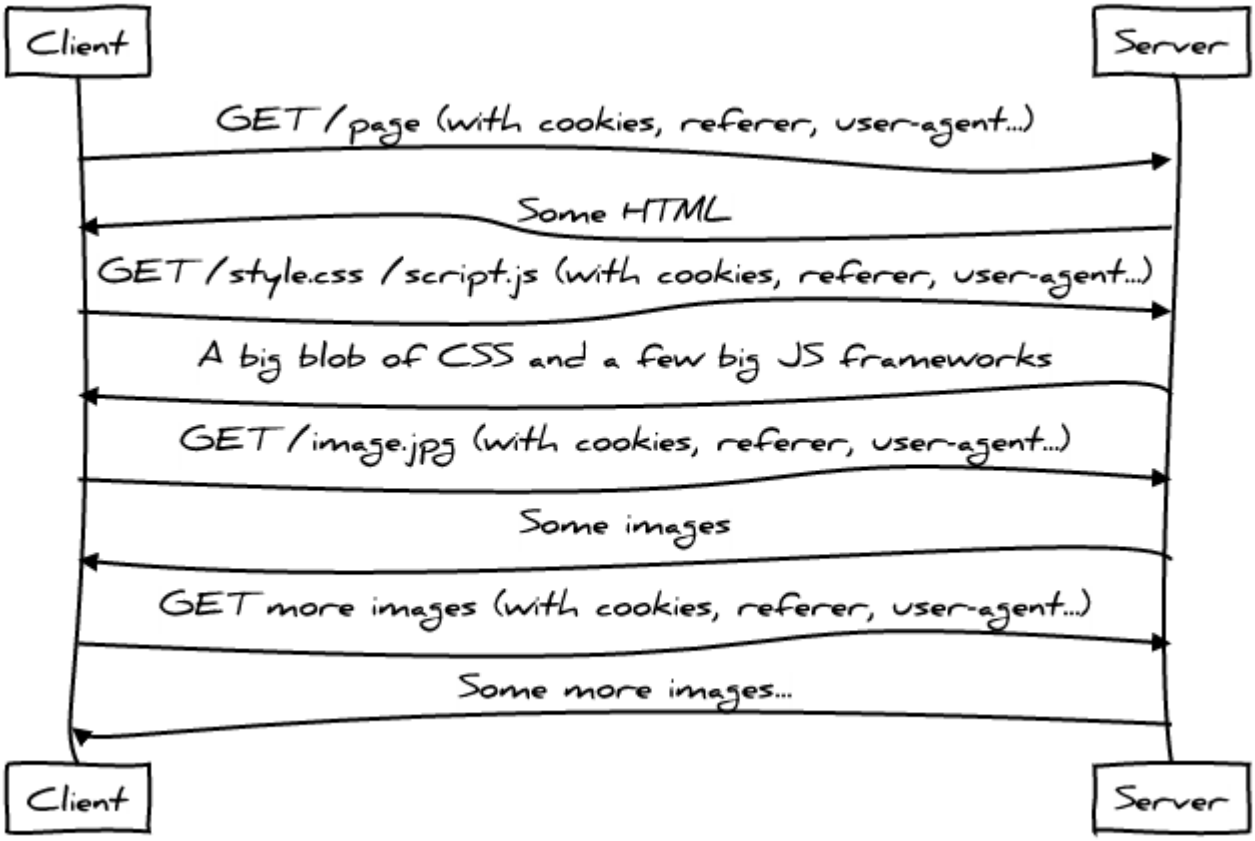


Клиент шлёт минимально необходимое количество данных, чтобы описать свой запрос, а сервер отдаёт ему минимально необходимое количество данных для отображения страницы и всё это происходит за минимально возможное количество раундов связи. Лишние данные, пересылаемые на сервер или получаемые с сервера, означают увеличение времени загрузки и повышение шансов потери пакетов, перегруженность канала связи. Лишние циклы отправки\приёма данных из-за «болтливости» протокола и задержки (особенно в мобильных сетях, где 100ms — лучшее возможное время отклика) тоже ухудшают ситуацию.

Итак, если мы описали идеальный случай — соответствует ли ему протокол HTTP? И можем ли мы ещё как-нибудь улучшить его?

HTTP/1.1

HTTP/1.1 это хороший протокол по целому ряду причин, но, к сожалению, производительность не одна из них по причине того, как современные веб-приложения его используют. А используют они его сегодня вот так:



Не очень-то хорошо.

Использование веб-приложениями протокола HTTP/1 достаточно «болтливо», поскольку клиент обращается к серверу снова и снова для загрузки необходимых ему файлов; сначала загружается HTML, затем CSS и Javascript. Загрузка каждого следующего файла добавляем в наш «разговор» с сервером новую главу, увеличивает общую задержку загрузки страницы, нарушая наше правило «минимальности необходимых раундов связи».

Более того, даже сами запросы к ресурсам уже добавляют много лишних данных, нарушая правило «минимальности необходимых данных». Это происходит из-за наличия заголовков вроде Referer, User-Agent и, конечно же, Cookie, которые повторяются в каждом запросе, умножаясь иногда в сотню раз от минимально необходимого их количества (по количеству ресурсов, необходимых средней страницей современного Веба).

Ну и наконец, из-за присущего протоколу HTTP/1 явлению HOL-блокировки, стало общей практикой помещать несколько отдельных ресурсов в один (например,CSS-spriting). Все эти изящные хаки протокола HTTP/1, тем не менее, имеют свою цену; они вынуждают клиента загружать больше данных, чем ему необходимо в данный момент для показа конкретной страницы, что нарушает описанный нами идеальный случай, а значит мы не покажем страницу так быстро, как это только возможно.

Несмотря на всё сказанное, HTTP/1.1 всё же не так плох, даже в плане производительности. Например, в нём есть кеширование, что избавляет вас от повторной загрузки неизменённых ресурсов, а также условные запросы, что позволяет вам понять, нужно ли загружать новую версию файла.

HTTP/2

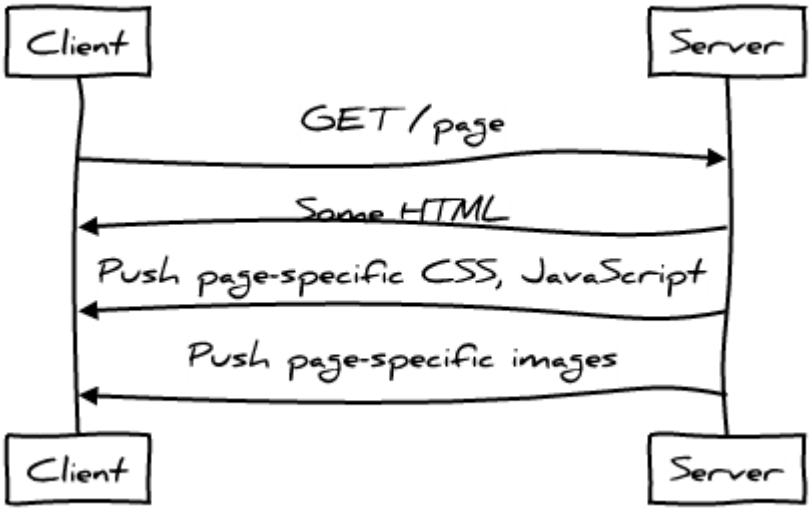
Протокол HTTP/2 пытается решать проблемы 1.1 несколькими путями:

1. Полное мультиплексирование означает решение проблемы HOL-блокировки. Вы можете загружать все ресурсы своей страницы по одному HTTP-соединению и не волноваться о том, сколько для этого понадобится сделать запросов. «Оптимизации» со склеиванием файлов в один можно оставить в прошлом.

2. Сжатие заголовков решает проблему их избыточности. Теперь вы можете вместить десятки (или даже сотни) запросов в буквально несколько IP-пакетов. Это серьёзно приближает нас к «минимально необходимому набору данных» нашего идеального протокола.

3. HTTP/2 позволяет серверу отправлять данные клиенту ещё до их запроса клиентом, исходя из предположения, что они ему скоро понадобятся. Это уменьшает количество раундов связи клиента и сервера.

Таким образом, сеанс связи с использованием протокола HTTP/2 выглядит так:

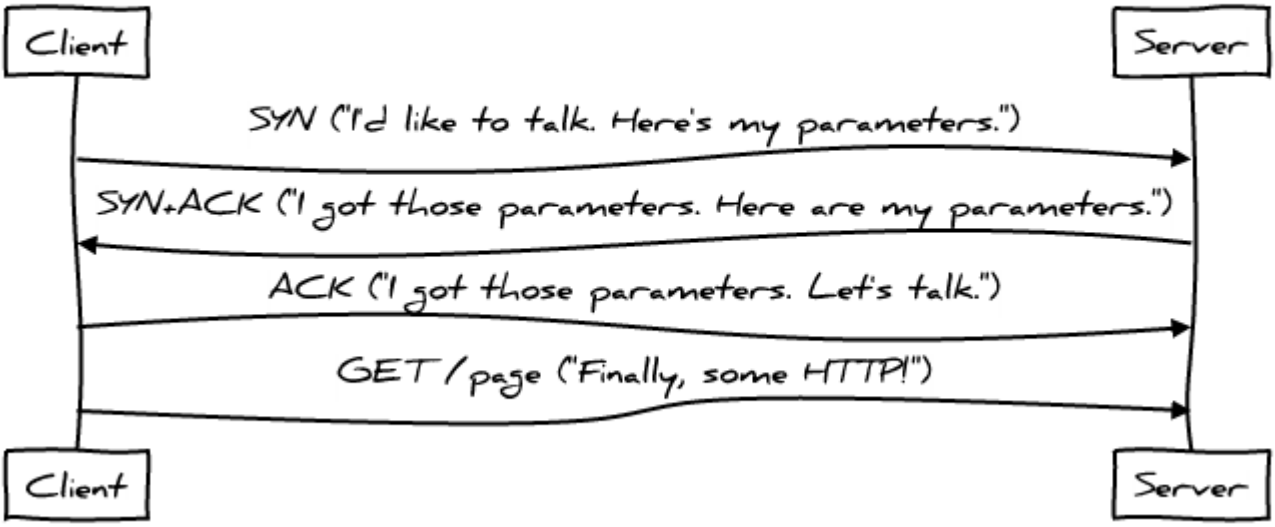


Здесь вы можете увидеть, как сервер начинает пересылку CSS и Javascript клиенту ещё до того, как клиент об этом попросил. Сервер знает, что клиент, запросивший HTML, с большой вероятностью запросит и относящиеся к нему CSS и Javascript файлы, а значит можно отправить их, не дожидаясь самого запроса. Таким образом, мы теряем меньше времени на ожидание, используем сеть более рационально.

Следует отметить, всё это работает не так уж просто. До сих пор в HTTP/2 есть открытые вопросы, касающиеся того, что и когда сервер должен считать необходимым к отправке без запроса клиента.

TCP

Давайте поговорим о влиянии на общую производительность других протоколов, используемых браузером для загрузки страниц. А оно тоже может быть существенным: TCP использует троекратное рукопожатие ещё до того, как первый байт стоящего выше HTTP будет переслан:



Это добавляет «болтливости» каждому сеансу связи.TCP Fast Open позволяет приложениям отправлять данные прямо в SYN и SYN+ACK пакетах. К сожалению, это в данный момент поддерживается только в Linux и OSX, и более того, есть некоторые особенности применения TCP Fast Open именно с протоколом HTTP, над которыми сейчас работает сообщество. Например, не гарантируется, что данные, прикреплённые к SYN-пакету, будут пересланы лишь один раз. Это открывает уязвимость с потенциальными повторными запросами, которая может быть использована для атак. Таким образом, запрос POST — не лучший кандидат для применения TCP Fast Open. Более того, некоторые GET-запросы тоже имеют заметные побочные эффекты, а браузеры не имеют никаких средств, чтобы отличить такие запросы от тех, которые таких эффектов не имеют.

HTTP/next

TCP Fast Open и TLS 1.3 уменьшают количество циклов связи клиента и сервера при открытии соединения. Другой способ достичь того же — переиспользовать уже ранее открытое соединение. Сейчас идёт дискуссия о том, как объединять соединения HTTP/2 более агрессивно; это позволит не только избежать затрат на открытие новых соединений, но и более эффективно использовать уже имеющиеся — протокол TCP наиболее хорош именно в долгоживущих, плотно заполненных данными соединениях. Это включает в себя отправку клиентам сертификатов, доказывающих, что соединение может быть безопасно переиспользовано для работы с другими источниками.

Сейчас обсуждаются даже более кардинальные эксперименты: замена TCP на UDP, навроде QUIC. Есть много спорных моментов, но сама перспектива свести начальный обмен данным фактически до нуля — очень привлекательна. Более того, возможность получить доступ к данным не в том порядке, как они были отправлены, тоже может быть очень полезна. Это ещё один способ избежать HOL-блокировок в TCP (протоколе с упорядоченной доставкой пакетов). Мы можем выбрать из потока пакетов нужные нам, понять, что какие-то были потеряны, запросить их повторно — и продолжить обработку следующих, не дожидаясь результатов повторного запроса.

QUIC только начинает свой путь, так что мы ещё не увидим хорошей его реализации какое-то время (а может быть и никогда вообще). Один из возможных вариантов — изучить на примере QUIC все плюсы и минусы подхода, чтобы понять, как мы можем улучшить производительность TCP, не ударяясь в столь кардинальные изменения архитектуры Web.