Хипертекстов протокол

Основното е хипертекста

Най-мощното приложение на Internet е **WWW** (world wide web).

Като технология и идея е създадено от физика Тим Бърнър Лий през 1990 година с цел да улесни комуникацията на група учени.

web-технологията се базира на хипертекста.

Хипертекст

Това е текст, който съдържа в себе си информация как да бъде изобразен на екрана.

Изобразяването става чрез специална програма, наречена хипертекстов browser.

Хипертекст, URL, хиперлинк

Хипертекстът се оформя като съвкупност от страници.

Всяка страница си има уникално **URL** – уникален адрес, който еднозначно указва местоположението на страницата в Internet.

Хиперлинк – под част от текста стои URL на друга страница.

Т.е хипертекстовите страници съдържат препратки към други хипертекстови страници.

Не се изисква тези страници да се намират на същия сървър.

HTTP - за комуникация между сървър и клиент

Web - паяжина – страниците, които са пръснати и са свързани като паяжина чрез хиперлинковите връзки.

Browser е клиентът, който изтегля и изобразява страниците.

Web server е сървърът, който съхранява страниците.

За комуникация между browser-и и сървъри е създаден протокола **HTTP** (hyper text transfer protocol).

URL

URL съдържа протокол, име на домейн и пътя на страницата върху диска на сървъра.

При осъществяване на връзка между browser-а и сървъра по URL-то първо се търси по името на сървъра, а после по пътя върху диска на сървъра.

Страницата физически се изтегля от сървъра, предава се на browser-а и той я изобразява.

Една страница се прехвърля в рамките на една HTTP-сесия.

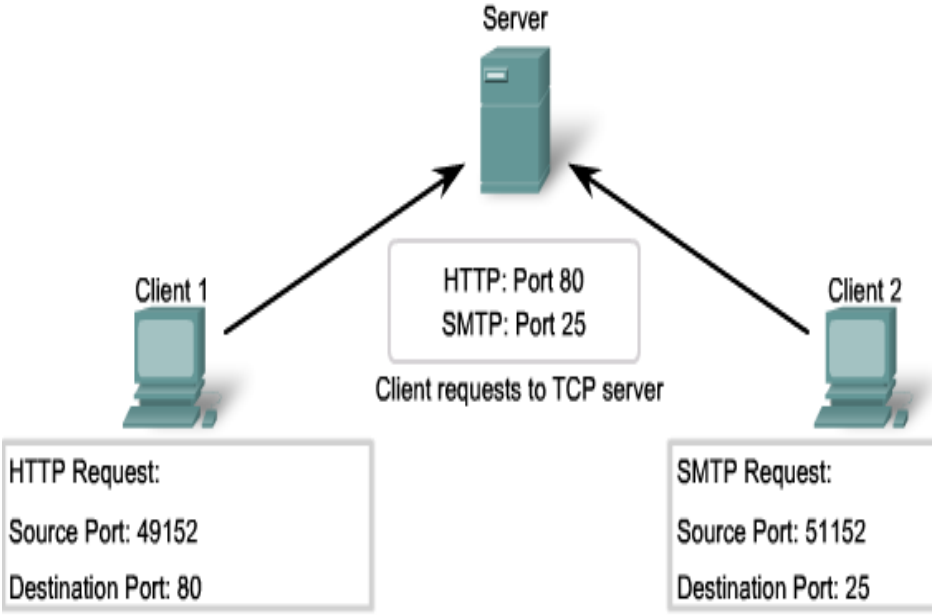
```
foo://example.com:8042/over/there/index.dtb?name=ferret#nose
\ /   \_________/ \__/\_________/ \_______/ \_________/ \__/
scheme authority port    path    filename      query  fragment
```

HTTP – TCP/80

Протоколът HTTP се базира на TCP.

HTTP клиентът отваря сесия на произволен порт с номер по-голям от 1024.

HTTP сървърът слуша за заявки на порт 80.



HTTP - request-response protocol

HTTP – request-response (заявка-отговор) протокол.

Клиентът отправя съобщение с HTTP заявка към сървъра.

Сървърът (съдържание, ресурси - HTML файлове и др.) връща съобщение с отговор:

- информация за състоянието - хедъра и заявеното съдържание – в тялото.

Посредници

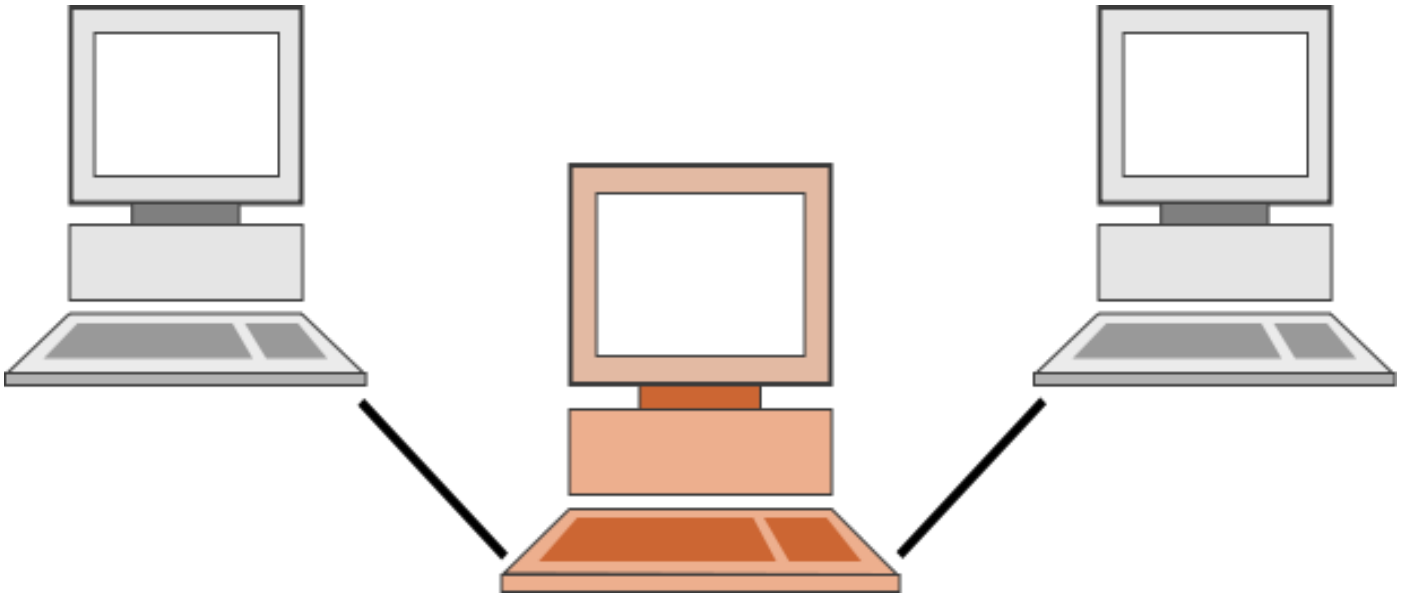
HTTP допуска "посредници” с цел повишаване на призводителността:

- web cache сървъри – предоставят съдържание за сметка на “оригиналния”;

- proxy сървъри – прави заявки за сметка на клиента.

HTTP е **stateless** протокол. Сървърът не задържа информация или състояние за всеки отделен потребител през цялото време на изпълнение на всяка заявка.

Proxy сървър



Проектът Tor (https://www.torproject.org/)

Мрежата **Tor** е група от сървъри, поддържани от доброволци, осигуряваща анонимността и сигурността на потребителите.

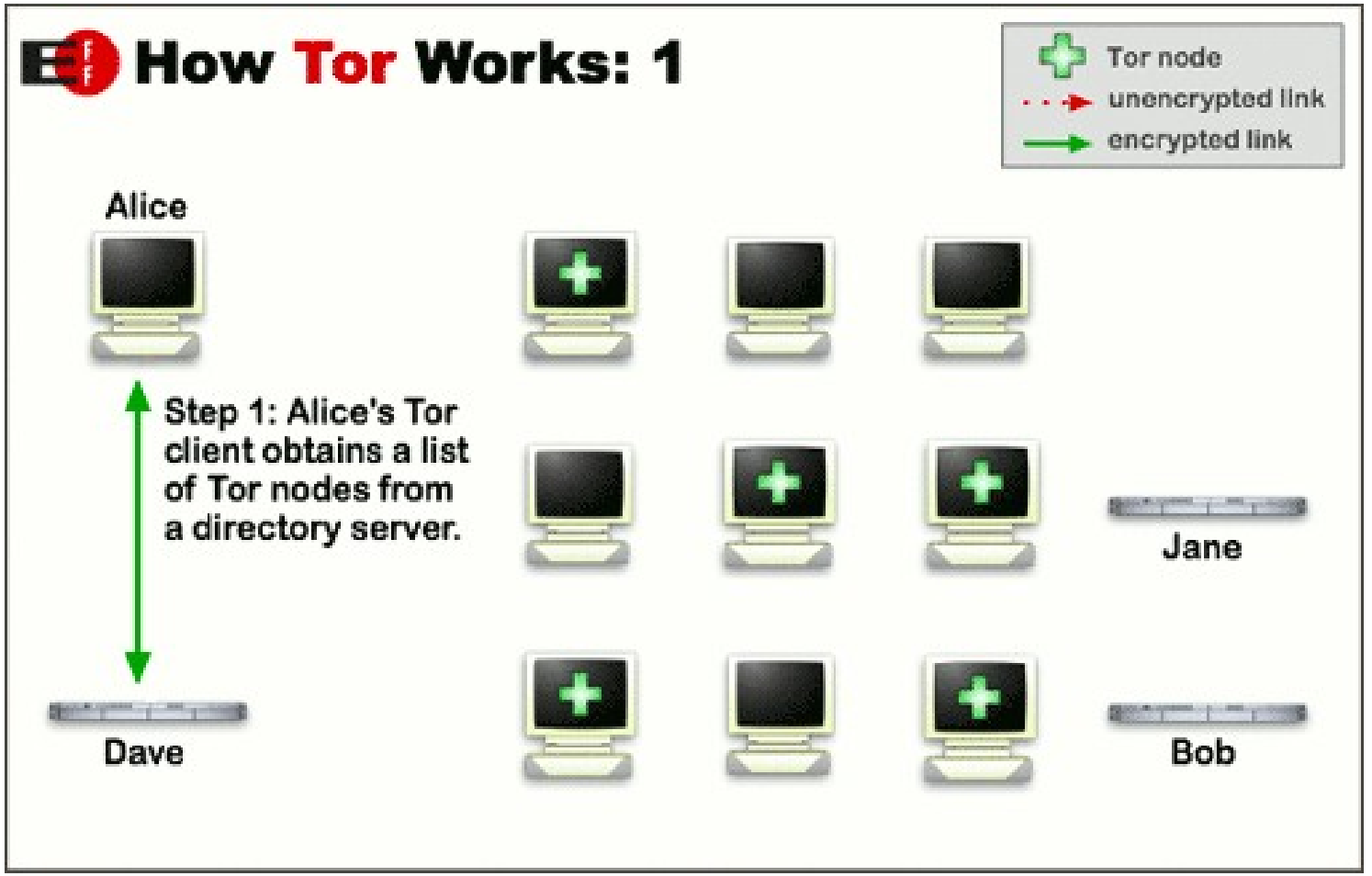
Потребителите на Tor се свързват чрез виртуални тунели, а не директно. Заобикаляйки цензура, достигайки иначе блокирани сайтове и съдържание.

Разработчиците на софтуер могат да използват Tor за създаване на нови комуникационни средства, включващи анонимност.

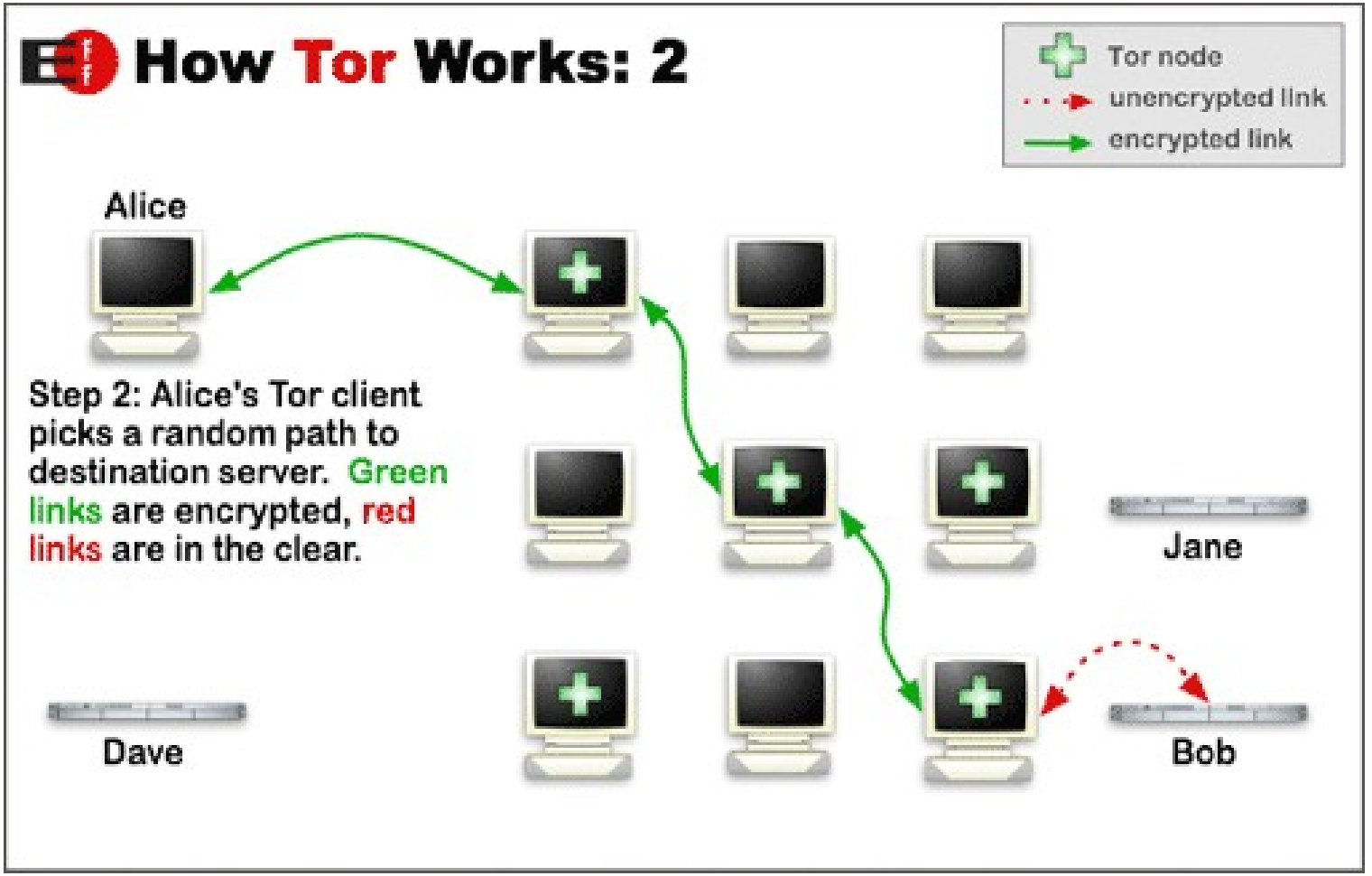
За да браузвате анонимно, трябва да си инсталирате **Tor Browser**:

https://www.torproject.org/projects/torbrowser.html.en

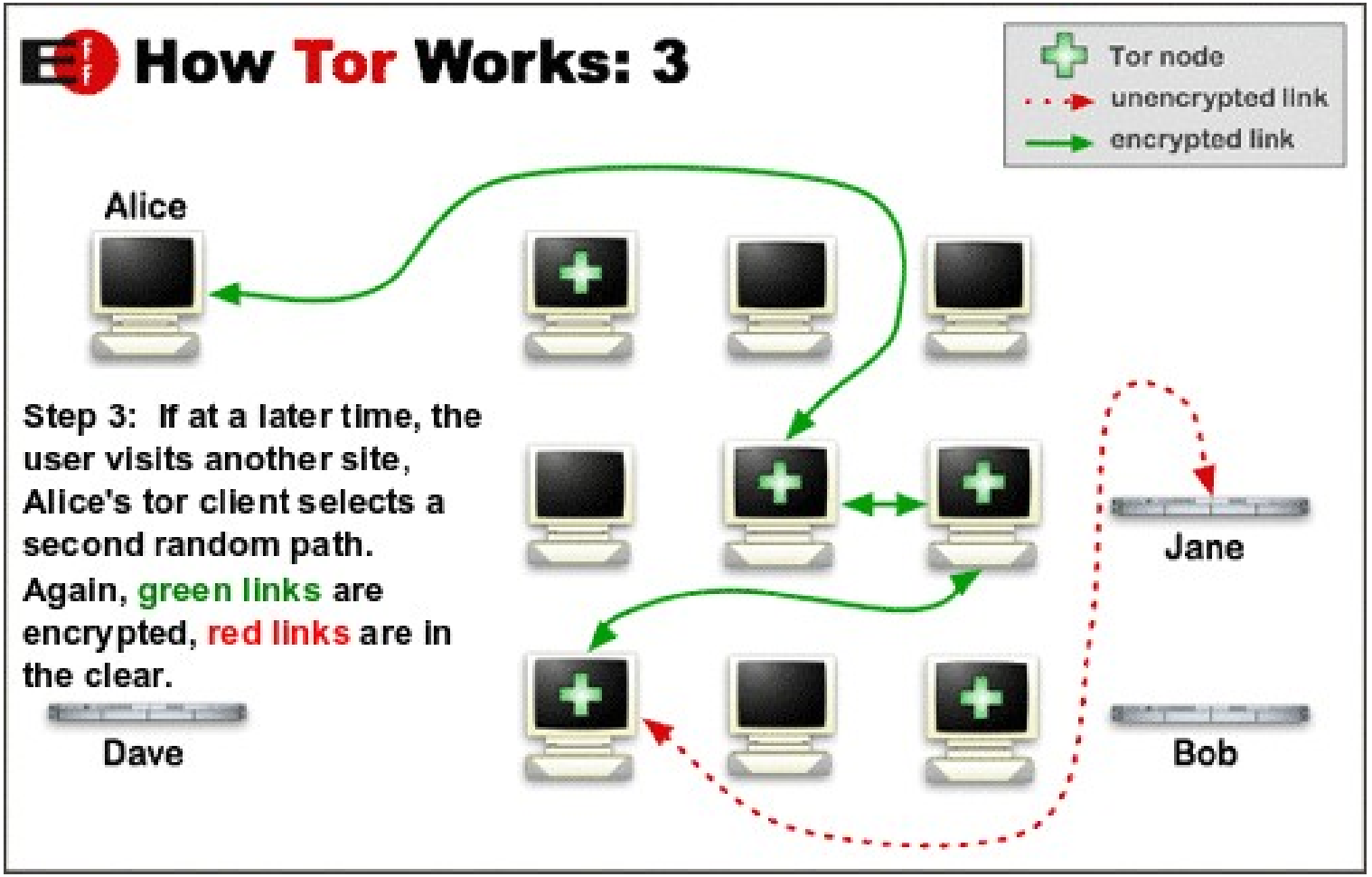
Как работи Tor (1)



Как работи Tor (2)



Как работи Tor (3)



Версия 1.0 на HTTP (RFC 1945)

При първите версии на HTTP протокола за изпълнението на всеки метод се прави отделна HTTP сесия.

Тя отваря TCP съединение, праща нещо, след това затваря последователно съединението и сесията.

Имаме една сесия, едно съединение.

Не е ефективно

Много служебен трафик.

Ако за всяко GET се затваря съединението, а предстои четене на серия от страници.

Правенето на всяко ново съединение:

- resolving на URL-адреса;
- на TCP ниво 3-way handshake;
- договаряне на HTTP сесия.

Колко време да е отворена сесията

Не може всяко обръщение към дадена страница да отваря сесия към даден сървър и да я държи

$$\Delta t \approx \infty$$

Нормално би било сесията да приключи с изтеглянето на страницата.

Версия 1.1 на HTTP (RFC 7230-35)

При версия 1.1 на HTTP тези недостатъци са преодолени (до някъде):

- на едно TCP съединение отговарят няколко сесии (няколко команди).
- различни хипертекстови страници, но с едно TCP съединение.

Създава се „канал“ в пълен дуплекс: в едната посока - заявки, а в обратната – отговори.

HTTP сесия

HTTP сесията е последователност от request-response транзакции.

HTTP клиентът инициира заявка. Установява съединение на порт 80/TCP.

HTTP сървърът “слуша” на порт 80 за съобщение със заявка от клиент.

При получаване на заявка сървърът връща "HTTP/1.1 200 OK" и съобщение, което съдържа:

- търсения ресурс или
- съобщение за грешка.

Съобщение **Request**

request съобщението се състои:

* ред Request, напр. GET /images/logo.png HTTP/1.1, т.е заявява от сървъра ресурса /images/logo.png

* хедъри, напр. Accept-Language: en

* празен ред, който отделя блока с данни от хедъра

* тяло на съобщението (евентуално)

Редовете да завършват с <CR><LF>

Примери на **Request-и:** получаване на главна страница

http://developer.mozilla.org/ на английски език:

```
GET / HTTP/1.1
Host: developer.mozilla.org
Accept-Language: en
```

Следва пример на изпращане на резултат от попълнен формуляр:

Request-и: изпращане на попълнен формуляр

```
POST /contact_form.php HTTP/1.1
Host: developer.mozilla.org
Content-Length: 64
Content-Type: application/x-www-form-urlencoded

name=Joe%20User&request=Send%20me%20one%20of%20your%20catalogue
```

HTTP Request методи

HTTP дефинира девет Request метода (наричани и "verbs" - глаголи), които показват какво действие да се изпълни върху желания ресурс.

Ресурсът е файл или данни, получени от изхода на програма на сървъра.

Основен метод **GET**

Основният метод е **GET**.

Като аргумент му се подава адреса на страницата върху диска на сървъра.

Страниците могат да се кешират върху proxy-сървъри, затова в метода GET има if-условие.

Метод **HEAD**

Всяка хипертекстова страница има заглавие:

- възрастта на страницата;
- къде се намира;
- датата на последната модификация и др.

Методът **HEAD**, за разлика от GET, взима само заглавието на страницата.

Полезен е за извличане на мета-информация.

Методи PUT и POST

Методът **PUT** служи за прехвърляне на страница върху сървъра:

- обмен на два етапа – първо се дава адреса на страницата, след това се прехърля самата страница.

Методът **POST** ≈ PUT, но той добавя новите данни към съществуващ адрес:

- създава се нов ресурс или се обновява съществуващ.

Методът TRACE

Методът **TRACE** връща от сървъра получените данни по заявката.

Той се използва за тестване – да видим дали сървърът е получил това, което сме изпратили.

Методът **DELETE** изтрива заявения ресурс.

OPTIONS проверява функционалността (методите) на web сървъра за дадено URL:

- иска **'*'** вместо конкретен ресурс.

Методи CONNECT и PATCH

CONNECT конвертира request съединението в прозрачен TCP/IP тунел, за да минат SSL-криптирани комуникации (HTTPS) през некриптирано HTTP прокси.

PATCH – за частични модификации върху ресурс.

HTTP трябва да реализират **поне GET и HEAD**, когато е възможно - и OPTIONS.

Структура на отговора на сървъра (**server response**)

Отговорът (response) на заявката от страна на web сървъра също се формира от текстови директиви, разделени с <CR><LF> и е от три блока:

1. Първият ред, status line, е потвърждение на използваната HTTP версия и кода за състояние на изпълнението на заявката.

2. Следват HTTP хедъри, информиращи клиента какви данни ще бъдат изпратени (тип, обем, алгоритъм за компресиране, кеширане). Както беше при client request-а, този блок завършва с празен ред.

3. Последният блок е блокът с данни.

Пример на response

```
HTTP/1.1 200 OK
Content-Type: text/html; charset=utf-8
Content-Length: 55743
Connection: keep-alive
Cache-Control: s-maxage=300, public, max-age=0
Content-Language: en-US
Date: Thu, 06 Dec 2018 17:37:18 GMT
ETag: "2e77ad1dc6ab0b53a2996dfd4653c1c3"
Server: meinheld/0.6.1
Strict-Transport-Security: max-age=63072000
X-Content-Type-Options: nosniff
X-Frame-Options: DENY
X-XSS-Protection: 1; mode=block
Vary: Accept-Encoding,Cookie
Age: 7
<empty line>
```

Пример на response (data block)

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="utf-8">
  <title>A simple webpage</title>
</head>
<body>
  <h1>Simple HTML5 webpage</h1>
  <p>Hello, world!</p>
</body>
</html>
```

Кодове за състояние на изпълнението (Response status codes)

Тези кодове се групират в пет класа: информиращи, успешно изпълнение, пренасочване, клиентски и сървърски грешки.

200: OK. Заявката е успешна.

301: Moved Permanently. URI (URL) на заявения ресурс се е променил.

404: Not Found. Сървърът не може да открие заявения ресурс.

HTTP Secure

Hypertext Transfer Protocol Secure (**HTTPS**) е:

HTTP с SSL/TLS

за криптирани комуникации и сигурна идентификация на web сървър.

HTTPS се използва за е-разплащания и други поверителни операции

(напр. СУСИ: https://susi4.uni-sofia.bg/)

S-HTTP

HTTPS да не се бърка със Secure HTTP (S-HTTP) - RFC 2660.

(S-HTTP криптира само данните в страницата, POST полетата. Не засяга инициране на съединението. S-HTTP може да споделя един порт с HTTP. Некриптираният хедър определя дали съдържанието е криптирано или не.)

HTTPS сесия

Можем да се доверим на HTTPS съединение, ако са изпълнени всички изброени по-долу условия:

1. Потребителят е сигурен, че неговият браузър правилно имплементира HTTPS с правилно инсталирани предварително Certificate Authorities (CAs).

(CAs са организации, които валидират идентичността на обекти – web сайтове, email адреси, юридически и физически лица, като им издават цифрови сертификати.)

2. Потребителят се доверява на CAs, гарантиращи легитимността на съответни web сайтове.

HTTPS Сесия

3. web сайтът има валиден сертификат, т.е подписан е от доверена трета страна.

4. Сертификатът коректно идентифицира web сайта.

5. Или междинните хопове по Internet са доверени, или потребителят се доверява на криптиращия слой (TLS или SSL), че не е податлив на “прослушване” или “разбиване”.

HTTP Strict Transport Security

HTTP Strict Transport Security (HSTS) предпазва от HTTPS downgrade атаки, като кешира HTTPS политиката на домейна локално на браузъра.

HSTS хедърът информира браузъра, че всички опити за достъп до даден сайт трябва да бъдат единствено и само по HTTPS.

HSTS хедър. Пример.

(https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Strict-Transport-Security)

```
Header type   Response header

Strict-Transport-Security: max-age=<expire-time>

Strict-Transport-Security: max-age=<expire-time>;
includeSubDomains

Strict-Transport-Security: max-age=<expire-time>;
preload
```

max-age=<expire-time> време в секунди, през което браузърът да помни, че сайтът трябва да се достъпва само по HTTPS.

includeSubDomains (Optional) Това правило се прилага и до всички поддомейни.

Preload (Optional) ; (вж. https://hstspreload.org/)

DNS over HTTPS (DoH)

DoH (RFC 8484) е протокол за DNS резолвинг по HTTPS.

Целта е да се повиши сигурността, HTTPS създава криптирана сесия върху TCP.

От март, 2018 г. Google и Mozilla Foundation тестват DoH.

Версията на Google прилага командите HTTP GET (върху HTTPS), за да достъпи DNS информация, а резултатът е представен с JSON нотация.

DoH – RFC 8484

Според утвърденото RFC това е протокол за изпращане на DNS заявки и получаване на DNS отговори върху HTTPS (т.е гарантиране на сигурността на трафика чрез TLS).

Поддържа "wire format", DNS отговорите са като съществуващите UDP отговори, но те са в полето за данни на HTTPS - application/dns-udpwireformat MIME тип.

Ако се прилага HTTP/2, сървърът може да приложи server push.

HTTP сървър Apache (httpd.apache.org)

В момента се използват два HTTP сървъра:

apache - в UNIX/Linux и

IIS на Microsoft.

Apache HTTP Server Project е open-source решение за UNIX/Linux, може и за Windows.

Apache **httpd** е най-популярният web сървър в Internet от Април 1996.

Apache “слуша” на порт 80

Binding (обвързване) с IP адреси и портове:

След стартиране httpd се обвързва (слуша) с порт и IP адрес на локалната машина и чака заявки.

По подразбиране слуша на всички адреси и на порт 80. Но може да му се зададат определени IP адреси и портове.

Директивата **Listen**

Директивата **Listen** казва на сървъра да приема входящи заявки само на определен(и) порт(ове) или комбинации адрес-порт:

- ако е зададен само port No., сървърът слуша на дадения порт и на всички интерфейси;

- ако е зададен IP и portNo., сървърът слуша на дадения порт и интерфейс.

Файлът httpd.conf може да съдържа >1 директива Listen. Сървърът ще отговаря на заявки от всеки от зададените адреси и портове.

Директивата Listen. Примери.

Искаме сървърът да приема конекции и на порт 80, и на порт 8000, на всички интерфейси:

```
Listen 80

Listen 8000
```

Сървърът да приема конекции на порт 80 за един интерфейс, и порт 8000 за друг:

```
Listen 192.0.2.1:80

Listen 192.0.2.5:8000
```

Слуша по **IPv6**

IPv6 адресите трябва да бъдат заградени в квадратни скоби:

```
Listen [2001:db8::a00:20ff:fea7:ccea]:80
```

Определяне на протокол с Listen

По подразбиране https “слуша” на порт 443.

Протокол се дефинира само ако работи на нестандартни портове. Напр., https се пренасочва към порт 8443:

```
Listen 192.170.2.1:8443 https
```

HTTP/2: по-висока производителност

През май, 2015 г. е публикувана **RFC 7540**.

Главните промени спрямо HTTP/1.1:

- Основната протоколна единица е кадъра (**frame**), който служи за различни цели.

- Мултиплексиране, компресия на хедъра, приоритетизация и “договаряне” (negotiation) на протокола.

Развитие на нестандартизиран протокол SPDY (speedy).

Chrome поддържа SPDY до версия 6, но вече поддържа HTTP/2.

HTTP/2: множество “разговори” в една TCP сесия

Мултиплексирането на заявките се постига, като всяка HTTP request/response сесия е обвързана със свой поток.

Потоците са независими един от друг и могат да се приоритетизират.

Така, в сравнение с HTTP/1.x, тук се установяват по-малко TCP сесии – по-добро оползотворяване на капацитета на мрежата.

HTTP/2: flow control, server push...

Управлението на потока (flow control) гарантира, че от източника на заявка (отговор) се подават само данни, които могат да бъдат използвани от приемника.

HTTP/2 въвежда нов режим на взаимодействие на сървъра с клиента: server push.

Сървърът изпраща само тези данни към клиента, които преценява, че са му необходими.

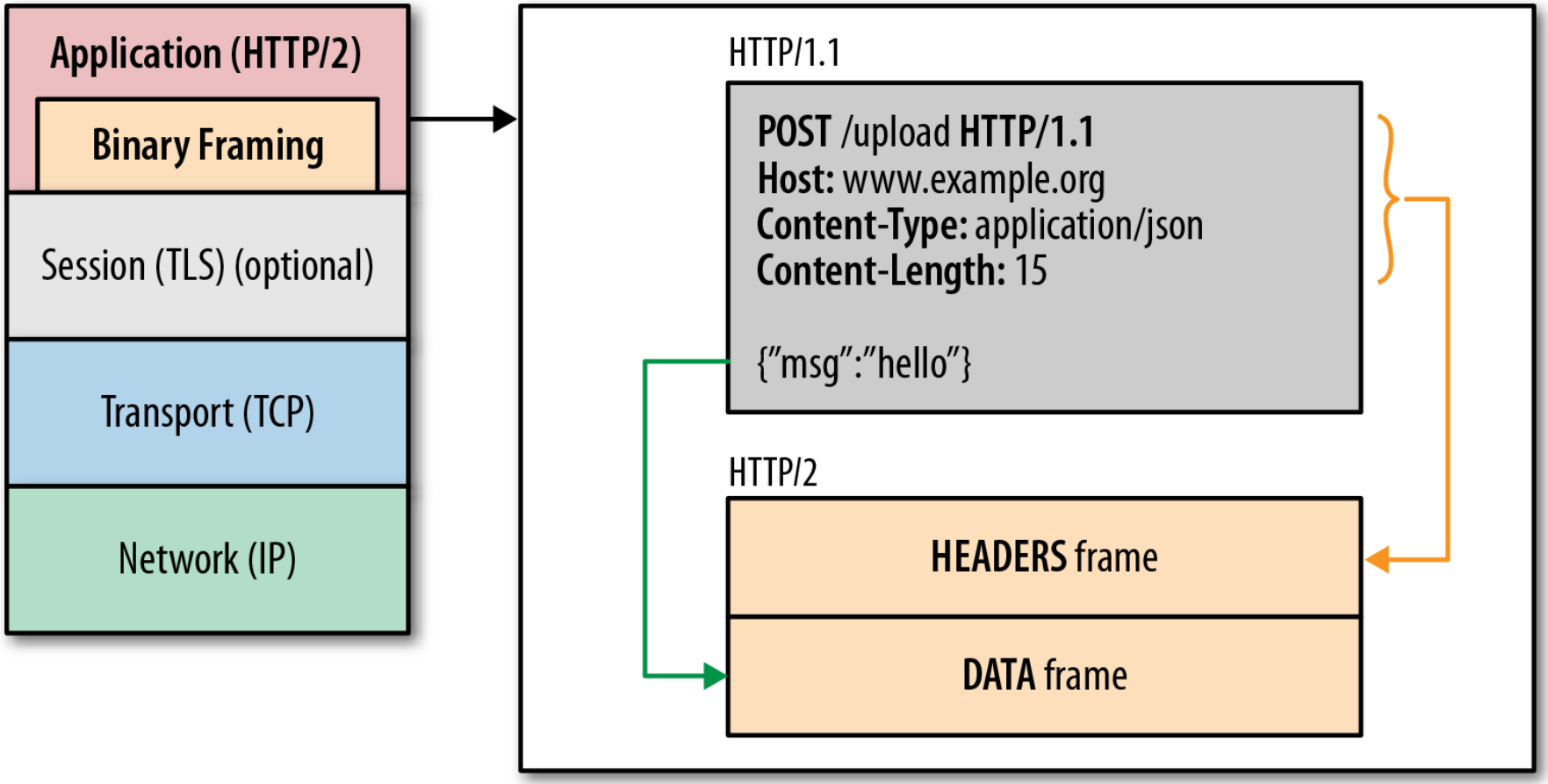
...server push и двоични формати

Сървърът синтезира заявка под формата на PUSH_PROMISE фрейм и след това връща отговор на тази заявка в отделен поток.

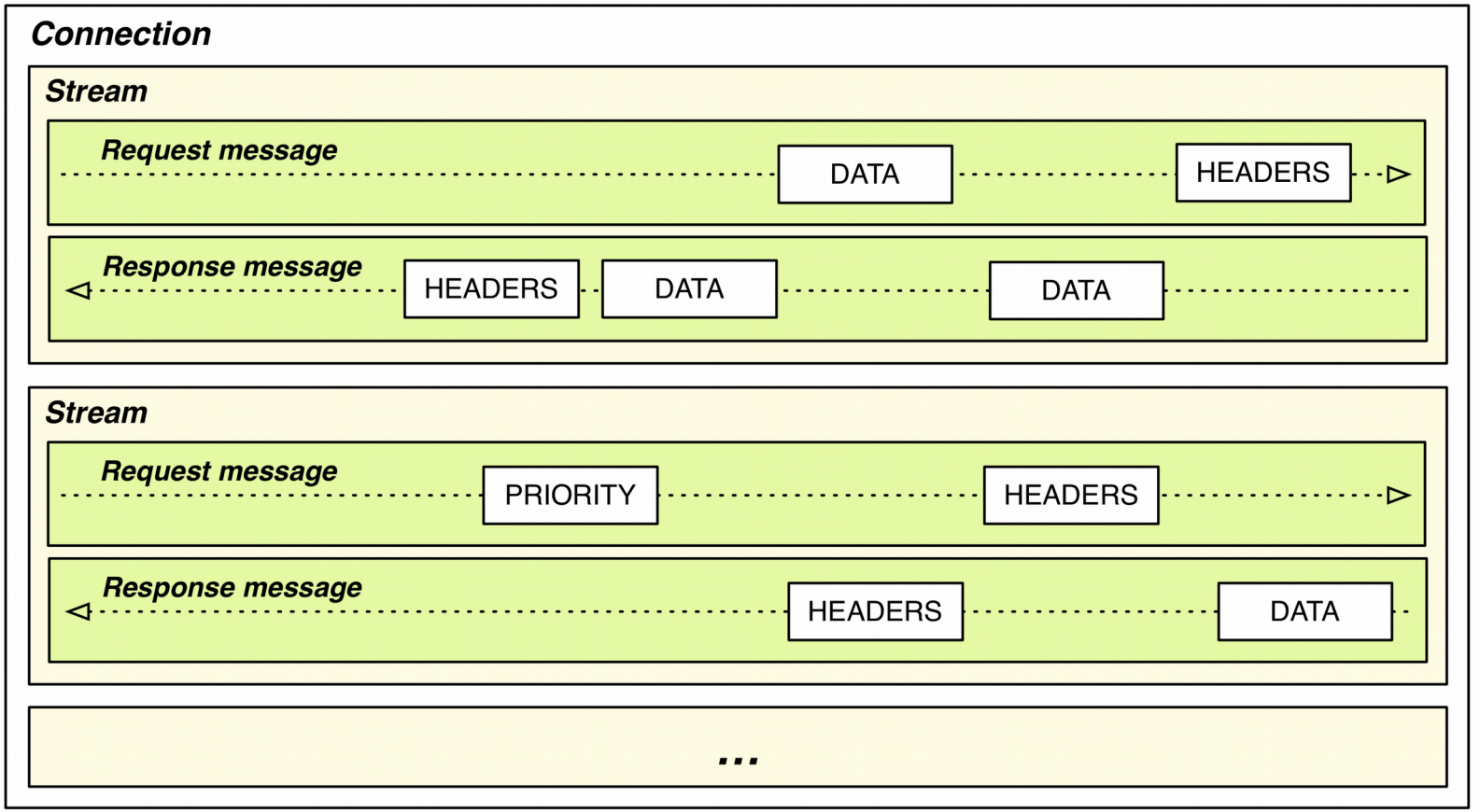
Така (евентуално) се вдига производителността на сесията клиент-сървър.

HTTP/2 използва двоични формати на фреймовете. Така по-ефективно се обработват съобщенията.

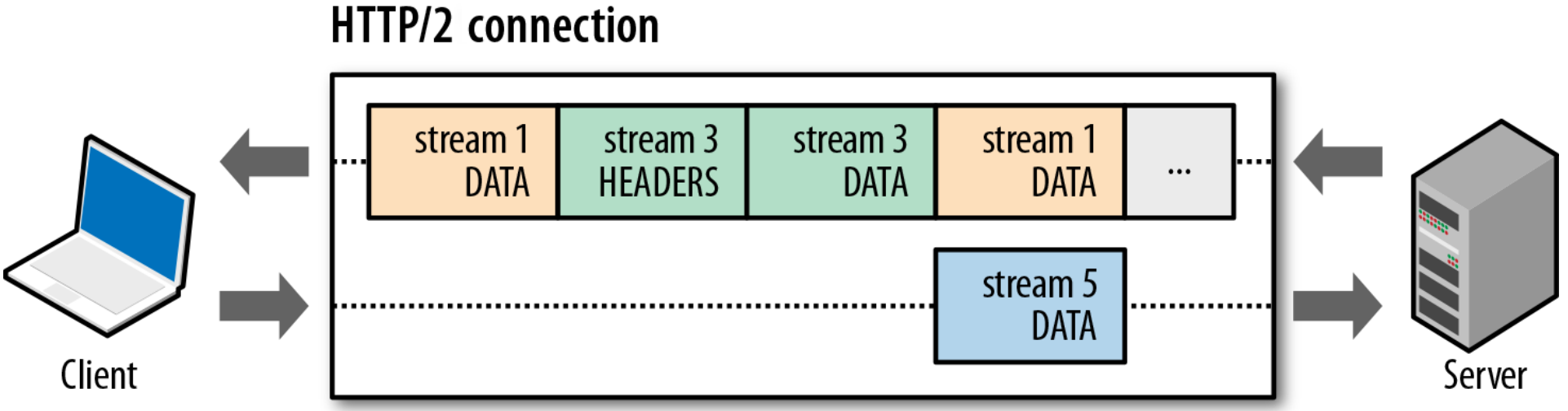
HTTP/2: Binary Framing Layer



HTTP/2: потоци, съобщения и фреймове

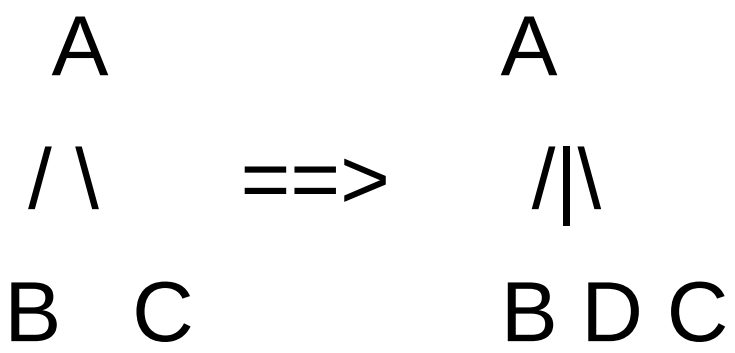


HTTP/2: Три потока в една сесия



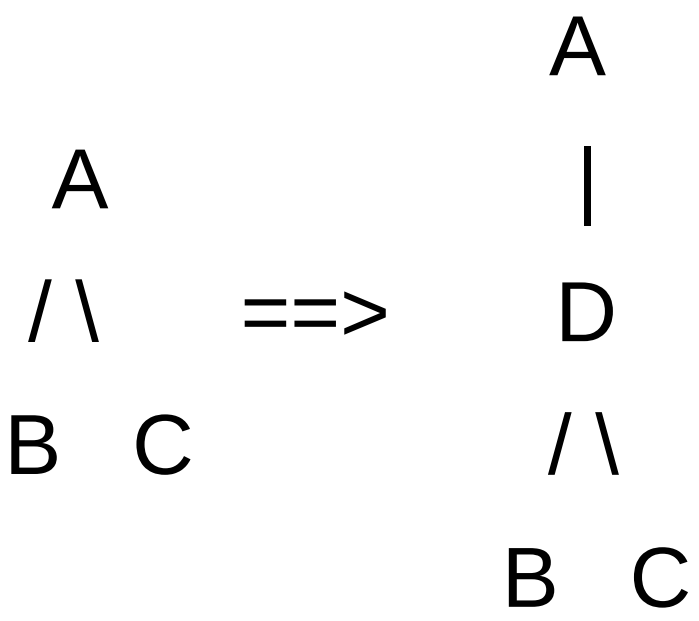
HTTP/2: създаване на потоци, зависимости - Default Dependency

Потоци B и C са зависими от A. Ако се създаде поток D със зависимост от A:



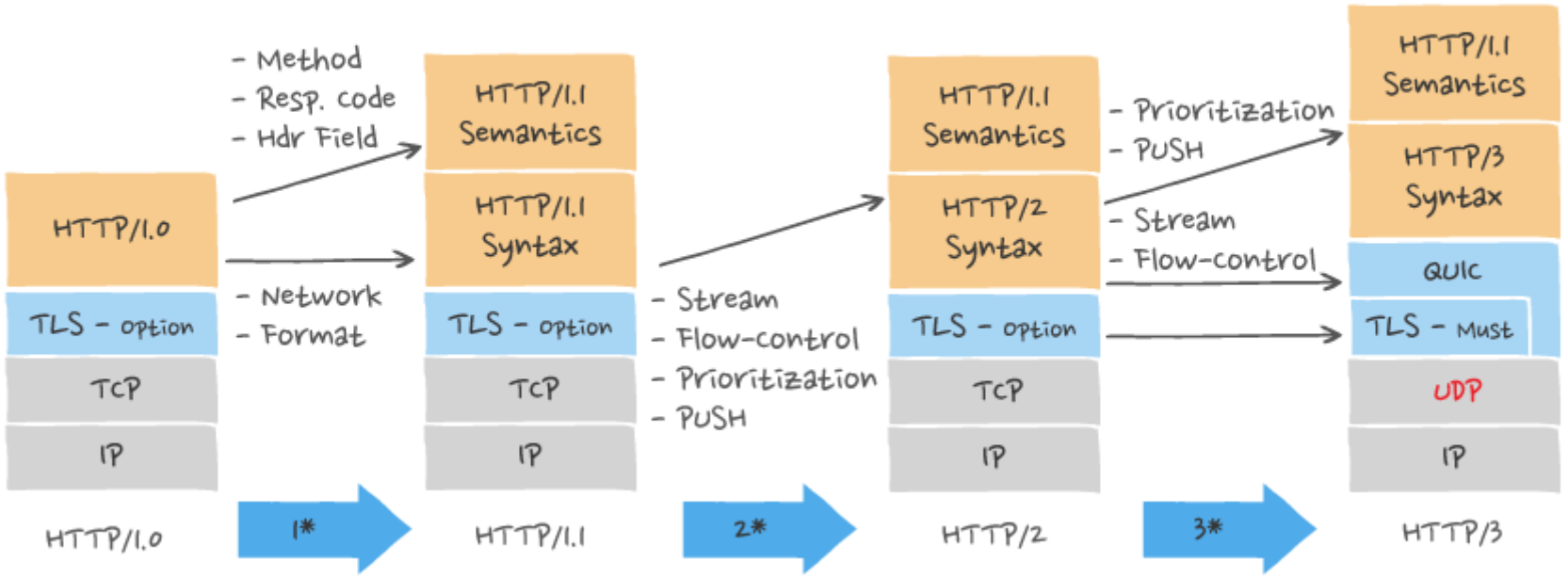
...Exclusive Dependency

Ако обаче поток D се създаде с изключителна зависимост от A:



HTTP/3: QUIC върху UDP

HTTP protocol stack transition and comparison



HTTP protocol stack transition and comparison

Защо от HTTP/2 към /3

HTTP/3 трябва да осигури бърза, надеждна и сигурна свързаност в web пространството за всякакъв вид устройства.

Използва специален транспортен протокол **QUIC**, работещ върху **UDP**.

Създаден е от Google под името gQUIC, но бързо е възприет от Интернет общността като QUIC (Quick UDP Internet Connections).

QUIC е подобен на TCP+TLS+HTTP2, но е самостоятелен протокол, реализиран върху UDP. Това позволява лесно да се правят иновации, без да се засягат съществуващи клиентски и middleware софтуери.

HTTP/3 преодолява TCP забавянията

UDP няма строго подредената схема на обмен на съобщения за установяване на сесия в TCP.

Това помага и за преодоляване на проблема с фронталното блокиране (head-of-line blocking - **HoL**) на пакетно ниво.

HOL блокирането в HTTP/1.1 се появява, когато позволеният брой паралелни заявки в браузъра се изчерпва.

HTTP/2 решава този проблем на приложно ниво чрез мултиплексирането на заявките, но HOL остава на транспортно ниво - TCP.

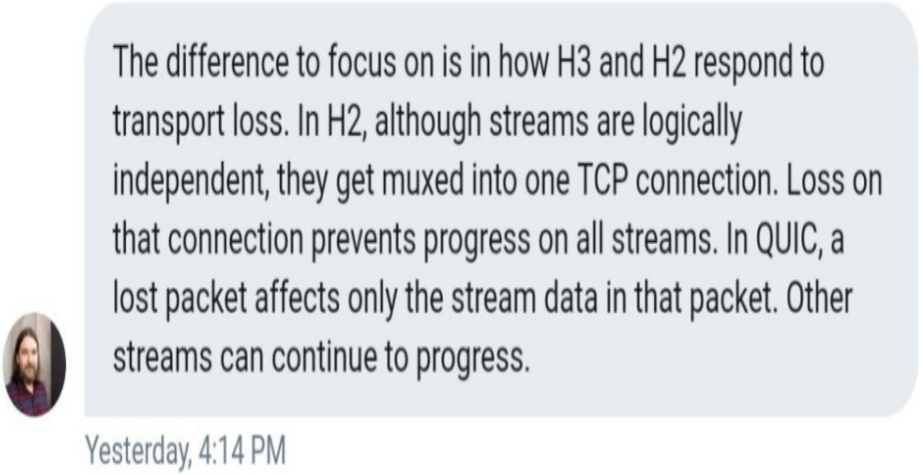
QUIC: мултиплексиране, Flow Control и приложение

QUIC въвежда мултиплексирането на множество потоци върху една единствена връзка.

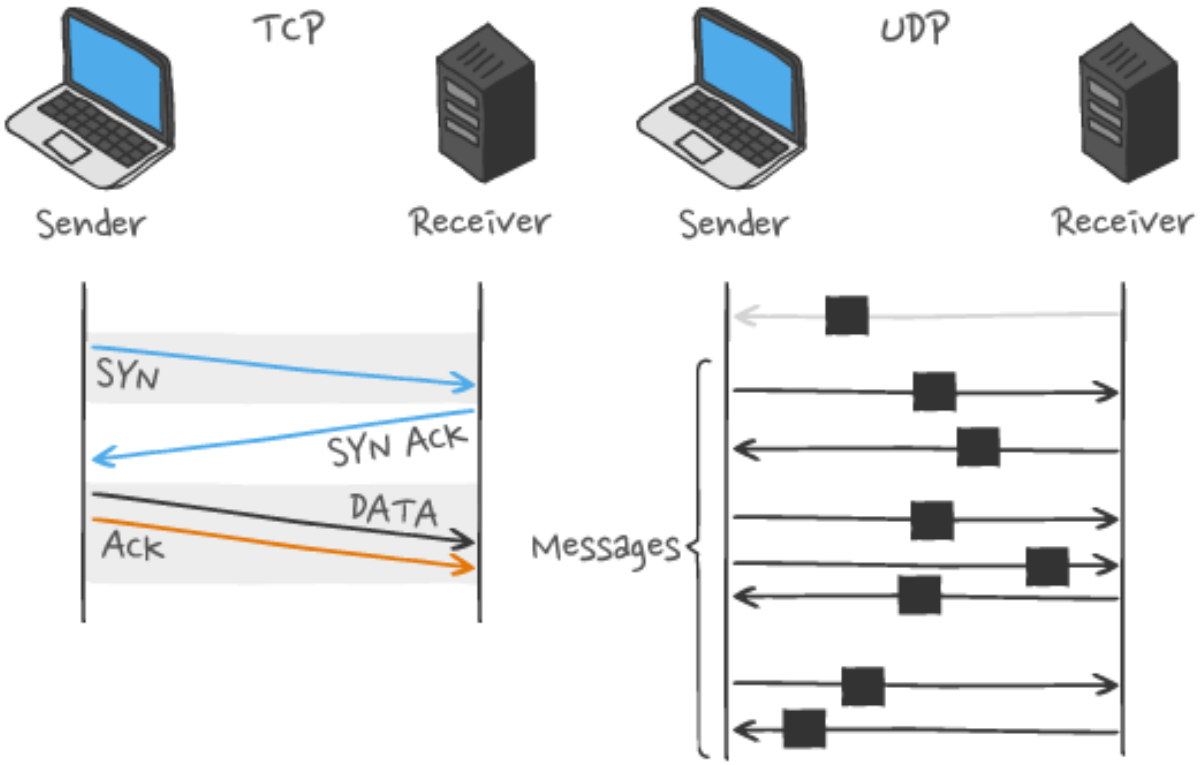
Flow Control за всеки отделен поток от данни, решавайки по този начин HoL проблема за цялата връзка.

HTTP/3 решава проблемите с влошаване на качеството на уеб браузването заради HOL блокирането в безжични среди, ако покритието не е добро.

HTTP/3 ще намери приложение в IoT, full-mesh микроуслуги, VR и др.



TCP vs. UDP Overhead



QUIC елиминара забавянията за установяване на сесии в TCP: