

Протокол HTTP и веб-сервисы

Олег Сухорослов

Распределенные системы

Факультет компьютерных наук НИУ ВШЭ

19.09.2022

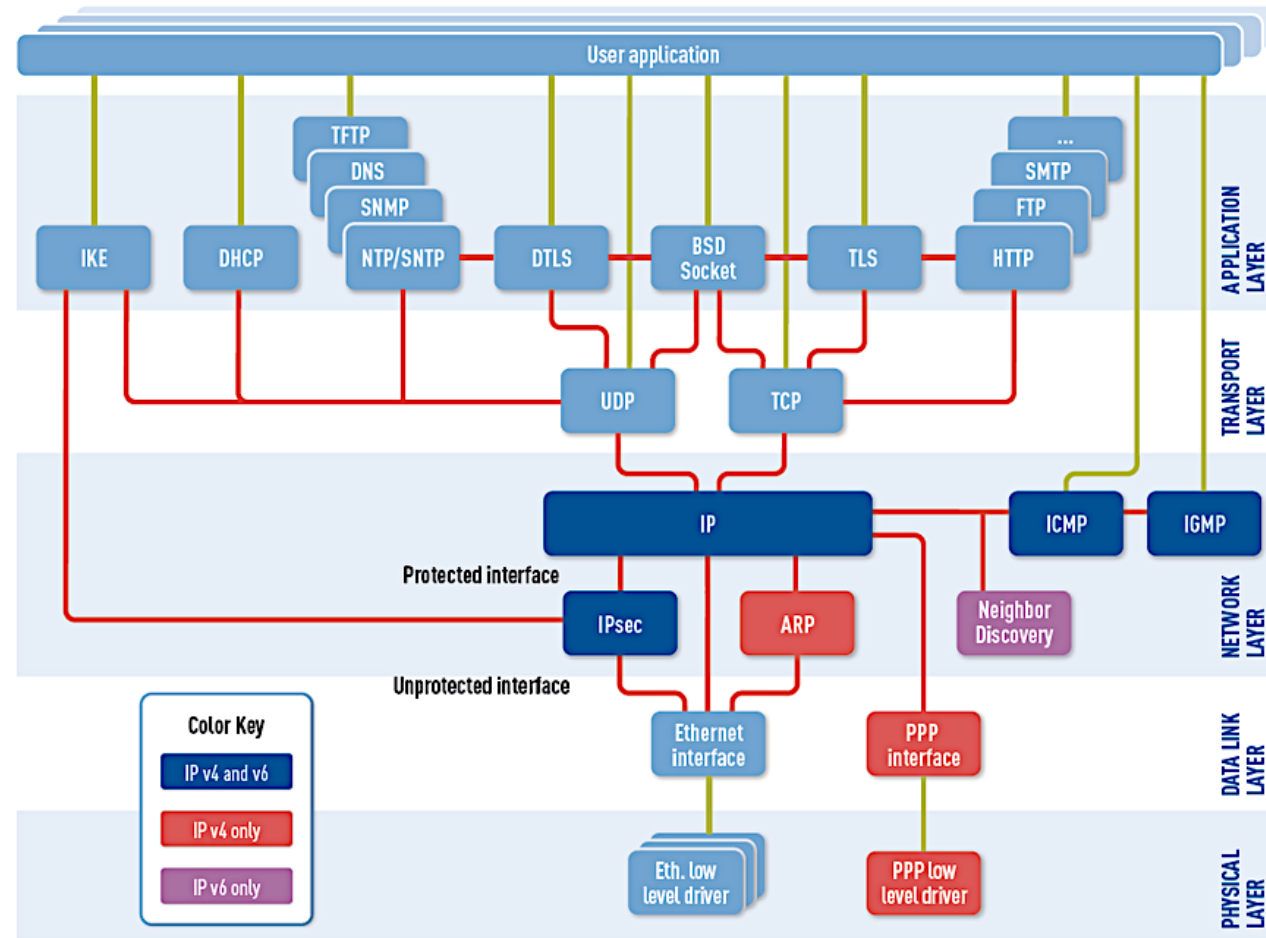
План

- Эволюция протокола HTTP
- Веб-сервисы, REST, сравнение с RPC

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 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**, object-oriented protocol which can be used for **many tasks**, such as name servers and distributed object management systems, through **extension** of its request methods. 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 (Uniform Resource Identifier)
 - только чтение, не меняет ресурс и 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 |

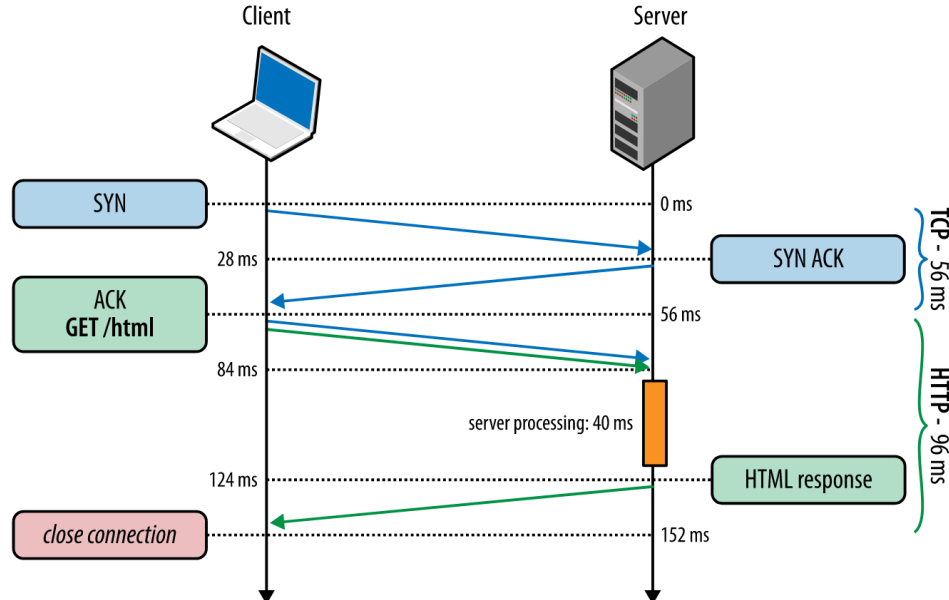
- **Безопасный метод** - не подразумевает эффектов на стороне сервера (read-only)
- **Идемпотентный метод** - несколько идентичных запросов имеют такой же эффект, что однократный запрос

Эволюция веб-приложений

- Одиночный гипертекстовый документ
- Веб-страница
 - 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

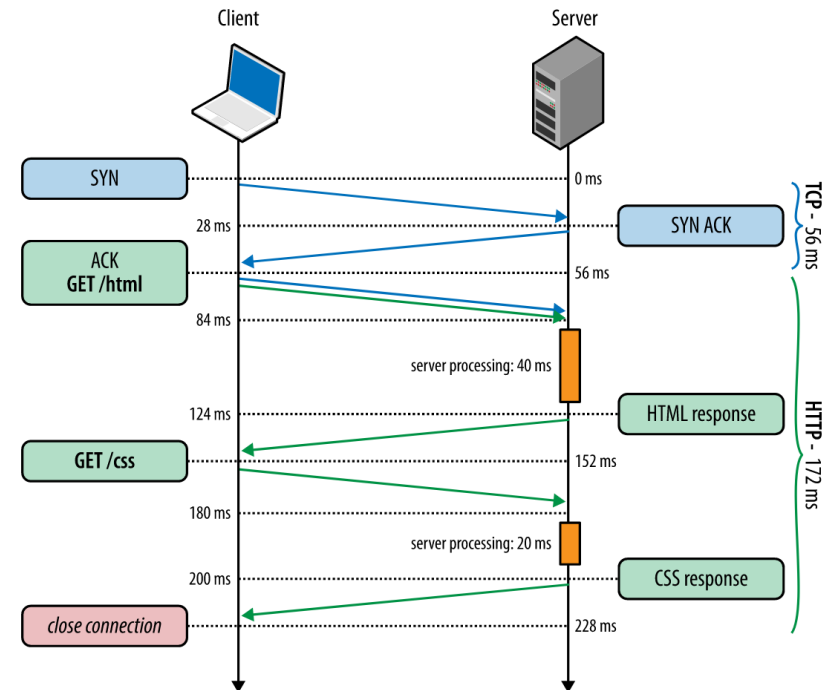
Задержка на стороне клиента

- 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

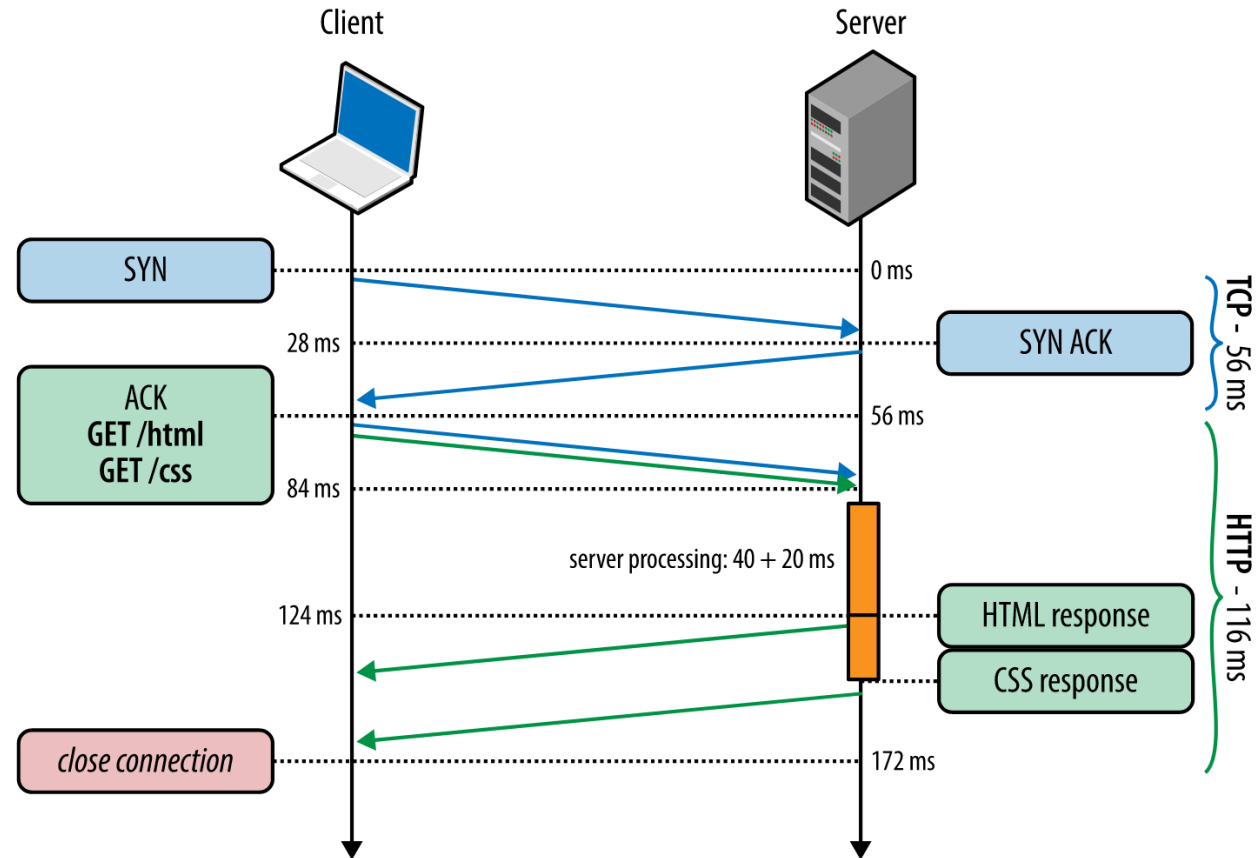


Keepalive Connections

- 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`

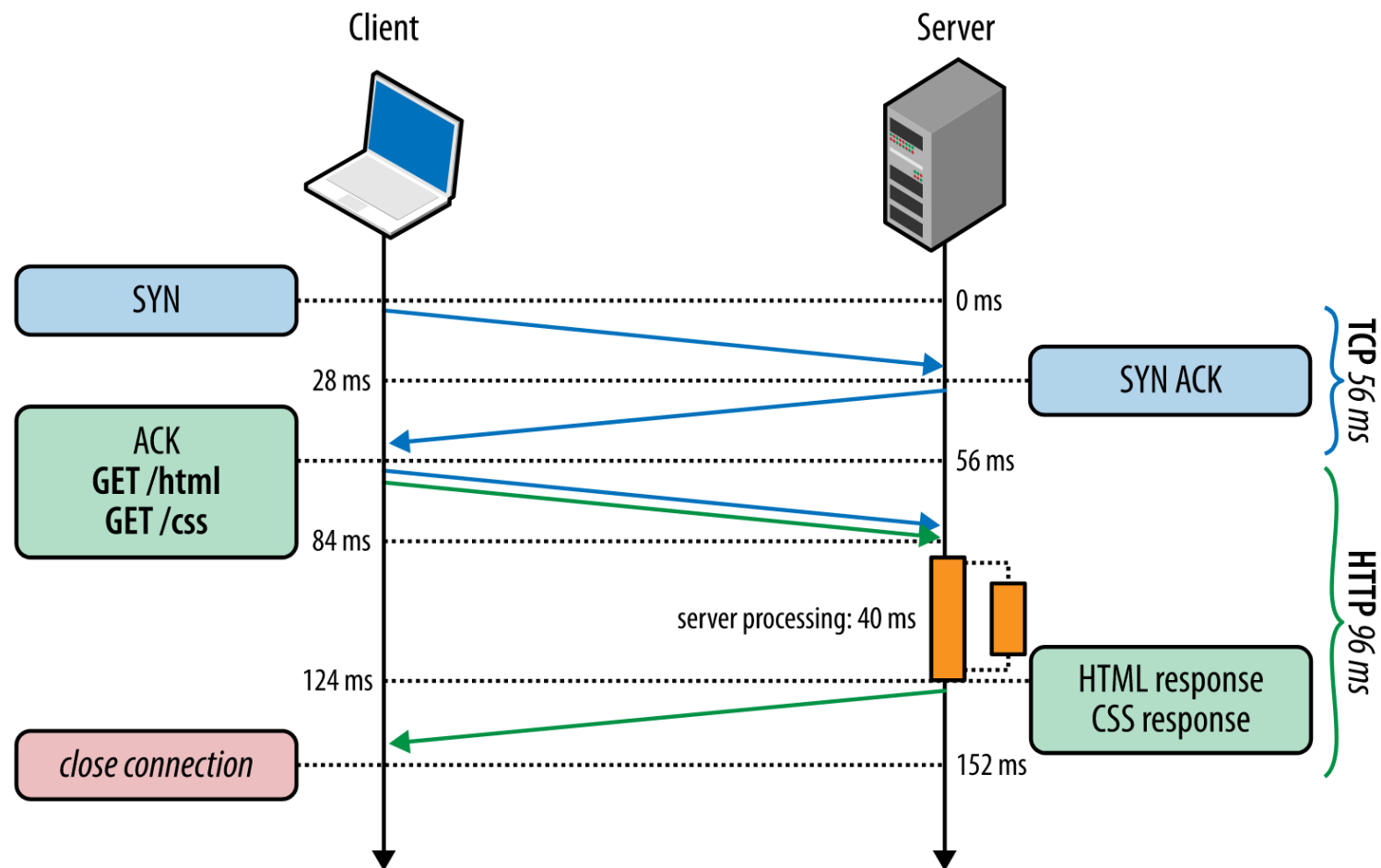


HTTP Pipelining (FIFO)



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 (protocol overhead)

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

Уменьшение числа запросов

- 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" ограничений 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.

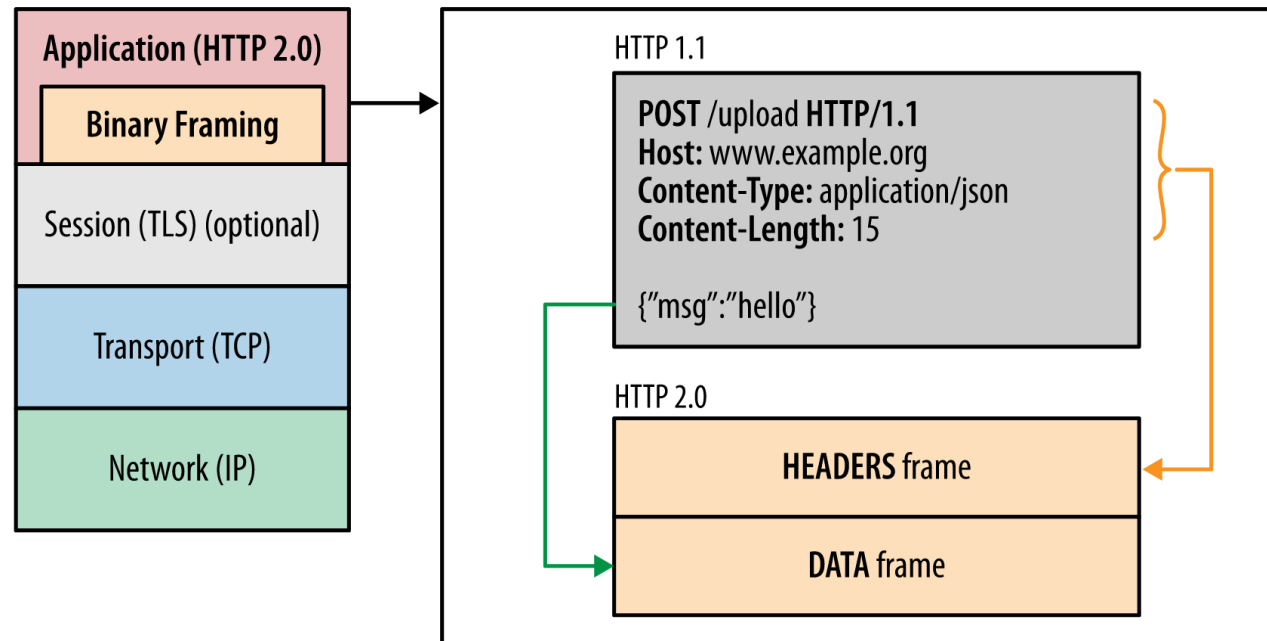
The working group will produce a specification of a new expression of HTTP's current semantics in **ordered, bi-directional streams**. As with HTTP/1.x, the primary target transport is TCP, but it should be possible to **use other transports**.

HTTP/2

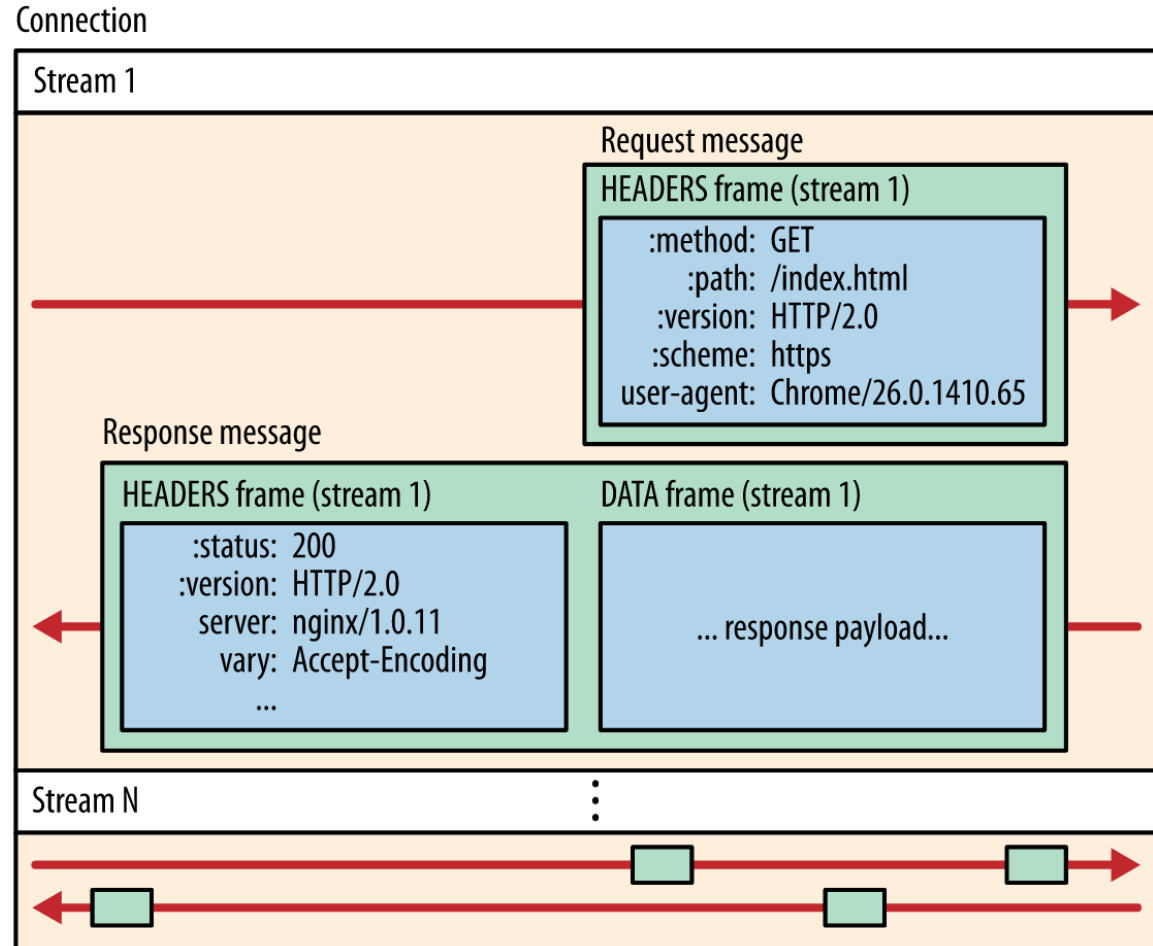
- 2015 - RFC 7540 (HTTP/2), RFC 7541 (HPACK)
- Не меняет семантику HTTP (методы, статусы, заголовки, URI), что позволяет безболезненно мигрировать существующие приложения
- Основные новшества находятся на уровне передачи данных между клиентом и сервером
 - Разбиение и передача данных в бинарном формате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

Представление данных

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



Передача данных

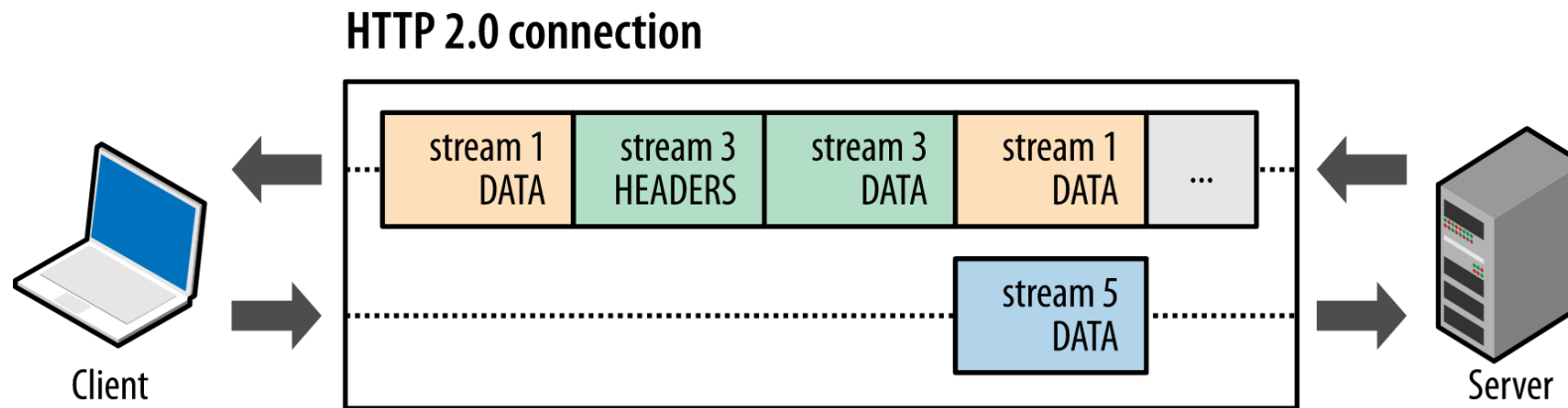


Передача данных

- *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 через одно соединение несколько запросов и получать несколько ответов, не блокируясь на каждом из них



Мультиплексирование запросов и ответов

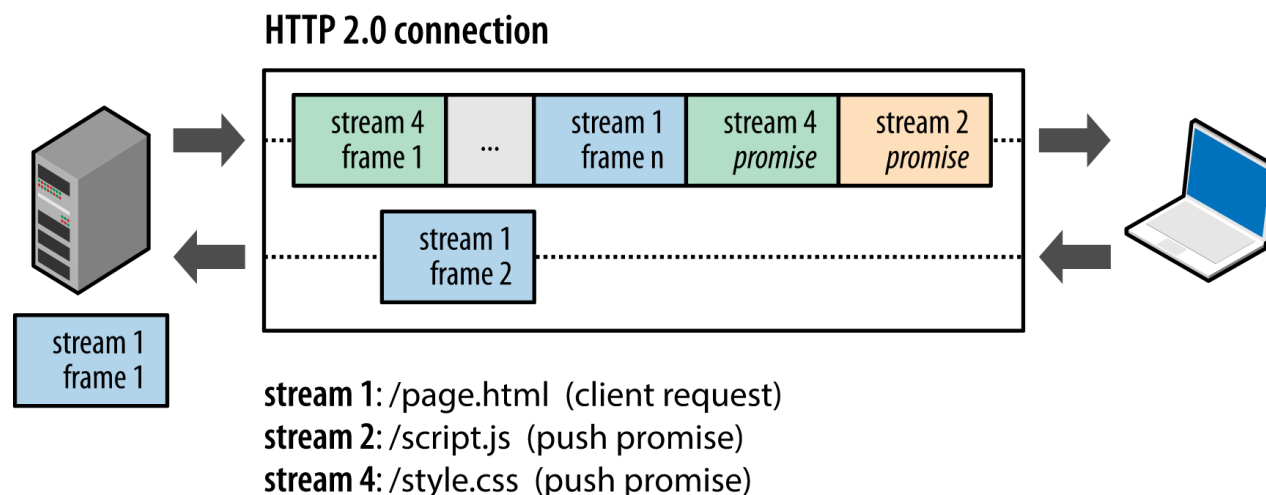
- Позволяет уменьшить задержки и более эффективно использовать ресурсы сети
- Устраняет head-of-line blocking
 - только на уровне HTTP, на уровне TCP проблема остается
- Делает ненужными описанные обходы проблем HTTP/1.x
- Достаточно одного соединения с сервером

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

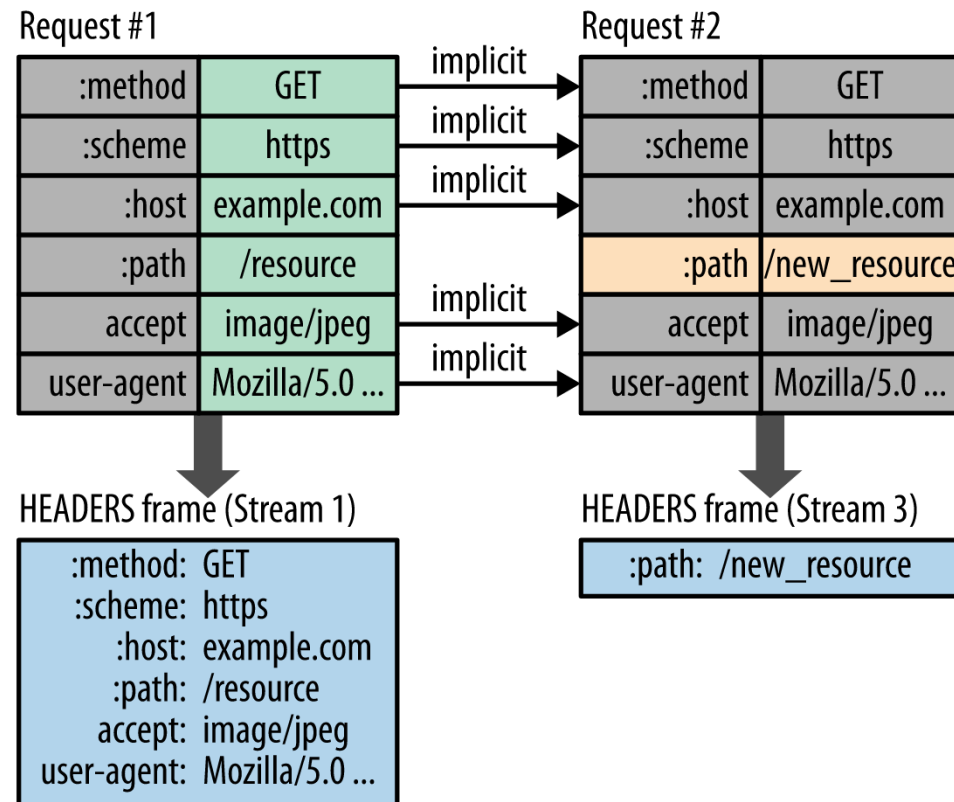
Server Push

- 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
- Спецификация RFC 9114 (статус Proposed Standard)
- Поддержка браузерами: 75%
- Поддержка сайтами: 25% (HTTP/2: 44%)

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)

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

REST-сервисы

- Второе поколение веб-сервисов (2010-е)
 - Первое - SOAP-сервисы на базе стека стандартов WS-* (2000-е)
- Лучше интегрированы со стандартами Web чем SOAP-сервисы
- Сейчас основной подход к реализации веб-сервисов
- Основаны на архитектурном стиле 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 систем
 - Использовался при проектировании протокола HTTP

Принципы REST

- Клиент-серверная модель
- Сервер предоставляет доступ к набору **ресурсов**, идентифицируемых с помощью URI
- Клиенты взаимодействуют с ресурсами через фиксированный набор **операций**
- Ресурсы могут иметь несколько **представлений** в различных форматах
- С ресурсом могут быть связаны **метаданные**, используемые для разных целей (кэширования, согласования форматов, авторизации...)
- В результате обработки запросов могут изменяться **состояния** ресурсов и клиентов
- Сервер не хранит информацию о состоянии клиентской **сессии** между запросами
 - протокол без сохранения состояния (stateless protocol)
 - каждый запрос должен содержать всю необходимую информацию для его выполнения
 - информация о состоянии сессии может быть передана сервером какому-либо другому сервису (например, базе данных)

Интерфейс REST-сервиса (REST API)

- Определение ресурсов
- Дизайн URI ресурсов (nouns vs verbs, иерархии ресурсов)
- Отображение действий на методы HTTP
- 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} |

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
 - Ограничения при передаче параметров GET-операций в URI
 - Поддержка только синхронных взаимодействий в стиле запрос-ответ

REST vs RPC

- Схема взаимодействия
- Формат сообщений
- Описание интерфейсов
- Генерация кода
- Обеспечение совместимости клиентов и серверов
- Области применения

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)