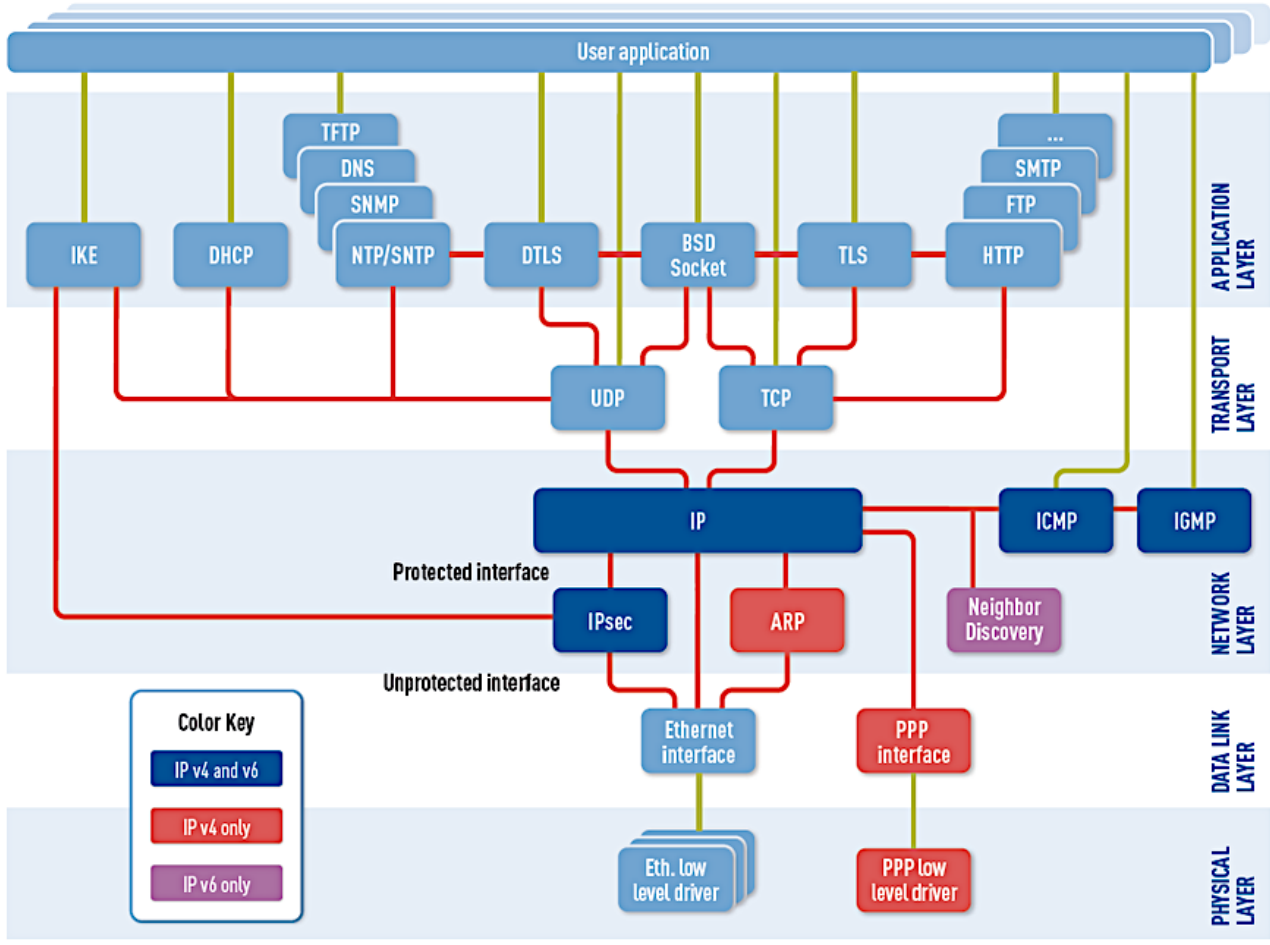
План

- Эволюция протокола HTTP
- Веб-сервисы
- Сравнение с RPC и альтернативы

Протокол

- Описание правил взаимодействия компонентов системы
- Типы, семантика и структура сообщений
- Форматы передачи данных
- Правила обработки сообщений
- Адресация компонентов
- Управление соединением
- Обнаружение и обработка ошибок
- ...

Стек протоколов



Hypertext Transfer Protocol (HTTP)

HTTP 0.9

- The Original HTTP as defined in 1991
- Клиент-сервер, запрос-ответ
- Представление данных - ASCII
- Запрос - одна строка (GET ...)
- Ответ - гипертекстовый документ (HTML)
- Транспорт - TCP, соединение закрывается после каждого запроса

HTTP 0.9: пример

```
$> telnet google.com 80

Connected to 74.125.xxx.xxx

GET /about/

(hypertext response)
(connection closed)
```

HTTP/1.0

- RFC 1945 (1996)
  - Документирует best practices, не является формальной спецификацией
  - Фокус на простоте реализации
- Методы GET, HEAD, POST
- Запрос и ответ содержат версию протокола
- Запрос и ответ могут содержать заголовки (дополнительные метаданные)
- Ответ включает статус обработки запроса
- Тело ответа может содержать не только гипертекст
- Content encoding, character set support, multi-part types, authorization, caching...
- Соединение по-прежнему закрывается после каждого запроса

HTTP/1.0: пример

```
$> telnet website.org 80

Connected to xxx.xxx.xxx.xxx

GET /rfc/rfc1945.txt HTTP/1.0
User-Agent: CERN-LineMode/2.15 libwww/2.17b3
Accept: */*

HTTP/1.0 200 OK
Content-Type: text/plain
Content-Length: 137582
Expires: Thu, 01 Dec 1997 16:00:00 GMT
Last-Modified: Wed, 1 May 1996 12:45:26 GMT
Server: Apache 0.84

(plain-text response)
(connection closed)
```

HTTP/1.1

- RFC 2068 (1997), RFC 2616 (1999), RFC 7230 (2014)
- Официальный Internet Standard

> In general, an Internet Standard is a specification that is stable and well-understood, is technically competent, has multiple, independent, and interoperable implementations with substantial operational experience, enjoys significant public support, and is recognizably useful in some or all parts of the Internet.

HTTP/1.1

> **The Hypertext Transfer Protocol (HTTP)** is an **application-level protocol** for distributed, collaborative, **hypermedia** information systems. It is a **generic**, **stateless**, protocol that can be used for **many tasks** beyond its use for hypertext, such as name servers and distributed object management systems, through **extension** of its request methods, error codes and headers. A feature of HTTP is the typing and **negotiation of data representation**, allowing systems to be built independently of the data being transferred.
>
> RFC 2068 (1997)

HTTP/1.1

- Методы OPTIONS, PUT, DELETE, TRACE, CONNECT
- Persistent (keepalive) connections
- Chunked encoding transfers
- Byte-range requests
- Additional caching mechanisms
- Transfer encodings
- Request pipelining

HTTP/1.1: пример

```
$> telnet website.org 80
Connected to xxx.xxx.xxx.xxx

GET /index.html HTTP/1.1
Host: website.org
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_7_4)... (snip)
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Encoding: gzip,deflate,sdch
Accept-Language: en-US,en;q=0.8
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.3
Cookie: __qca=P0-800083390... (snip)

HTTP/1.1 200 OK
Server: nginx/1.0.11
Connection: keep-alive
Content-Type: text/html; charset=utf-8
Via: HTTP/1.1 GWA
Date: Wed, 25 Jul 2012 20:23:35 GMT
Expires: Wed, 25 Jul 2012 20:23:35 GMT
Cache-Control: max-age=0, no-cache
Transfer-Encoding: chunked
```

```
100
<!doctype html>
(snip)

100
(snip)

0

GET /favicon.ico HTTP/1.1
Host: www.website.org
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_7_4)... (snip)
Accept: */*
Referer: http://website.org/
Connection: close
Accept-Encoding: gzip,deflate,sdch
Accept-Language: en-US,en;q=0.8
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.3
Cookie: __qca=P0-800083390... (snip)
```

```
HTTP/1.1 200 OK
Server: nginx/1.0.11
Content-Type: image/x-icon
Content-Length: 3638
Connection: close
Last-Modified: Thu, 19 Jul 2012 17:51:44 GMT
Cache-Control: max-age=315360000
Accept-Ranges: bytes
Via: HTTP/1.1 GWA
Date: Sat, 21 Jul 2012 21:35:22 GMT
Expires: Thu, 31 Dec 2037 23:55:55 GMT
Etag: W/PSA-GAu26oXbDi

(icon data)
(connection closed)
```

Методы HTTP

- GET
  - запрос представления ресурса с данным URI
  - только чтение, не меняет состояние сервера
- POST
  - создание нового ресурса (с новым URI!), отправка формы, запуск операции
  - необходимые данные передаются в теле запроса
- PUT
  - запись представления ресурса с данным URI
  - в теле запроса передается представление ресурса
- DELETE
  - удаление ресурса с данным URI

Особенности методов

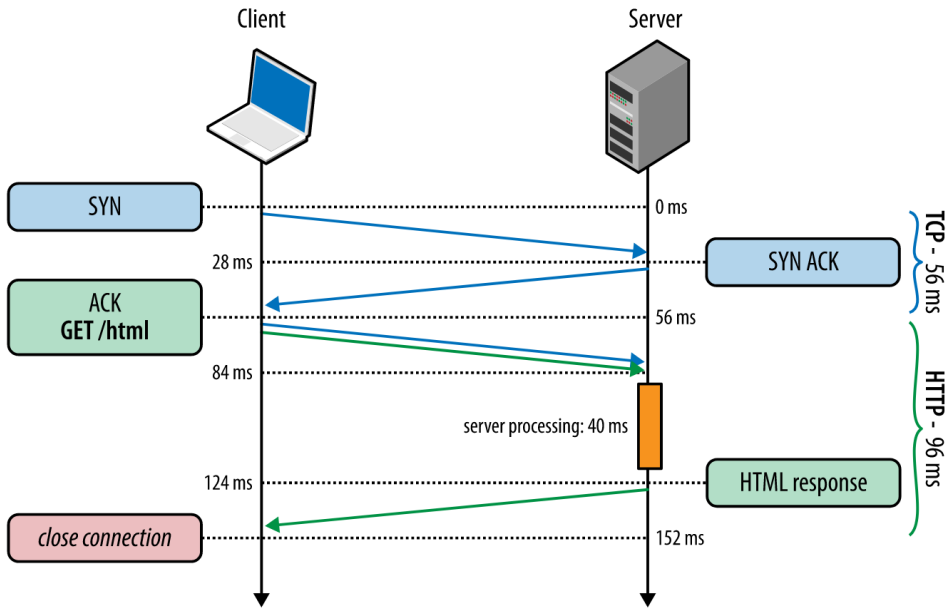
HTTP method	RFC	Request has Body	Response has Body	Safe	Idempotent	Cacheable
GET	RFC 7231	Optional	Yes	Yes	Yes	Yes
HEAD	RFC 7231	Optional	No	Yes	Yes	Yes
POST	RFC 7231	Yes	Yes	No	No	Yes
PUT	RFC 7231	Yes	Yes	No	Yes	No
DELETE	RFC 7231	Optional	Yes	No	Yes	No
CONNECT	RFC 7231	Optional	Yes	No	No	No
OPTIONS	RFC 7231	Optional	Yes	Yes	Yes	No
TRACE	RFC 7231	No	Yes	Yes	Yes	No
PATCH	RFC 5789	Yes	Yes	No	No	No

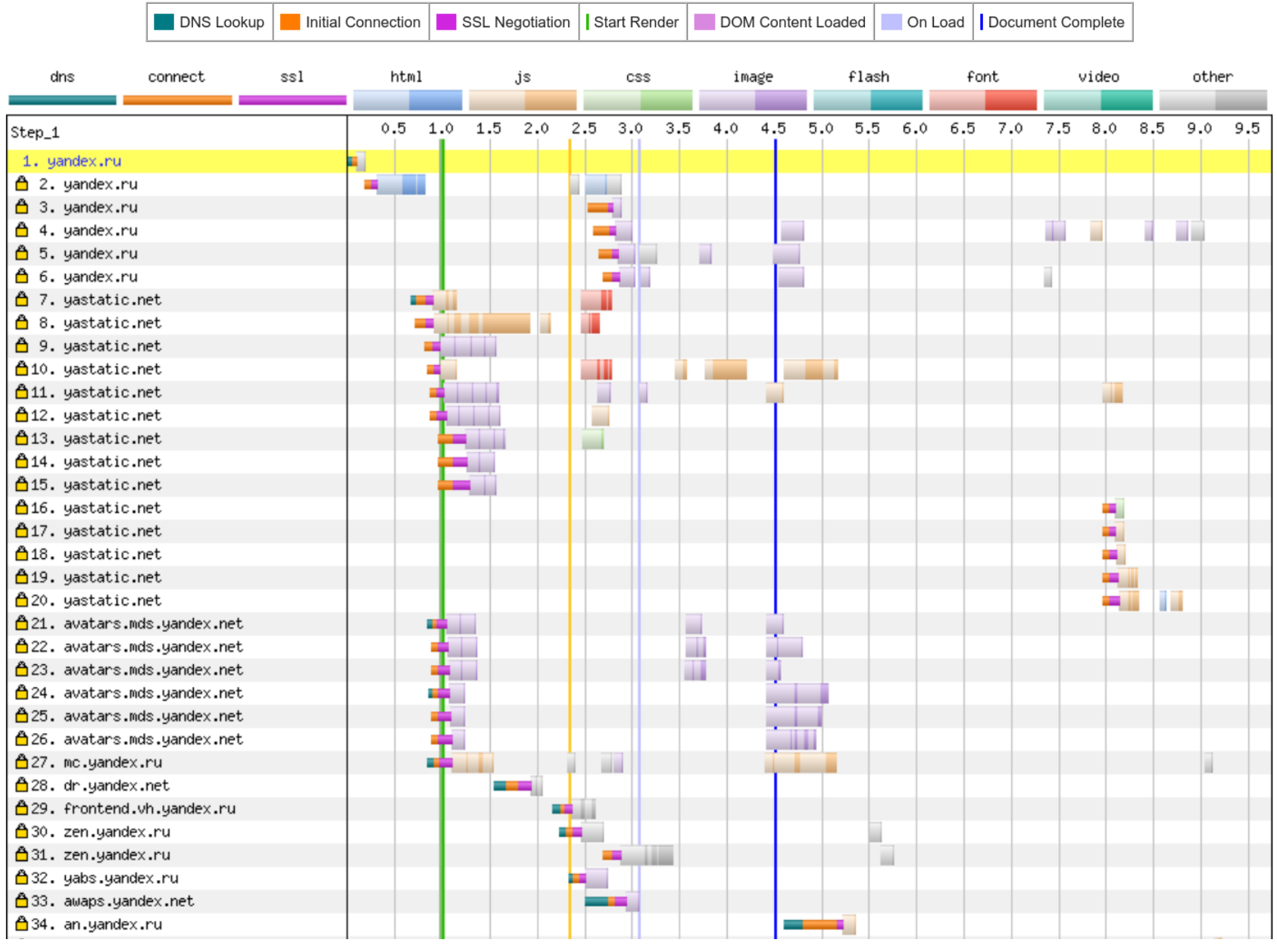
Эволюция веб-приложений

- Одиночный гипертекстовый документ
- Веб-страниц
  - стили, изображения, нет интерактива
- Веб-приложение
  - интерактив, JavaScript, динамический HTML, AJAX, SPA
- Растут объемы и количество запросов
  - высокие требования к производительности приложений
  - задержка более 300 мс неприемлема

Задержка на стороне клиента

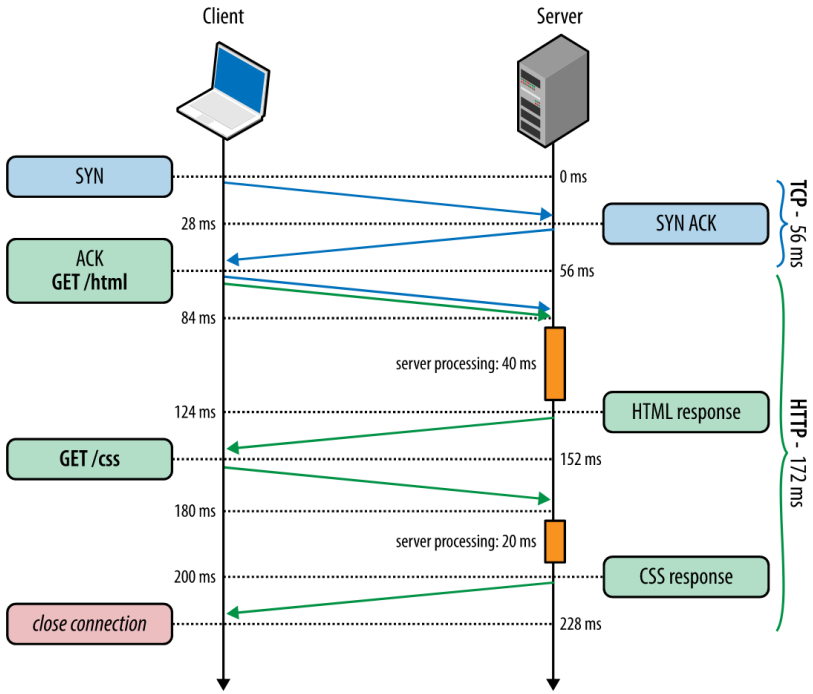
- Разрешение доменного имени (DNS lookup)
- Установление TCP-соединения: RTT
  - HTTPS требует дополнительной процедуры TLS handshake: 1-2 RTT
- Отправка запроса и получение ответа: RTT + время обработки запроса



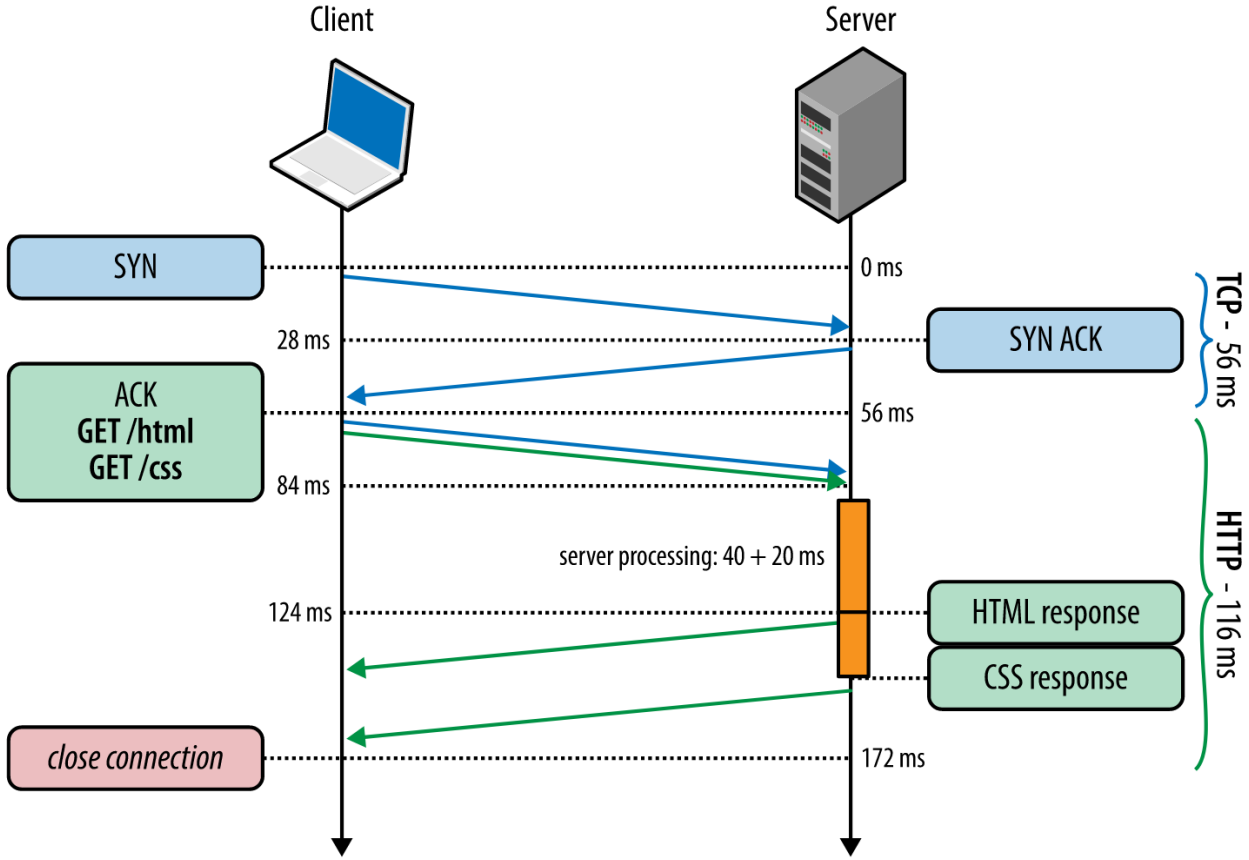

DNS Lookup
Initial Connection
SSL Negotiation
Start Render
DOM Content Loaded
On Load
Document Complete
dns
connect
ssl
html
js
css
image
flash
font
video
other
Step_1
0.5 1.0 1.5 2.0 2.5 3.0 3.5 4.0 4.5 5.0 5.5 6.0 6.5 7.0 7.5 8.0 8.5 9.0 9.5
1. yandex.ru
2. yandex.ru
3. yandex.ru
4. yandex.ru
5. yandex.ru
6. yandex.ru
7. yastatic.net
8. yastatic.net
9. yastatic.net
10. yastatic.net
11. yastatic.net
12. yastatic.net
13. yastatic.net
14. yastatic.net
15. yastatic.net
16. yastatic.net
17. yastatic.net
18. yastatic.net
19. yastatic.net
20. yastatic.net
21. avatars.mds.yandex.net
22. avatars.mds.yandex.net
23. avatars.mds.yandex.net
24. avatars.mds.yandex.net
25. avatars.mds.yandex.net
26. avatars.mds.yandex.net
27. mc.yandex.ru
28. dr.yandex.net
29. frontend.vh.yandex.ru
30. zen.yandex.ru
31. zen.yandex.ru
32. yabs.yandex.ru
33. awaps.yandex.net
34. an.yandex.ru

Keepalive Connections

- Использование установленного TCP-соединения для отправки последующих запросов
  - Уменьшает задержку на RTT
- По умолчанию соединение не закрывается после обработки запроса (HTTP/1.1)
- Клиент может запросить закрытие соединения с помощью заголовка `Connection: close`

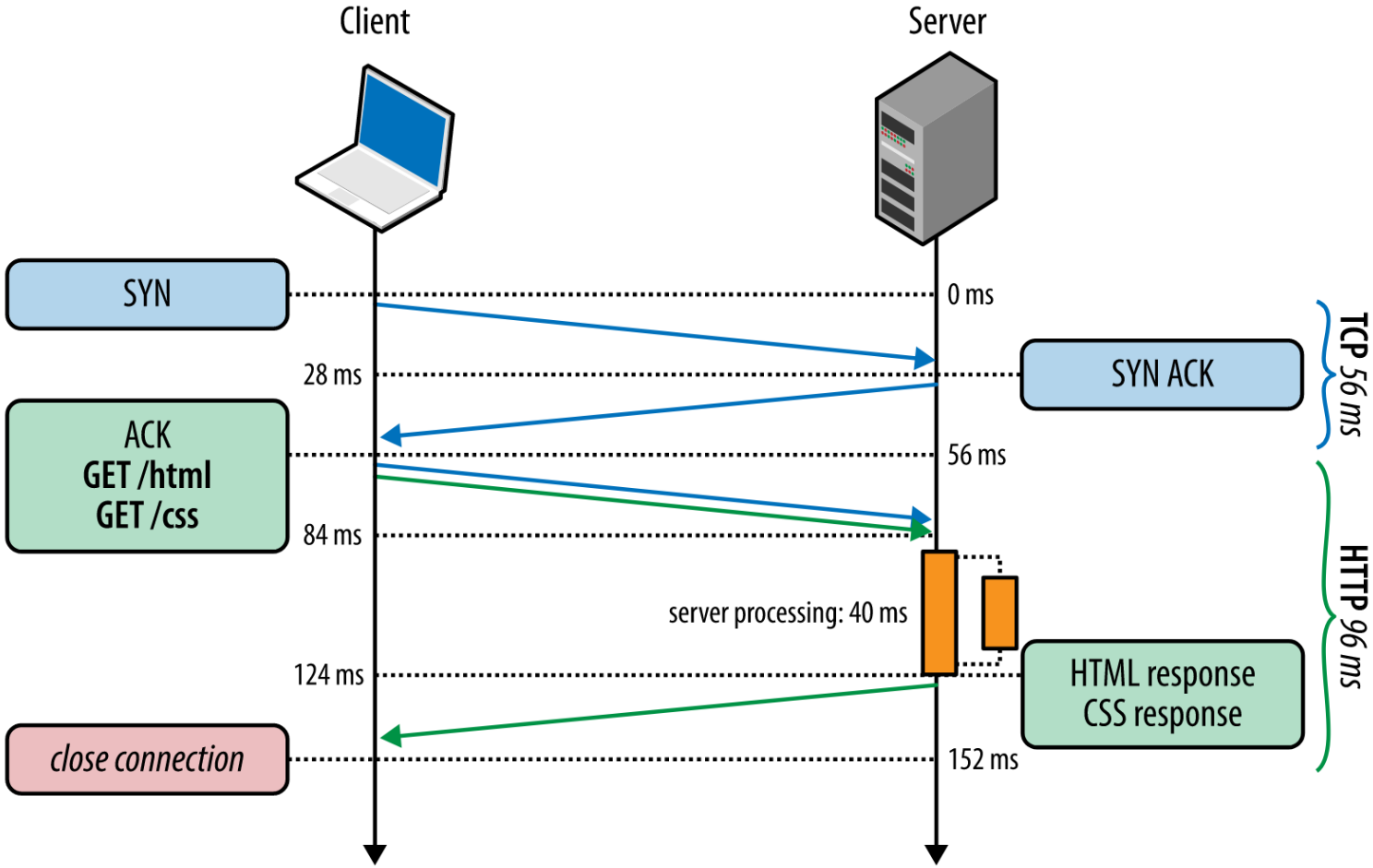


HTTP Pipelining (FIFO)



Отправка нескольких запросов в одном соединении, не ожидая ответов

Параллельная обработка запросов



Проблемы HTTP/1.x

- Нельзя "перемешивать" ответы на разные запросы в рамках соединения
- Ответы возвращаются целиком в порядке их поступления на сервер
- Один медленный запрос может заблокировать все запросы после него
  - т.н. проблема head-of-line blocking, ср. с доставкой пакетов в TCP
- При параллельной обработке серверы вынуждены буферизовать ответы
- Ошибка при обработке одного запроса может повлечь за собой закрытие соединения, повторную отправку и обработку всех последующих запросов
- Промежуточные серверы (прокси) могут не поддерживать или затруднять использование pipelining

Обход проблем

- Браузеры открывают сразу несколько (до 6) соединений с каждым сервером
- Дополнительные накладные расходы на стороне клиента и сервера
- Повышенная сложность реализации клиента
- Параллелизм ограничен небольшим числом, долгие запросы могут блокировать выполнение остальных
  - отсюда другой workaround - domain sharding
- Никак не решает проблемы DNS-запросов и медленного старта TCP

Накладные расходы

- Для обеспечения обратной совместимости заголовки и другие метаданные передаются как текст
- С учётом cookies их размер может составлять несколько КБ в несжатом виде

```
$> curl --trace-ascii - -d'{"msg":"hello"}' http://www.igvita.com/api

== Info: Connected to www.igvita.com
=> Send header, 218 bytes
POST /api HTTP/1.1
User-Agent: curl/7.24.0 (x86_64-apple-darwin12.0) libcurl/7.24.0 ...
Host: www.igvita.com
Accept: */*
Content-Length: 15
Content-Type: application/x-www-form-urlencoded
=> Send data, 15 bytes (0xf)
{"msg":"hello"}

<= Recv header, 134 bytes
HTTP/1.1 204 No Content
Server: nginx/1.0.11
Via: HTTP/1.1 GWA
Date: Thu, 20 Sep 2012 05:41:30 GMT
Cache-Control: max-age=0, no-cache
```

Уменьшение накладных расходов

- Загрузка нескольких ресурсов с помощью одного HTTP-запроса
- Concatenation - несколько файлов объединяются в один файл (JS, CSS)
- Spriting - аналогичный подход для изображений
- Resource inlining - вставка содержимого ресурсов в документ
- Привносят новые проблемы
  - например, загрузка всех файлов при обновлении одного
- Очередные "обходы" органичений HTTP/1.x

В чем причина проблем?

- Ограничения HTTP/1.x
- Протокол TCP плохо подходит для эпизодических коротких взаимодействий в стиле запрос-ответ с небольшими данными
  - изначально был оптимизирован для длительной передачи относительно больших объемов данных

SPDY

- 2009 - экспериментальный протокол от Google
  - улучшение производительности HTTP, в первую очередь - уменьшение задержек
  - загрузка страниц быстрее на 50%
- 2012
  - поддержка SPDY в основных браузерах
  - начало работ над новой спецификацией HTTP на основе SPDY

HTTP/2

> There is emerging implementation experience and interest in a protocol that **retains the semantics** of HTTP without the legacy of HTTP/1.x **message framing and syntax**, which have been identified as hampering performance and encouraging misuse of the underlying transport.

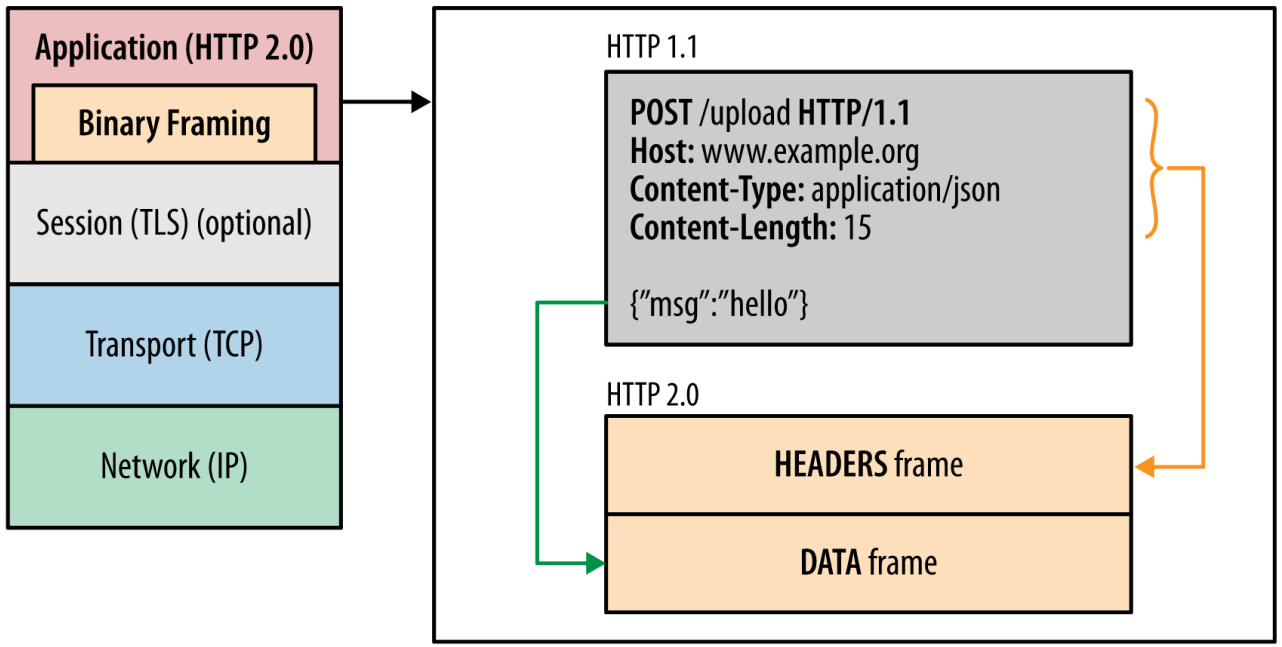
> The working group will produce a specification of a new expression of HTTP's current semantics in **ordered, bi-directional streams**. As with HTTP/1.x, the primary target transport is TCP, but it should be possible to **use other transports**.

HTTP/2

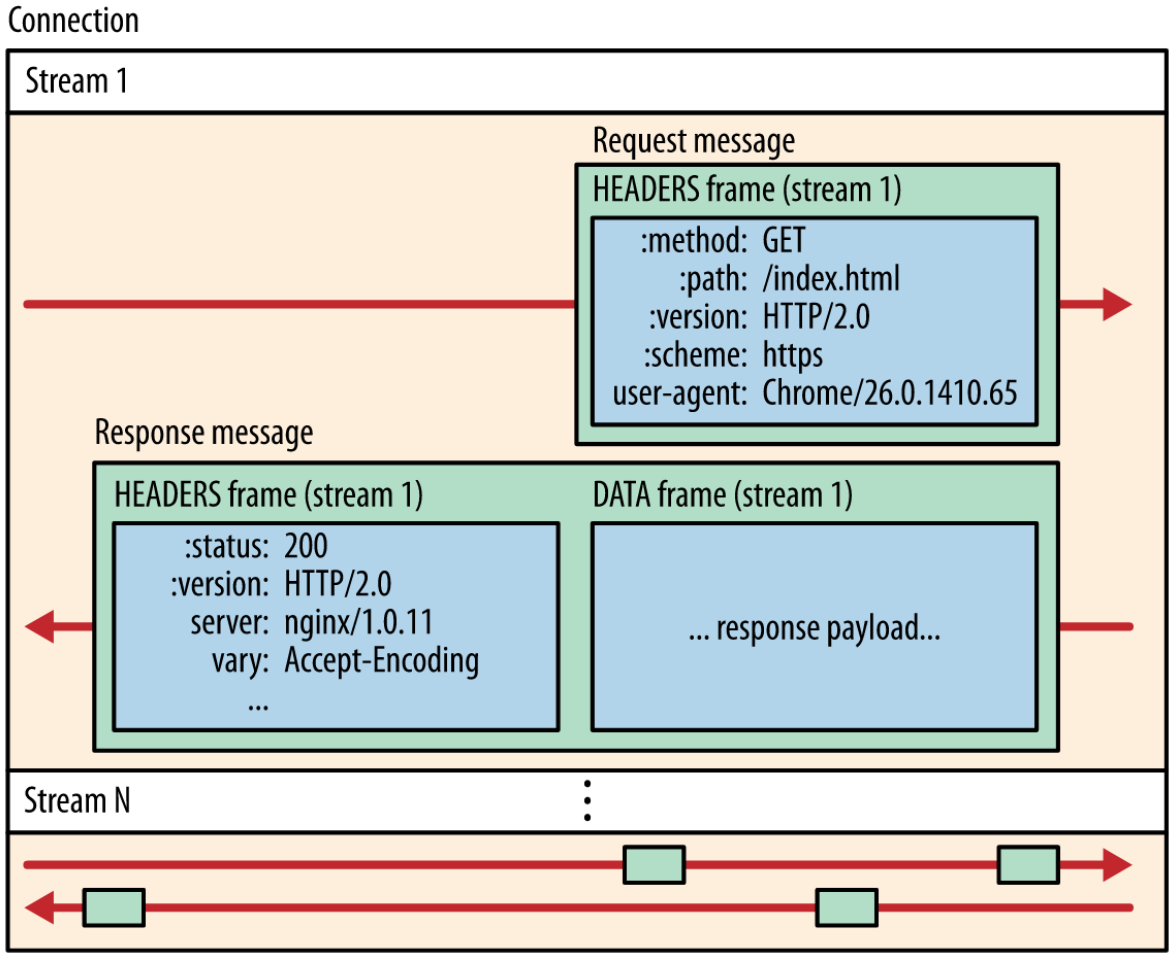
- 2015 - RFC 7540 (HTTP/2), RFC 7541 (HPACK)
- Не меняет семантику HTTP (методы, статусы, заголовки, URI), что позволяет безболезненно мигрировать существующие приложения
- Основные новшества находятся на уровне передачи данных между клиентом и сервером
  - Разбиение и передача данных в бинарном формате
  - Мультиплексирование запросов и ответов
  - Сжатие заголовков, RFC 7541 (HPACK)
  - Приоритезация запросов
  - Server push

Представление данных

- Бинарный протокол
- Данные, которыми обмениваются клиент и сервер, разбиты на фрагменты



Передача данных

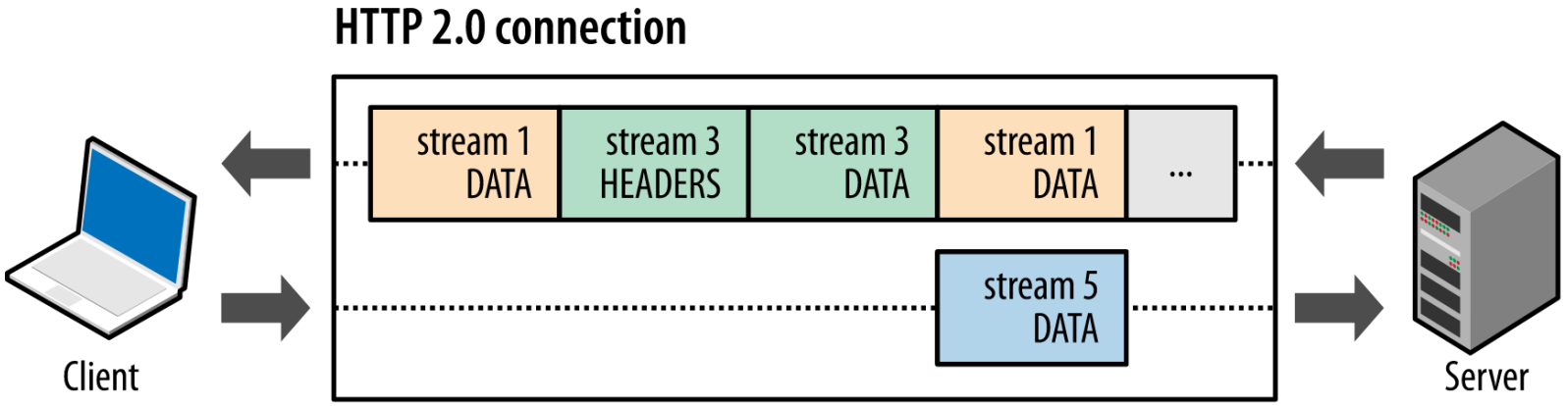


Передача данных

- *Поток (stream)* - двусторонний поток байтов по установленному соединению
  - В рамках одного соединения может быть несколько потоков
  - Поток имеет уникальный идентификатор и приоритет
- *Сообщение (message)* - запрос или ответ
  - Сообщение разбивается на один или несколько кадров
- *Кадр (frame)* - единица передачи данных
  - имеет заголовок, содержащий id потока
  - содержит определенный тип данных (DATA, HEADER, SETTINGS, PING...)
  - размер кадра DATA обычно ограничен ~16 KB
- Кадры разных потоков могут чередоваться при передаче внутри соединения

Мультиплексирование запросов и ответов

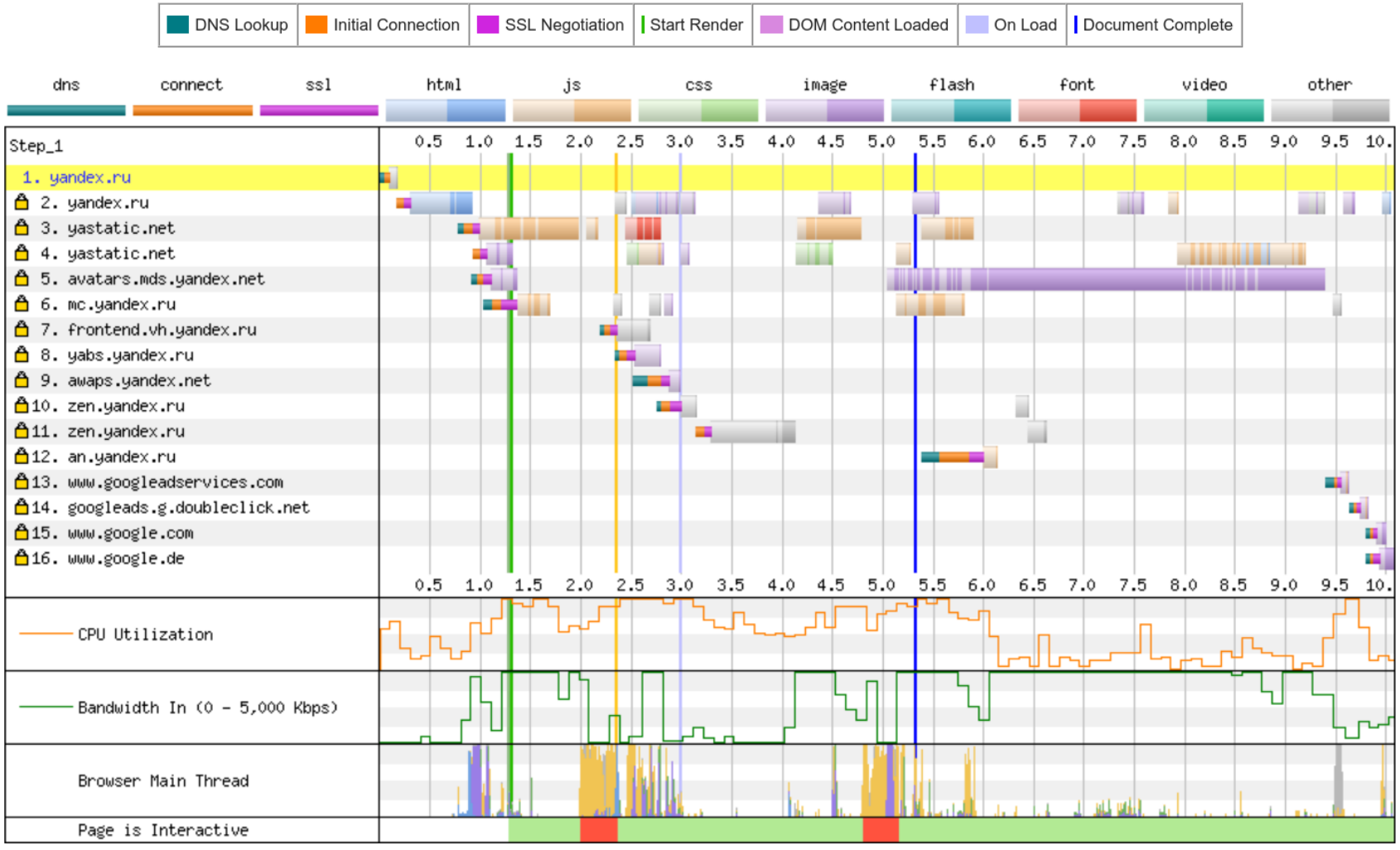
Новая модель передачи данных позволяет параллельно отправлять несколько запросов и получать несколько ответов, не блокируясь на каждом из них, через одно соединение



Мультиплексирование запросов и ответов

- Позволяет уменьшить задержки и более эффективно использовать ресурсы сети
- Устраняет head-of-line blocking
  - только на уровне HTTP, на уровне TCP проблема остается
- Делает ненужными ранее описанные трюки для HTTP/1.x
- Достаточно одного соединения с сервером

Connection View



Приоритезация потоков

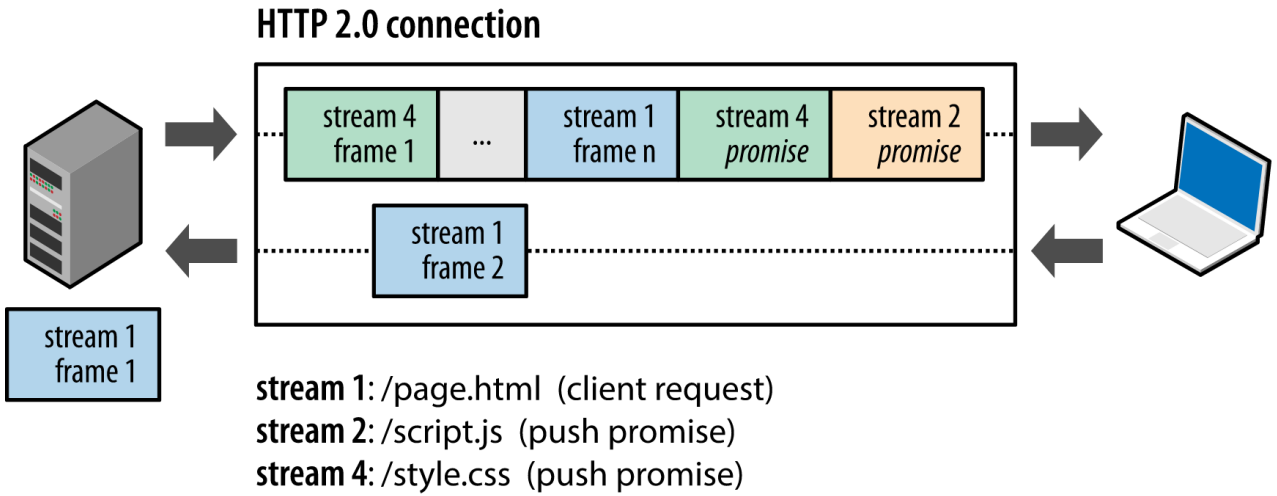
- Каждый поток имеет вес (1-256)
- Поток может зависеть от других потоков
- Клиент может организовывать потоки в дерево и определять их приоритеты, в том числе динамически
- Сервер учитывает приоритеты потоков при получении/обработке запросов и отправке ответов

Управление передачей (flow control)

- Позволяет принимающей стороне регулировать скорость отправки данных отправителем
- HTTP/2 содержит механизм управления передачей на уровне потоков и всего соединения (применяется только к DATA кадрам)
- Получатель указывает размер окна (по умолчанию 64 КБ)
- Отправитель уменьшает окно при отправке данных и увеличивает при получении кадра WINDOW_UPDATE от получателя
- Может настраиваться на промежуточных узлах (hop-by-hop) в отличие от TCP (end-to-end)
- Спецификация не предписывает конкретный алгоритм, он определяется реализацией клиента/сервера

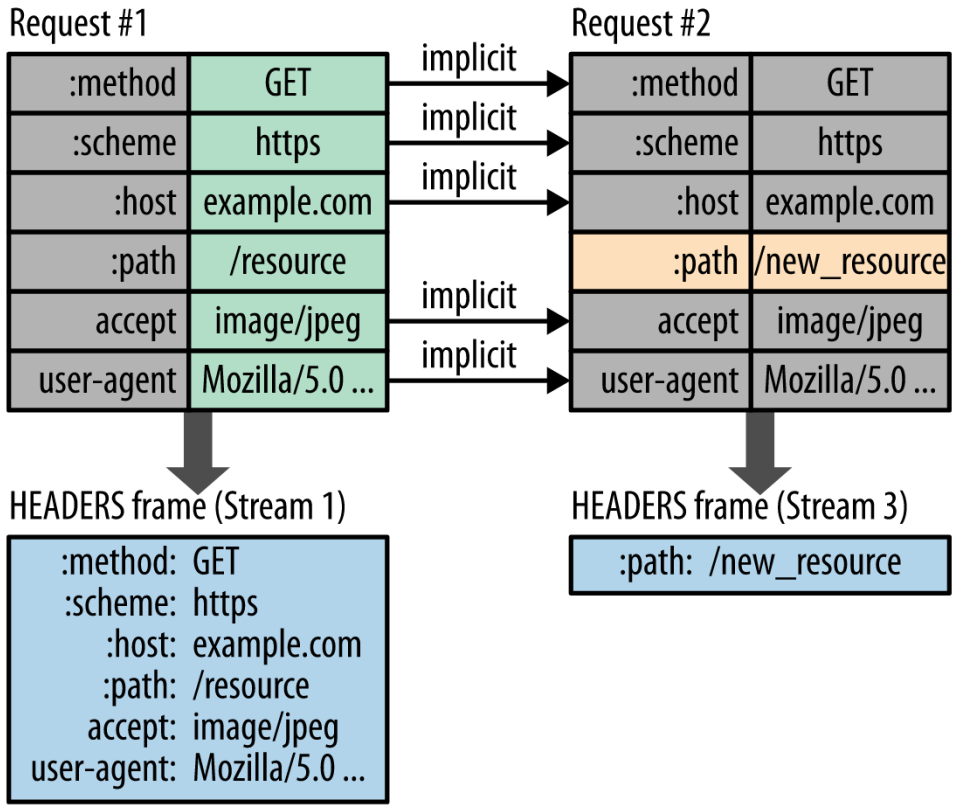
Server Push

- Возможность сервера отправить несколько ответов на один запрос клиента
  - предварительная загрузка ресурсов, отправка уведомлений...
- Данные передаются в отдельном потоке
  - перед отправкой данных сервер отправляет кадр PUSH_PROMISE
  - клиент может отказаться от получения данных



Сжатие заголовков

Заголовки запросов и ответов сжимаются с помощью формата HPACK



Переход с HTTP/1.x на HTTP/2

```
GET /page HTTP/1.1
Host: server.example.com
Connection: Upgrade, HTTP2-Settings
Upgrade: h2c
HTTP2-Settings: (SETTINGS payload)

HTTP/1.1 200 OK
Content-length: 243
Content-type: text/html

(... HTTP/1.1 response ...)

          (or)

HTTP/1.1 101 Switching Protocols
Connection: Upgrade
Upgrade: h2c

(... HTTP/2 response ...)
```

Дальнейшее развитие HTTP

- The moment you remove one performance bottleneck, you unlock the next one
- После устранения проблем HTTP/1.x настала очередь TCP
- Спецификации HTTP не предписывали использование именно TCP

> HTTP communication usually takes place over TCP/IP connections... This does not preclude HTTP from being implemented on top of any other protocol on the Internet, or on other networks. HTTP only presumes a reliable transport; any protocol that provides such guarantees can be used; the mapping of the HTTP/1.1 request and response structures onto the transport data units of the protocol in question is outside the scope of this specification.

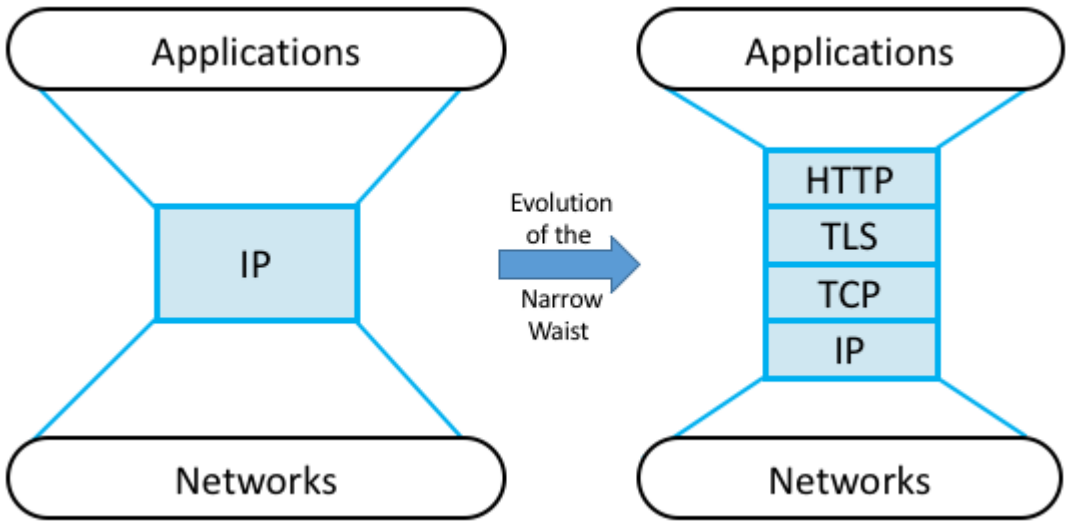
HTTP/3

- HTTP/3 реализует семантику HTTP поверх нового протокола транспортного уровня QUIC
- Черновая спецификация (статус Internet Draft)
- Поддержка в Chrome (декабрь 2019) и Firefox (январь 2020)
- Usage statistics of QUIC for websites (4.8%)
  - HTTP/2: 48.3%

HTTP как универсальный протокол

> HTTP is a **generic interface protocol for information systems**. It is designed to hide the details of how a service is implemented by presenting a **uniform interface** to clients that is independent of the types of resources provided. Likewise, servers do not need to be aware of each client's purpose: an HTTP request can be considered in isolation rather than being associated with a specific type of client or a predetermined sequence of application steps. The result is a protocol that can be used effectively in many different contexts and for which implementations can evolve independently over time.
>
> (HTTP/1.1, RFC 7230, 2014)

HTTP как универсальный протокол



Web как платформа для приложений

- Изначально:
  - доступ пользователей к информации и общение друг с другом
  - стандартные клиентские приложения (браузеры)
  - пользовательские интерфейсы
- Сейчас:
  - взаимодействие между произвольными приложениями по сети
  - программные интерфейсы (API)
  - веб-сервисы

Сервис-ориентированная архитектура

- Подход к проектированию распределенных систем
  - основан на использовании слабо связанных (loosely coupled) компонентов и единых соглашений по взаимодействию
- Сервис - доступный по сети компонент с описанным набором функций
  - Является автономным
  - С точки зрения клиентов является "черным ящиком"
  - Может включать в себя или использовать в своей работе другие сервисы

Аспекты слабой связанности

- Клиенты могут динамически определять адрес нужного им сервиса
- Клиенты и сервисы могут взаимодействовать не работая одновременно (непрямые взаимодействия)
- Реализация сервиса может развиваться, не влияя на работу существующих клиентов
- Сервис может повторно использоваться в рамках различных приложений

Контракт между сервисом и клиентом

- В простейшем случае - описание интерфейса сервиса, включая операции и форматы сообщений
- Дополнительно в контракт могут входить гарантии качества обслуживания (QoS, SLA)
- Два подхода к определению контракта:
  - Contract first - разработка сервиса начинается с описания контракта на некотором языке
  - Contract last - контракт автоматически генерируется по реализации сервиса

Jeff Bezos API Mandate

1. All teams will henceforth expose their data and functionality through service interfaces.
2. Teams must communicate with each other through these interfaces.
3. There will be no other form of interprocess communication allowed: no direct linking, no direct reads of another team's data store, no shared-memory model, no back-doors whatsoever. The only communication allowed is via service interface calls over the network.
4. It doesn't matter what technology they use. HTTP, Corba, Pubsub, custom protocols -- doesn't matter. Bezos doesn't care.
5. All service interfaces, without exception, must be designed from the ground up to be externalizable. That is to say, the team must plan and design to be able to expose the interface to developers in the outside world. No exceptions.
6. Anyone who doesn't do this will be fired.
7. Thank you; have a nice day!

Jeff Bezos API Mandate (Amazon, 2002)

SOAP-сервисы

- Первое поколение веб-сервисов (2000-е)
- Стек стандартов WS-*
  - SOAP, WSDL, WS-Addressing, WS-ReliableMessaging, WS-Security...
- Ориентированы на интеграцию разнородных систем в рамках предприятий и между ними
  - Некоторые системы могут ничего не знать про Web (COBOL-программа на банковском мэйнфрейме)
  - Поддержка разных языков, платформ и моделей программирования

Simple Object Access Protocol (SOAP)

- Клиент и сервис обмениваются сообщениями в формате XML
- Помимо модели "запрос-ответ" возможны другие модели
  - очереди сообщений, publish/subscribe
- Сообщение включает заголовок и тело
  - Заголовок содержит метаданные и служебную информацию
  - Тело сообщения формируется в соответствии с описанием интерфейса сервиса
- Использование XML позволяет легче менять структуру сообщений, не "ломая" существующих клиентов
  - Ср. с бинарными форматами в классических реализациях RPC
- Может передавать сообщения поверх различных протоколов
  - HTTP, SMTP, TCP, JMS, MQ, IIOP...

Другие элементы стека WS-*

- Web Services Description Language (WSDL)
  - XML-язык для описания веб-сервисов
  - Автоматическая генерация кода для клиентов и серверов по WSDL-документу
- Universal Description, Discovery, and Integration (UDDI)
  - Механизм для регистрации и поиска веб-сервисов
  - Не получил широкого распространения
- Качество обслуживания
  - Надежная доставка сообщений, безопасность, поддержка распределенных транзакций...
  - Разработчик сервиса может использовать готовую реализацию WS-* для обеспечения требуемых гарантий

SOAP-сервисы

- Преимущества
  - Возможность использования разных протоколов для транспорта
  - Полноценный язык описания интерфейсов сервисов
  - Поддержка асинхронных вызовов и передачи сообщений
  - Автоматическая генерация кода по WSDL
- Недостатки
  - Сложность стандартов, много возможностей и альтернативных опций
  - Продвинутые функции часто доступны только в коммерческих продуктах
  - Проблемы с интероперабельностью между реализациями, зависимость от вендора
  - Накладные расходы на трансляцию между XML и структурами в памяти программ

SOAP-сервисы и HTTP

- SOAP-сервисы используют HTTP исключительно как транспорт и не интегрированы в Web
- Все операции сервиса доступны через один URI
- Используется только метод POST

REST-сервисы

- Второе поколение веб-сервисов (2010-е)
- Основаны на походе REpresentational State Transfer (REST)
  - Fielding R. Architectural Styles and the Design of Network-based Software Architectures (2000)
  - Изначально был создан для построения масштабируемых распределенных гипермедийных (hypermedia) систем
  - Использовался при проектировании протокола HTTP
- Лучше интегрированы со стандартами Web
- Сейчас основной вариант реализации веб-сервисов

Принципы REST

- Сервис предоставляет доступ к набору ресурсов, идентифицируемых с помощью URI (Uniform Resourc Identifier). Все операции проводятся в контексте некоторого ресурса.
- Манипуляции над ресурсами производятся с помощью фиксированного набора операций (создание, чтение, изменение, удаление). Применительно к HTTP это методы POST, GET, PUT/PATCH, DELETE.
- Ресурсы могут иметь несколько представлений в различных форматах. С ресурсом могут быть связаны метаданные, используемые для кэширования, согласования форматов, проверки целостности, авторизации и т.д.
- Взаимодействия с ресурсами не предполагают хранения дополнительного состояния на стороне сервиса (stateless), каждый запрос содержит все необходимые данные для выполнения операции.

Дизайн REST API

- Определение ресурсов
- Дизайн URI ресурсов (nouns vs verbs, иерархии ресурсов)
- Отображение действий на методы HTTP (CRUD, non-CRUD)
- Представления ресурсов и форматы сообщений
- Связи между ресурсами и навигация

Task	Method	Path
Create a new task	POST	/tasks
Delete an existing task	DELETE	/tasks/{id}
Get a specific task	GET	/tasks/{id}
Search for tasks	GET	/tasks
Update an existing task	PUT	/tasks/{id}

Описание REST API

- Описание API в произвольной форме
  - Список предоставляемых ресурсов и их URI
  - Методы, доступные для каждого ресурса
  - Форматы запросов/ответов
  - Возможные коды статусов для каждого метода ресурса
- Спецификация OpenAPI
  - Инструменты для генерации описания по коду сервиса (например, Swagger)

REST-сервисы

- Преимущества
  - Полноценное использование стандартов и инфраструктуры Web
  - Низкий барьер, легковесные реализации, больше количество открытых решений
  - Масштабируемость за счет отсутствия состояния и поддержки кэширования
  - Возможность использования различных форматов сообщений, например JSON
  - Фиксированные методы и гибкий формат данных позволяют легче развивать сервисы
- Недостатки
  - Ориентация на CRUD-приложения
  - Поддержка только синхронных взаимодействий в стиле запрос-ответ
  - Отсутствие четко прописанных правил реализации сервисов
  - Нет единого формата сообщений, стандарт описания интерфейсов появился недавно
  - Ограничения при передаче параметров GET-операций в URI

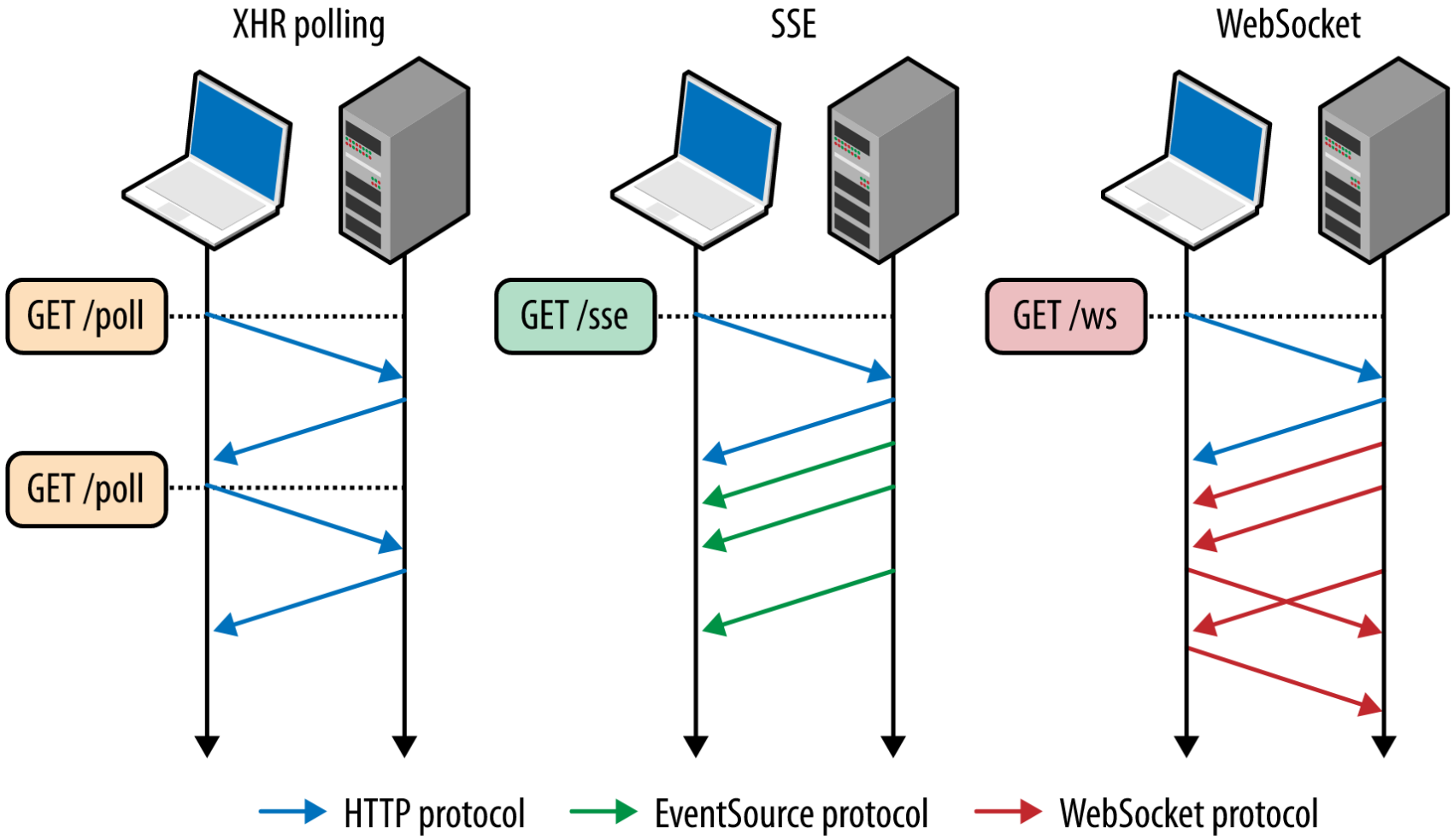
Веб-сервисы vs RPC

- Синхронная модель "запрос-ответ"
- Формат сообщений
- Язык описания интерфейсов
- Генерация кода
- Обеспечение совместимости клиентов и серверов
- Области применения

Прямой обмен сообщениями

- Клиент и сервер могут отправлять сообщения друг другу
  - Симметричный двусторонний канал для асинхронных взаимодействий
  - Для отправки данных от сервера не требуется запрос клиента
- Примеры
  - Сокеты, ZeroMQ, MPI, Erlang...
  - Web: WebSocket
  - RPC: bidirectional streams в gRPC
- Как в таком случае описываются интерфейсы?

Протокол WebSocket



Литература

- Grigorik I. High Performance Browser Networking (главы 9, 11, 12)
- Pollard B. HTTP/2 in Action (главы 1, 4, 9)
- Webber J., Parastatidis S., Robinson I. REST in Practice: Hypermedia and Systems Architecture (главы 1-4)

Дополнительно

- Grigorik I. High Performance Browser Networking (главы 10, 17)
- Другие главы из HTTP/2 in Action
- A QUICk Introduction to HTTP/3
- REST API Tutorial
- Vogels W. Web Services Are Not Distributed Objects (2003)
- Vinoski S. Putting the "Web" into Web Services (2002)
- Pautasso C., Zimmermann O., Leymann F. RESTful Web Services vs. “Big” Web Services:Making the Right Architectural Decision (2008)