Лекция 4

Как работает Интернет?

Технология облачных вычислений

Что для нас есть Интернет?

Что для нас есть Интернет?

- Интернет – это абстрактное место размещения ресурсов, для доступа к которым необходимо знать:
  1) IP-адрес
  2) Порт
  3) Протокол
  4) Наименование ресурса

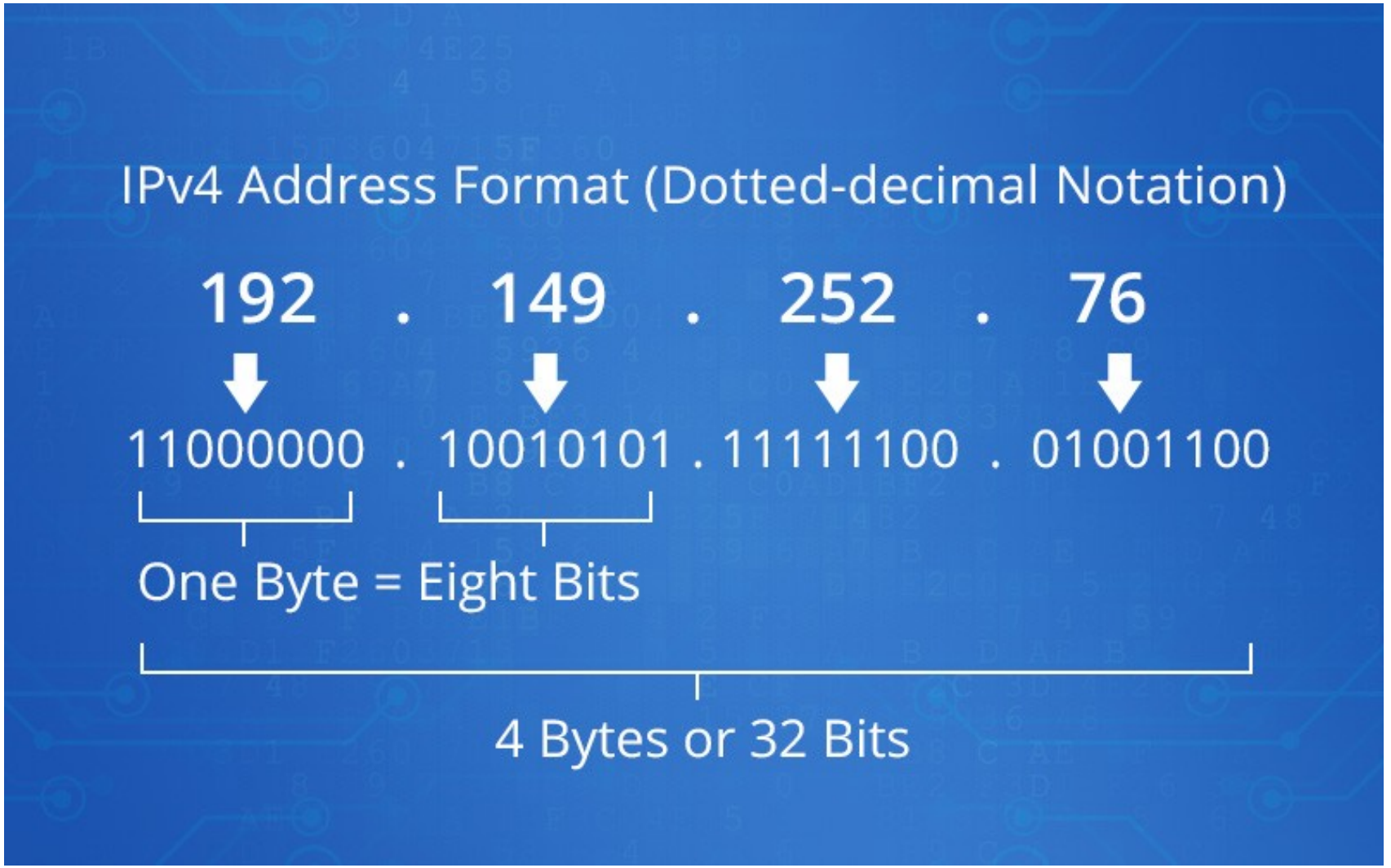
IP протокол

- 1974г. V. Cerf and R. Kahn, "A Protocol for Packet Network Intercommunication," in IEEE Transactions on Communications, vol. 22, no. 5, pp. 637-648, May 1974, doi: 10.1109/TCOM.1974.1092259.
- Декабрь 1974г. RFC 675 – TCP протокол
- **IPv0** Март 1977г. IEN 5 – TCPv2 (RFC 775)
- **IPv1** Январь 1978г. IEN 21 – TCPv3 (RFC 775)
- **IPv2** Февраль 1978г. IEN 28 – IP протокол draft
- **IPv3** Февраль 1978г. IEN 27 TCP Version 3.1
- **IPv4** Июнь 1978г. IEN 40, IEN 41 – **IP, TCPv4**
- **IPv5** Сентябрь 1979г. – 1995г. “Internet Stream Protocol” IEN 119, RFC 1190, 1819.
- **IPv6** Декабрь 1995г. RFC 1883, 2460, 8200 – новый IP протокол
- **IPv7, IPv8, IPv9** – RFC 1475, 1707, 1621, 1622, 1347

IPv4 протокол

- IPv4 идентифицирует устройства, подключенные к сети через назначается уникальный, числовой IP-адрес, например 192.149.252.76.
- IPv4 использует 32-битную адресную схему, позволяя различать 2^32 адреса (4,19 млрд. Адресов).
- Минимальный размер равен 20 байтам (заголовок без данных), максимальный — 65535 байт (16-бит на размер пакета в байтах). Пакеты большего размера, чем поддерживает канал связи, фрагментируются.
- Время жизни» (TTL) пакета определяет максимальное количество маршрутизаторов на пути следования пакета. Каждый маршрутизатор при обработке пакета должен уменьшить значение TTL на единицу.
- В феврале 2011 года IANA выделила 5 последних блоков адресов для RIR. Блоки свободных IP-адресов начали заканчиваться у региональных регистраторов с 2011 года
- 25 ноября 2019 года были распределены последние свободные IPv4-адреса в Европе и на Ближнем Востоке. Теперь получить IPv4-адрес можно будет, только если его освободит текущий владелец

IPv4 адрес



IPv4-адрес:

подсеть

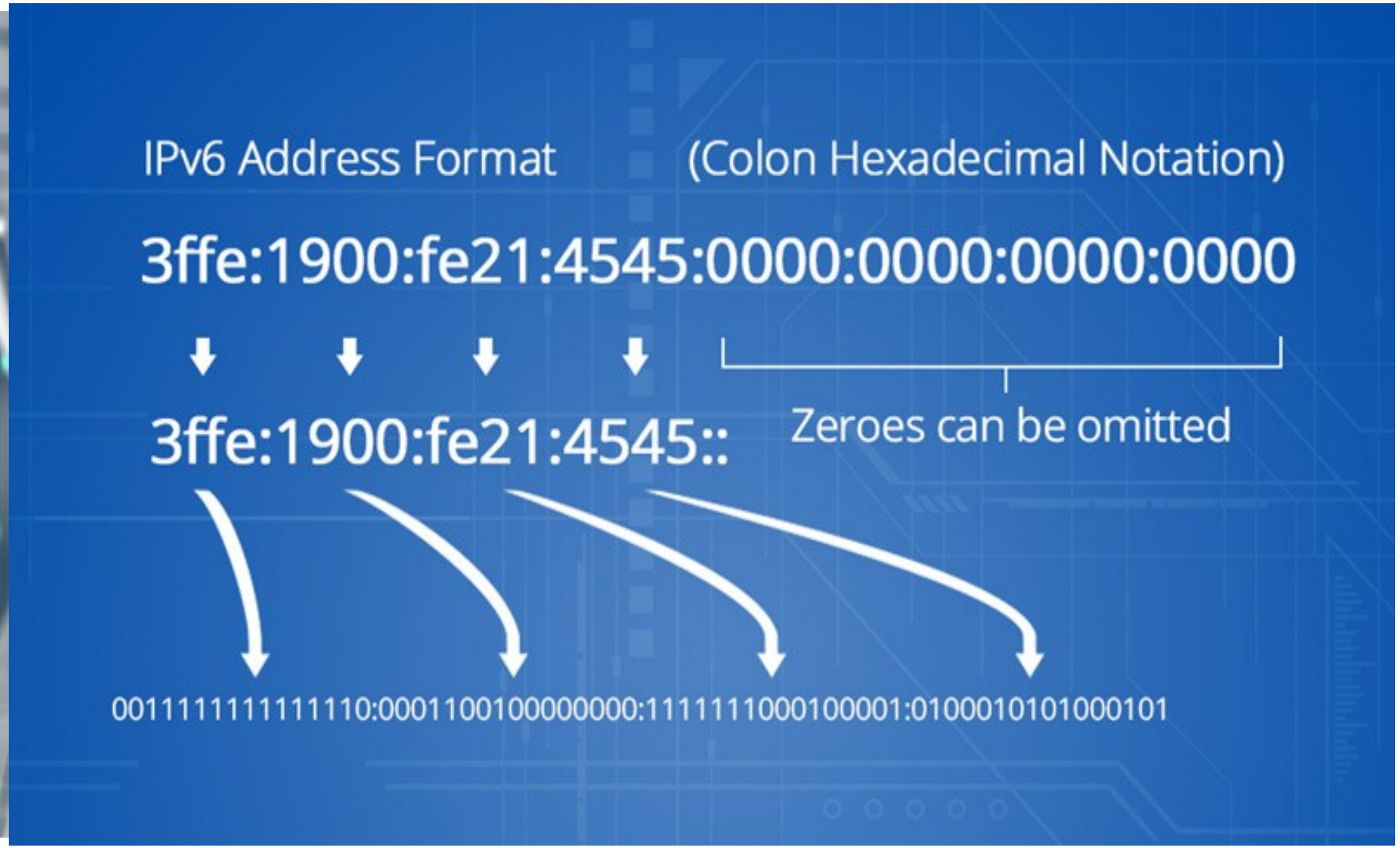
(префикс)

+

ID

устройства

IPv6 адрес



IPv6-адрес:

префикс

+

подсеть

+

64bit ID устройства

IPv4 vs IPv6

IPv4 Header

Version	IHL	Type of Service	Total Length
Identification		Flags	Fragment Offset
Time to Live		Protocol	Header Checksum
Source Address			
Destination Address			
Options			Padding

Legend

- Field's Name Kept from IPv4 to IPv6
- Fields Not Kept in IPv6
- Name and Position Changed in IPv6
- New Field in IPv6

IPv6 Header

Version	Traffic Class	Flow Label	
Payload Length		Next Header	Hop Limit
Source Address			
Destination Address			

Где порт?

Интернет протоколы

UDP

TCP/IP

BGP

FTP

DNS

HTTP

HTTPS

NTP

RADIUS

RTP

SSH

SMTP

SNMP

DHCP

Интернет протоколы

UDP — User Datagram Protocol

TCP/IP — Transmission Control Protocol/Internet Protocol

BGP — Border Gateway Protocol

FTP — File Transfer Protocol

DNS — система доменных имён

HTTP — HyperText Transfer Protocol

HTTPS — Hypertext Transfer Protocol Secure

NTP — Network Time Protocol

RADIUS — протокол аутентификации, авторизации и работы с аккаунтами

RTP — Real-time Transport Protocol

SSH — Secure Shell

SMTP — Simple Mail Transfer Protocol

SNMP — Simple Network Management Protocol

DHCP — Dynamic Host Configuration Protocol

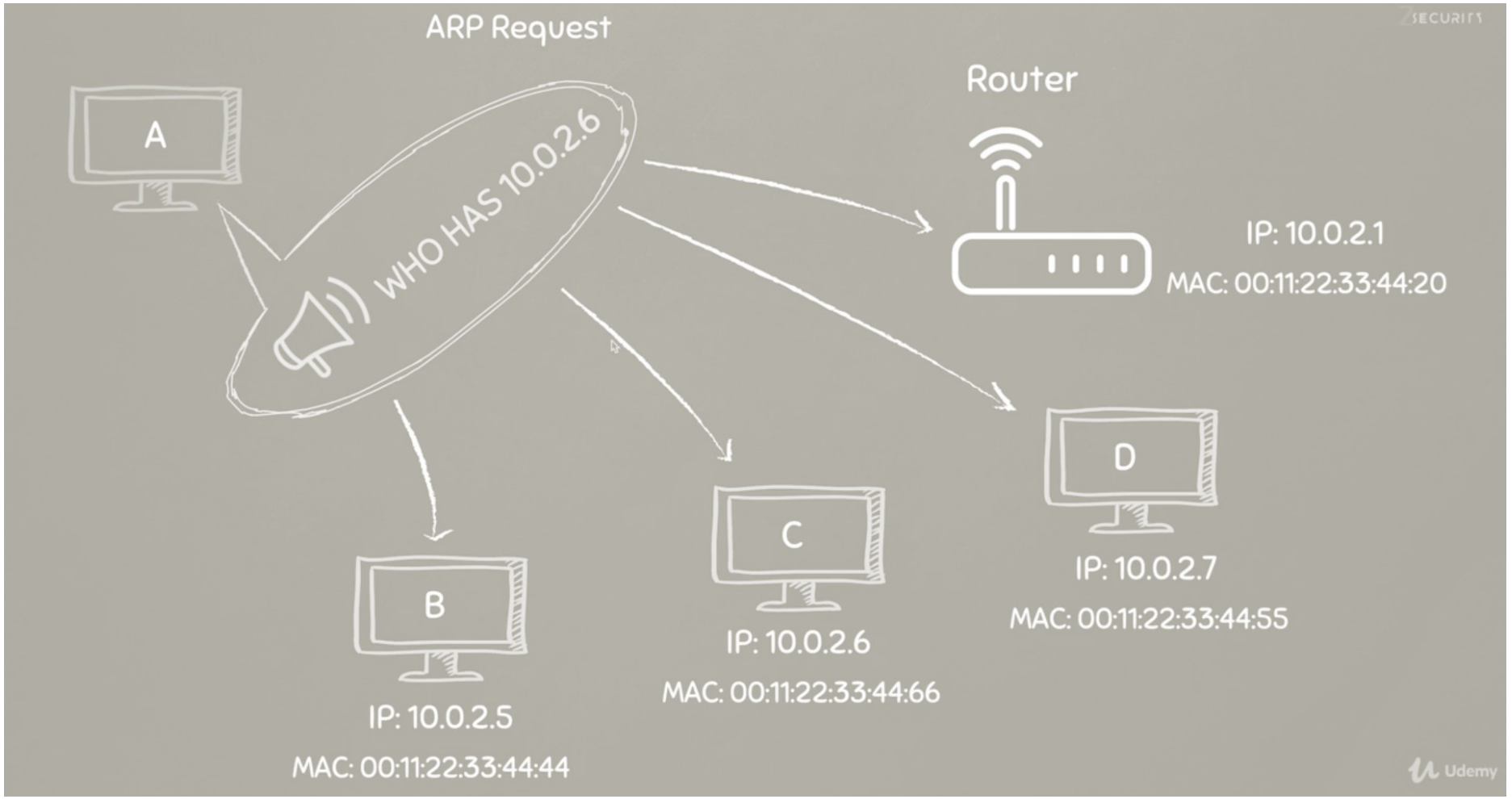
Порт

Порт	Описание	Протокол
20	FTP-DATA — для передачи данных FTP	TCP
21	FTP — для передачи команд FTP	TCP
22	SSH (Secure SHell) — сетевой протокол безопасной передачи данных	TCP,UDP
25	SMTP (Simple Mail Transfer Protocol) — пересылка почты	TCP,UDP
53	DOMAIN (Domain Name System, DNS)	TCP,UDP
67	BOOTPC (Bootstrap Protocol Client), DHCP — для серверов	UDP
68	BOOTPC , DHCP (Dynamic Host Configuration Protocol) — для клиентов	UDP
80	HTTP (HyperText Transfer Protocol); ранее — WWW	TCP,UDP
115	SFTP (Simple File Transfer Protocol[en])	TCP,UDP
123	NTP (Network Time Protocol) — для синхронизации времени	TCP,UDP
161	SNMP (Simple Network Management Protocol) — удалённый мониторинг	TCP,UDP
162	SNMPTRAP (Simple Network Management Protocol Trap)	TCP,UDP
179	BGP (Border Gateway Protocol)	TCP,UDP
443	HTTPS (HyperText Transfer Protocol Secure) — HTTP с шифрованием	TCP,UDP

ARP – address resolution protocol

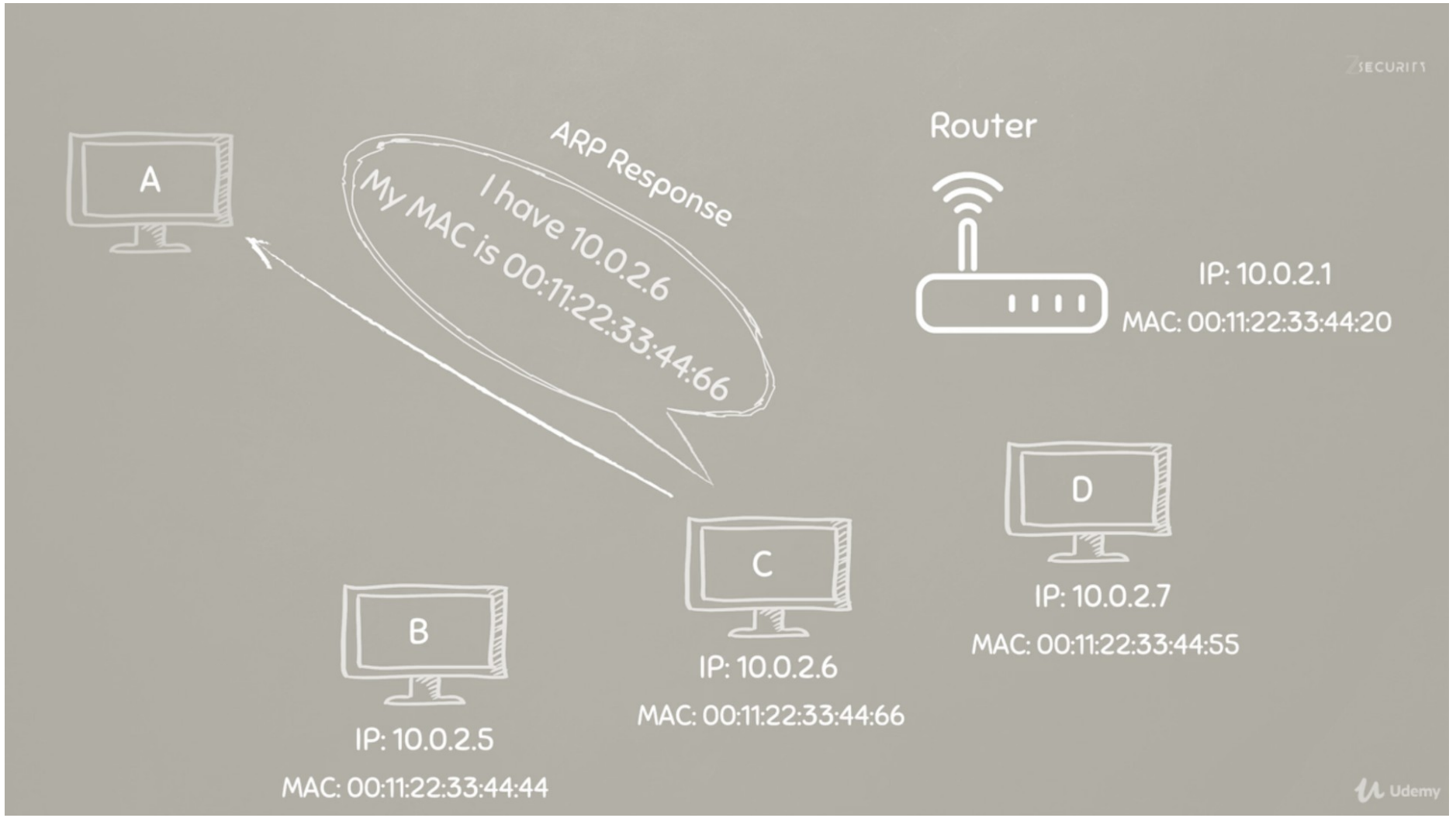
ARP – address resolution protocol

Широковещательный запрос – есть кто нибудь с таким адресом?



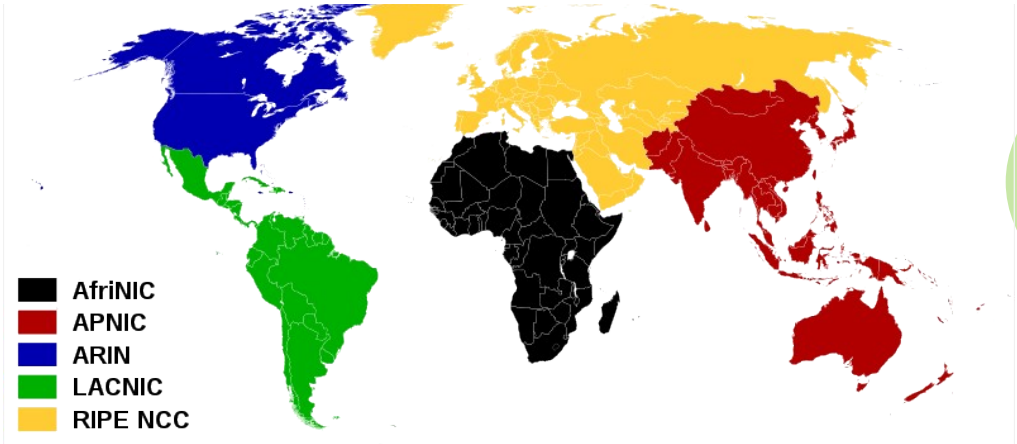
ARP – address resolution protocol

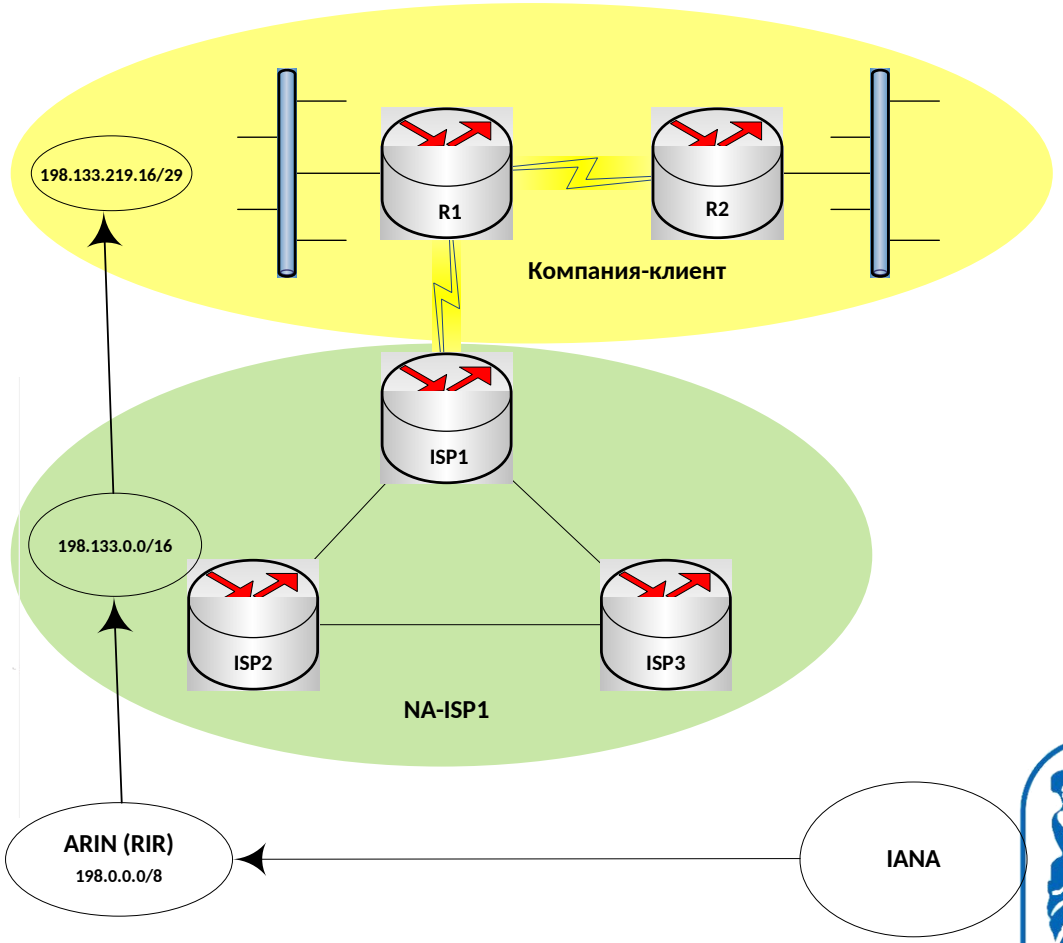
В случае успеха получаем MAC адрес устройства?



Распределение IP-адресов

- ICANN/IANA
- RIR - Regional Internet Registries
- NIR - National Internet Registries
- LIR - Local Internet Registries
- (обычно ISP)
- Конечный пользователь

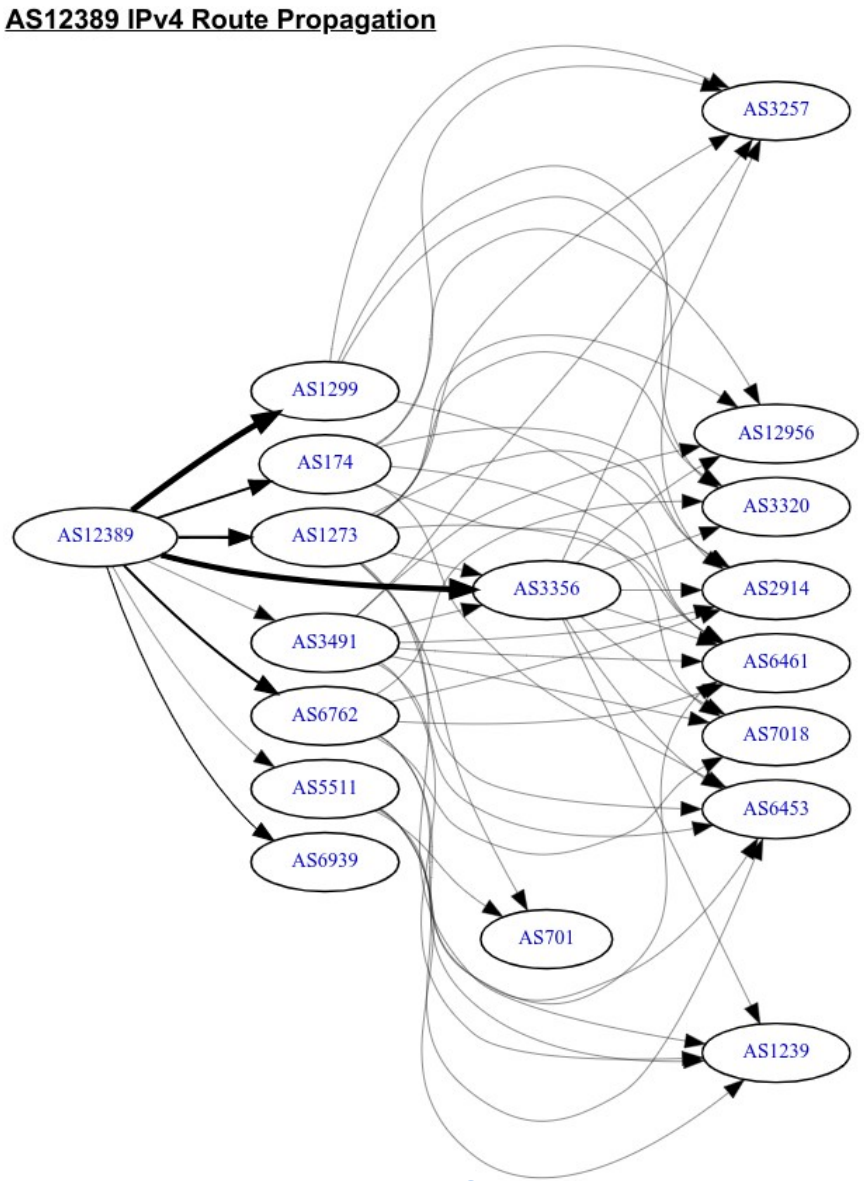




Типы валидных IP-адресов

- PI – Provider Independent
- PA – Provider Aggregatable

Автономные системы Рунета

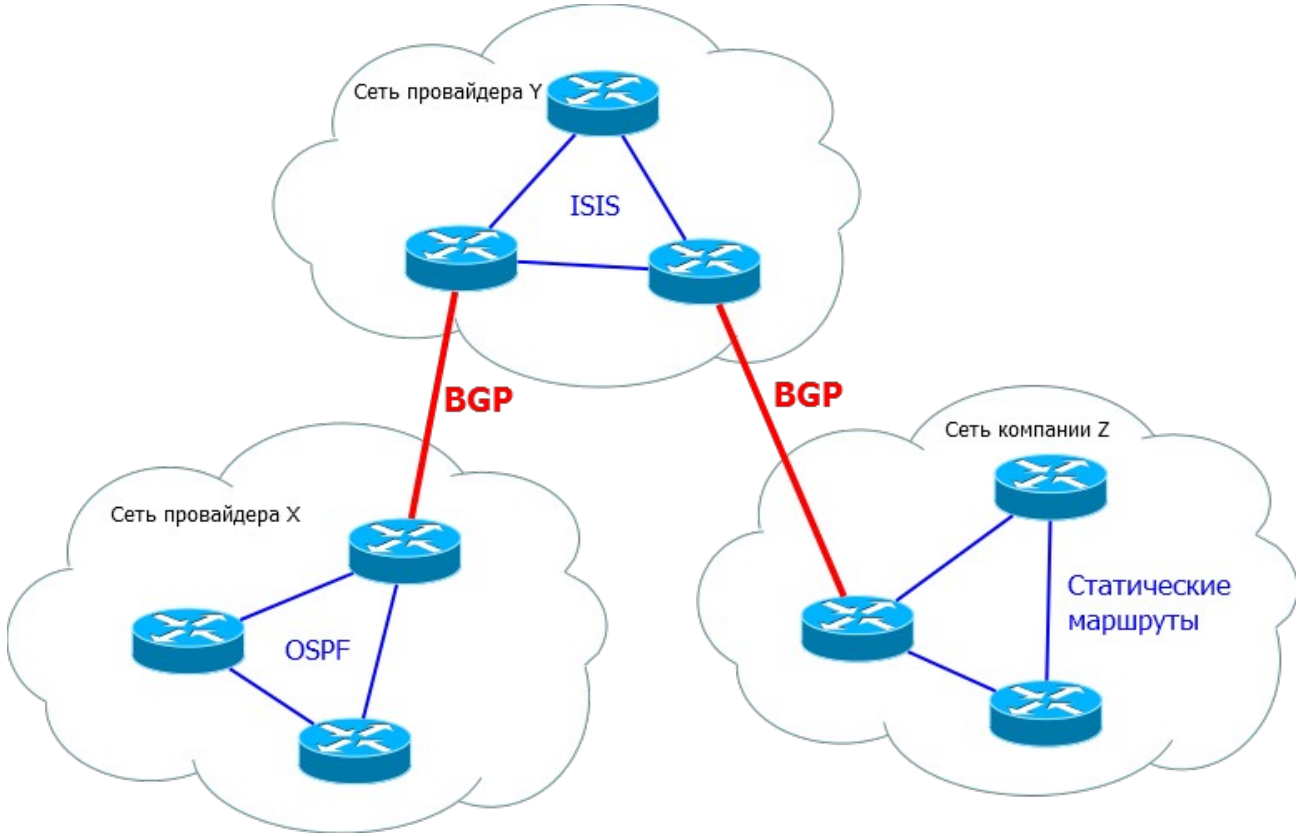


https://bgp.he.net/AS12389#_graph4

N	aut-num	as-name	Кол-во IPv4 адресов	Кол-во IPv6-сетей /48	Кол-во префиксов IPv4
1	AS12389	ROSTELECOM-AS	9563776	458759	2843
2	AS8402	CORBINA-AS	2300416	65536	6584
3	AS8359	MTS	1249280	1310720	147
4	AS3216	SOVAM-AS	1244416	65538	1299
5	AS12714	TI-AS	1238272	131072	210
6	AS31133	MF-MGSM-AS	790016	5951	360
7	AS20485	TRANSTELECOM	656128	720897	337
8	AS31200	NTK	546304	65536	84
9	AS42610	NCNET-AS	523264	524288	30

https://www.ididb.ru/autnum/

BGP – border gateway protocol



Border Gateway Protocol — это протокол граничного шлюза, разработанный для синхронизации сведений о маршрутизации и данных о связности между автономными системами.

На данный момент актуальная четвертая версия BGP, которая опубликована как RFC 4271 в 2006 году.

BGP - соединение

Устройства, между которыми устанавливается BGP-сессия называются BGP Peer или BGP-соседями.BGP не обнаруживает соседей автоматически – каждый сосед настраивается вручную.

Алгоритм работы BGP:

1) Изначальное состояние BGP-соседства – IDLE. Ничего не происходит. BGP находится в состоянии IDLE, если нет маршрута к BGP-соседу.

2) BGP-маршрутизатор (BGP-спикер/speaker или BGP-оратор) «слушает» и посылает пакеты на 179-й TCP порт. Когда «слушает» – это состояние CONNECT. Когда и ожидает ответа от соседа – это состояние ACTIVE.

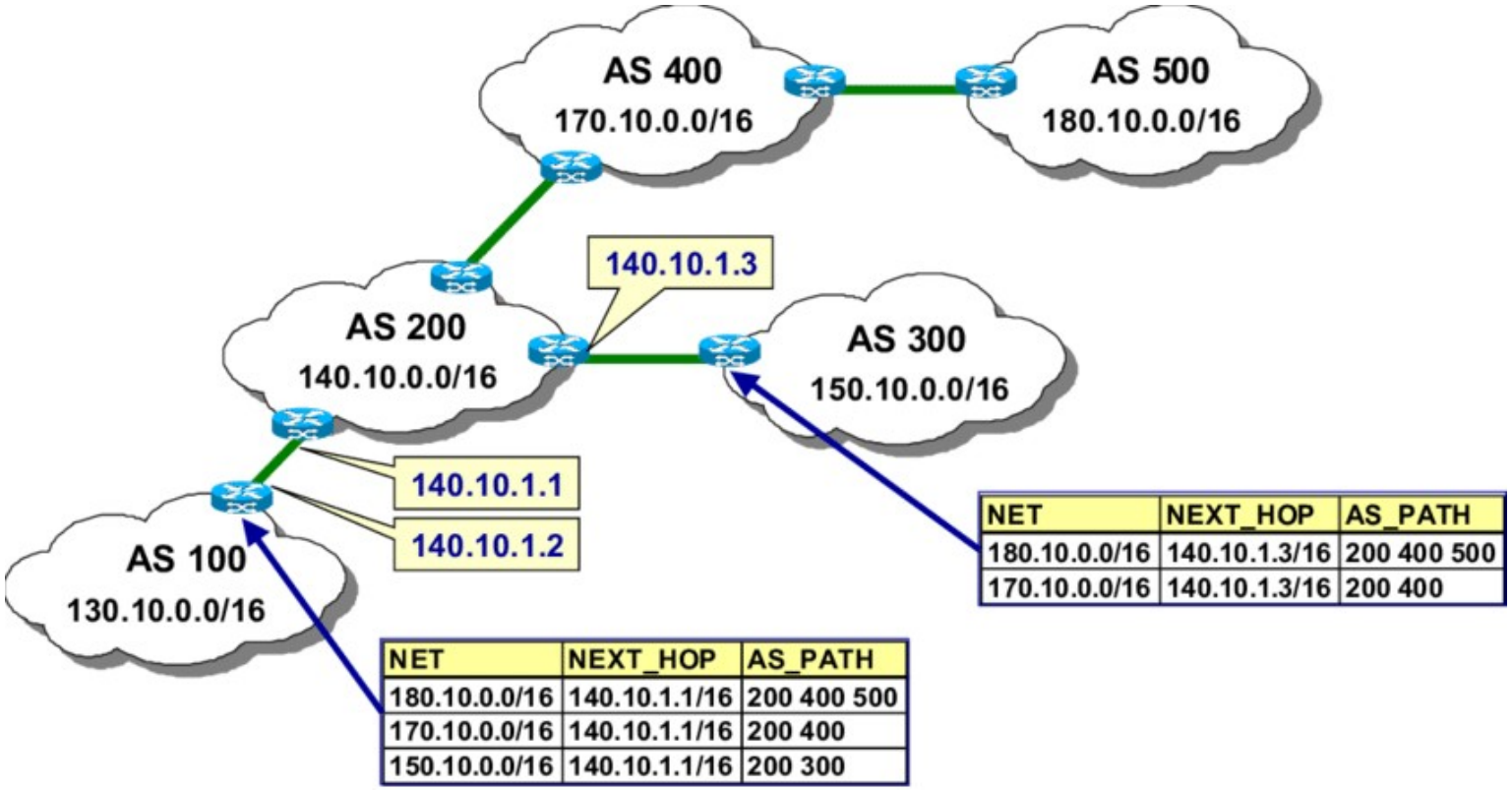
3) После того, как TCP-сессия установлена, BGP-ораторы начинают обмен сообщениями – OPEN, содержит версия протокола BGP, номер автономной системы отправителя, максимальное время в секундах, которое может пройти между сообщениями Update, BGP Identifier — номер маршрутизатора при наличии более одного канала связи.

4) BGP спикеры переходят в стабильное состояние ESTABLISHED. Это означает, что запущена правильная версия BGP и все настройки консистентны. Для каждого соседа можно посмотреть Uptime.

5) В первые мгновения после установки BGP-сессии в таблице BGP присутствует только информация о локальных маршрутах. Далее для обмена маршрутной информацией используется сообщение UPDATE. Сообщение UPDATE может нести информацию об одном новом маршруте или о удалении группы старых (одновременно). UPDATE передаются при каждом изменении в сети до тех пор пока длится BGP-сессия.

6) Пока всё хорошо, каждый BGP-маршрутизатор регулярно будет рассылать сообщения KEEPALIVE. Как и в любом другом протоколе это означает: «Я всё ещё жив». Это происходит с истечением таймера Keepalive – по умолчанию 60 секунд.

BGP – таблица маршрутов



NET	NEXT_HOP	AS_PATH
180.10.0.0/16	140.10.1.3/16	200 400 500
170.10.0.0/16	140.10.1.3/16	200 400

NET	NEXT_HOP	AS_PATH
180.10.0.0/16	140.10.1.1/16	200 400 500
170.10.0.0/16	140.10.1.1/16	200 400
150.10.0.0/16	140.10.1.1/16	200 300

BGP – Full view vs Default route

Full View. Полная информация о структуре Интернета.

- До любого адреса в Интернете можно просчитать путь от себя, узнать какие на маршруте AS. Через сайт RIPE можно посмотреть какие провайдеры обеспечивают транзит. Если вдруг у кого-то что-то *упадёт* маршрет через первый линк, BGP это отследит и перестроит свою таблицу маршрутизации для передачи данных через второго провайдера.
- Можно управлять маршрутами, вмешиваясь в стандартную процедуру выбора наилучшего пути,путём настройки, например, приоритетов маршрутов для определённых префиксов.
- Full View обязателен, если АС транзитная.
- Приходится платить производительностью: высокая утилизация оперативной памяти и весьма долгое изучение маршрутной информации после установления BGP-сессии. Например, после восстановления соединения с вышестоящим провайдером, полное восстановление связности может занять несколько минут.

Default Route.

- Сильно экономит ресурсы вашего оборудования.
- Проще в обслуживании, можно сказать. Не нужно по всей своей AS гонять сотни тысяч маршрутов.
- Никакого представления о состоянии интернета и реальной доступности получателей нет – вы просто доверяетесь вышестояшему провайдеру в надежде, что у него надёжность сети на порядки выше и нам не о чем беспокоиться.
- Балансировка и распределение входящего трафика при получении маршрута по умолчанию никак не затрагивается.

BGP - Выбор лучшего маршрута

Если существует несколько маршрутов до одной сети назначения, будет выбран только один из них. Каждый шаг в алгоритме выбора лучшего маршрута пытается устранить все, кроме одного маршруты к пункту назначения. Если на шаге алгоритма маршрутов все еще больше одного, будет выполнен переход на следующий шаг алгоритма. Таким образом, алгоритм работает до тех пор, пока это необходимо. В устройствах Juniper выбор наилучшего маршрута происходит по следующему алгоритму:

1) проверка на доступность next-hop в локальной таблице маршрутизации. Если next-hop не доступен, маршрут отбрасывается.
2) маршрутизатор выбирает маршрут с наибольшим Local Preference атрибутом.
3) маршрутизатор выбирает маршрут с кратчайшим AS Path length.
4) маршрутизатор выбирает маршрут с наименьшим значением атрибута Origin (то есть отдается предпочтение IGP).
5) маршрутизатор выбирает маршрут с наименьшим значением MED. Этот шаг выполняется, по умолчанию, только для маршрутов из одной AS.
6) маршрутизатор выбирает маршруты, полученные от соседей EBGP нежели полученные от IBGP соседей. Если остальные маршруты EBGP-маршруты, маршрутизатор переходит к шагу 9.
7) маршрутизатор выбирает маршрут с наименьшей метрикой IGP к анонсируемому BGP Next Hop.
8) если используется Route Reflection для IBGP пиринга, маршрутизатор выбирает путь с наименьшим Cluster-List length.
9) маршрутизатор выбирает маршрут от партнера с наименьшим Router ID.
10) маршрутизатор выбирает маршрут от партнера с наименьшим Peer Address.

Что для нас есть Интернет?

- Интернет – это абстрактное место размещения ресурсов, для доступа к которым необходимо знать:
  1) IP-адрес
  2) Порт
  3) Протокол
  4) Наименование ресурса

URL и URI

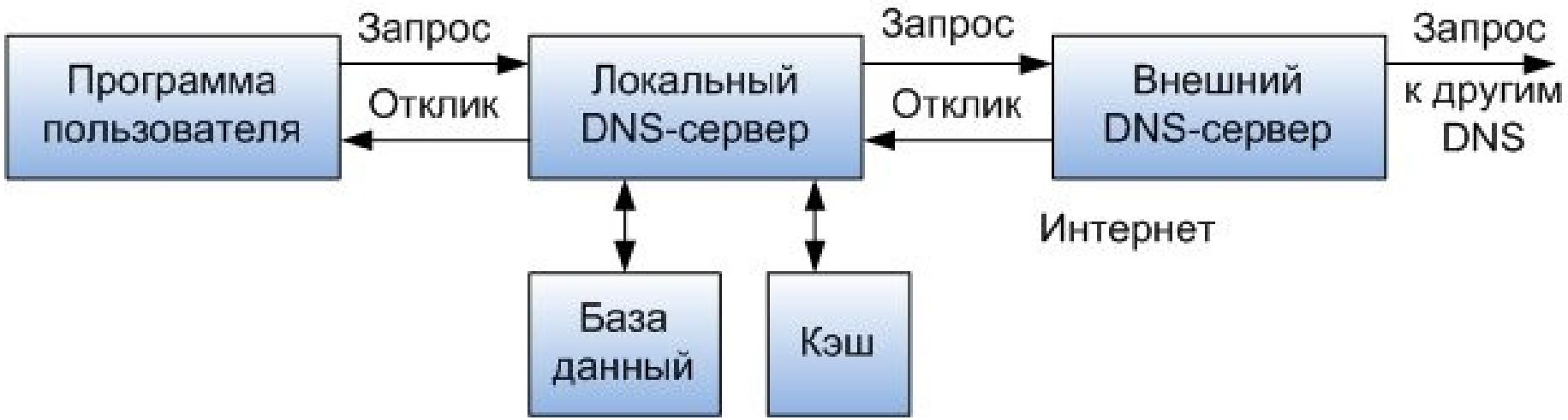
URI (англ. Uniform Resource Identifier) — унифицированный идентификатор ресурса. URI — последовательность символов, идентифицирующая абстрактный или физический ресурс. Ранее назывался Universal Resource Identifier — универсальный идентификатор ресурса.

- URI – имя и адрес ресурса в сети, включает в себя URL и URN
- [protocol]://www.[domain_name]:[port 80]/[resource location]?[query]#[fragment]
- URL – адрес ресурса в сети, определяет местонахождение и способ обращения к нему
- URN – имя ресурса в сети, определяет только название ресурса, но не говорит как к нему подключиться

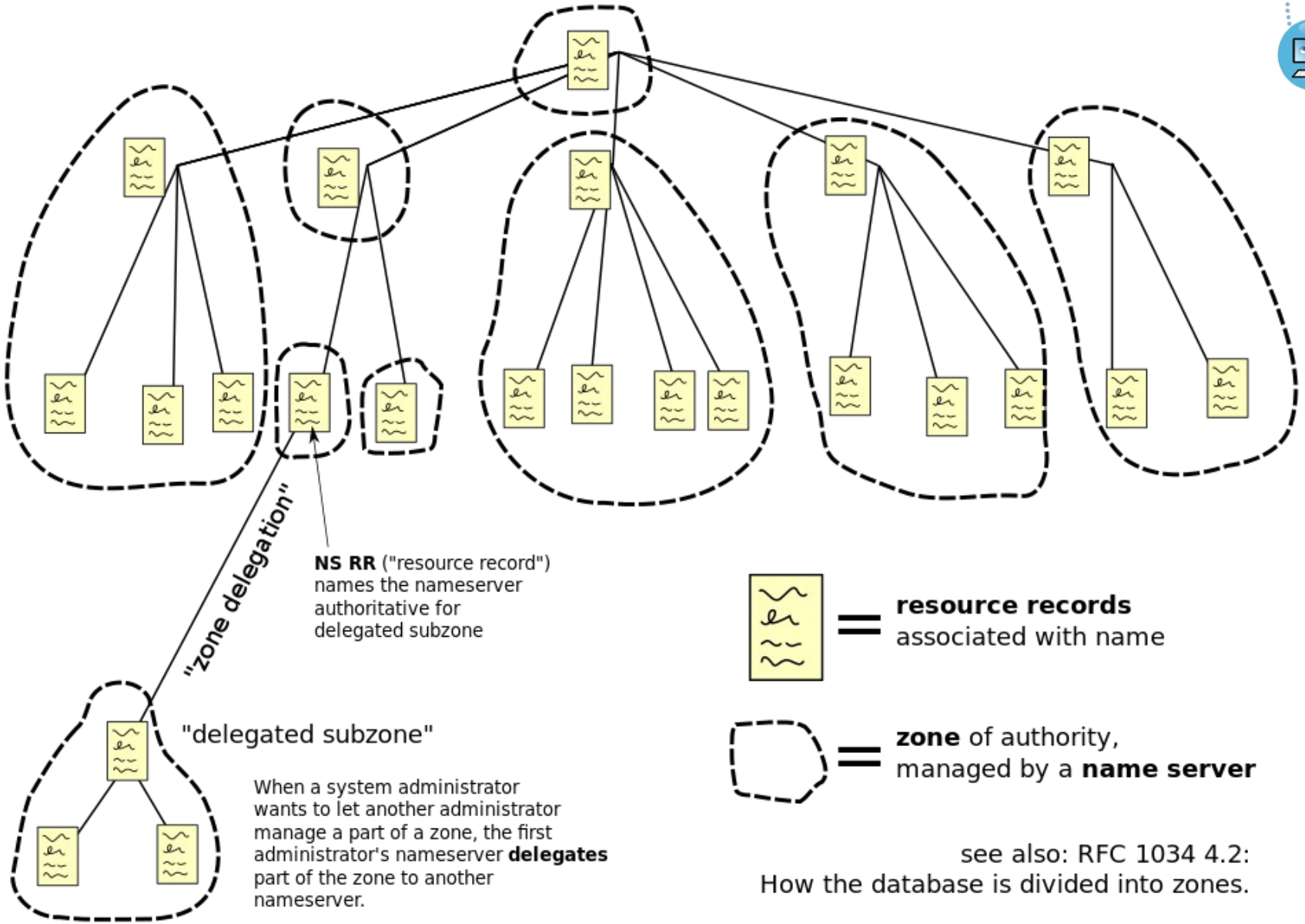
- URI – https://wiki.merionet.ru/images/vse-chto-vam-nuzhno-znat-pro-devops/1.png
- URL - https://wiki.merionet.ru
- URN - images/vse-chto-vam-nuzhno-znat-pro-devops/1.png

Domain Name System

1. Пользователь вводит запрос в строке браузера.
2. Браузер перенаправляет его DNS-серверу, который ищет совпадения между доменным именем и IP. При обнаружении совпадений браузер делает запрос по IP-адресу сервера и получает в ответ нужную информацию, после чего браузер отображает ее.
3. Если совпадения не обнаружены, тогда запрос перенаправляется корневому серверу.
4. Корневой сервер снова перенаправляет запрос серверу первого уровня, а тот отправляет запрос серверу второго уровня. Процесс продолжается до тех пор, пока совпадение не будет найдено.
5. Как только IP-адрес найден, браузер направляет запрос серверу, получает ответ и отображает полученную информацию.



Доменная система имён



DNS lookup (nslookup)

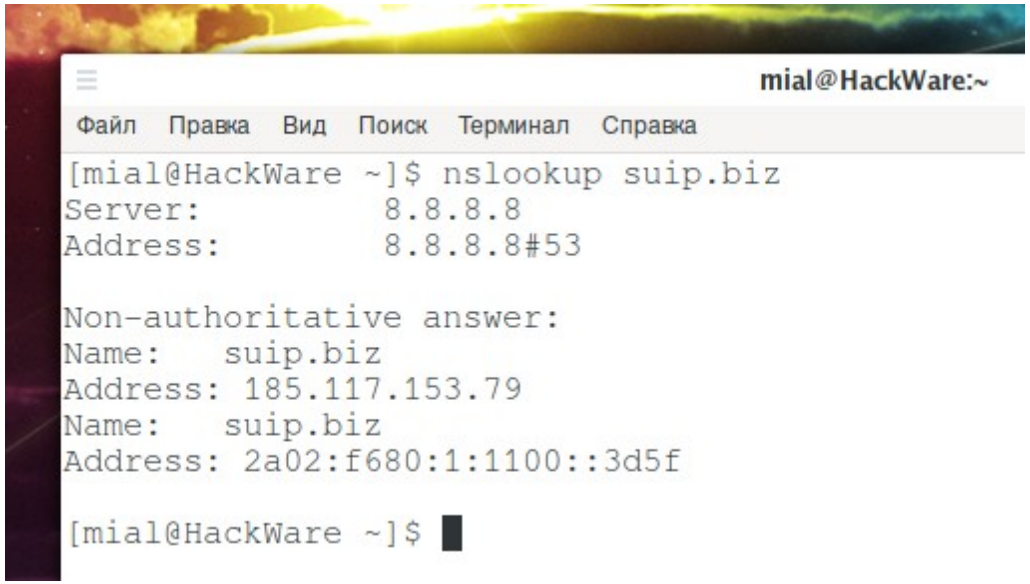
nslookup <опции> <Домен> <Сервер>

Домен - это то доменное имя, для которого необходимо посмотреть информацию,

Сервер - необязательный параметр, который указывает, что нужно использовать другой dns сервер.

Основные **опции** nslookup:

- type - тип информации, которую хотим получить, возможные типы: txt, soa, ptr, ns, mx, mr, minfo, mg, mb, hinfo, gid, cname, a, any;
- port - другой порт DNS сервера;
- recurse - использоваться другие DNS серверы, если на этом нет ответа;
- retry - количество попыток получить нужную информацию;
- timeout - время между попытками запросов к серверу;
- fail - пробовать другой сервер имен, если этот вернул ошибку.



Reverse DNS lookup

Обратный просмотр DNS (англ. reverse DNS lookup) — обращение к особой доменной зоне для определения имени узла по его IP-адресу с помощью PTR-записи.

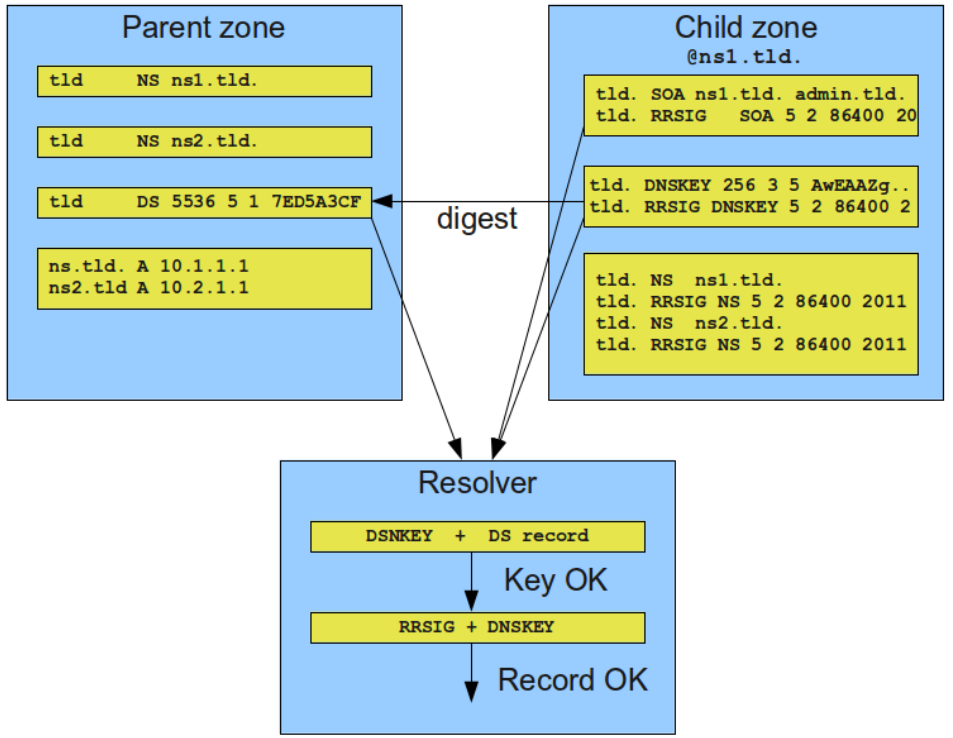
Для выполнения запроса адрес узла переводится в обратную нотацию, способ перевода зависит от версии IP:

IPv4-адрес 192.168.0.1
превращается в
1.0.168.192.in-addr.arpa.;
IPv6-адрес 2001:db8::567:89ab
превращается в
b.a.9.8.7.6.5.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.8.b.d.0.1.0.0.2.ip6.arpa.[1].

Благодаря иерархической модели управления именами появляется возможность делегировать управление зоной владельцу диапазона IP-адресов. Для этого в записях авторитетного DNS-сервера указывают, что за зону CCC.BBB.AAA.in-addr.arpa (то есть за сеть AAA.BBB.CCC.000/24) отвечает отдельный сервер.

Domain Name System Security Extensions

- DNSSEC — набор расширений IETF протокола DNS, позволяющих минимизировать атаки, связанные с подменой IP-адреса при разрешении доменных имён.
- DNSSEC предназначен для DNS-клиентам (англ. термин resolver) аутентичных ответов на DNS-запросы (или аутентичную информацию о факте отсутствия данных) и обеспечение их целостности.
- Используется криптография с открытым ключом. Все ответы от DNSSEC имеют цифровую подпись. При проверке цифровой подписи DNS-клиент проверяет верность и целостность информации.
- Не обеспечивается доступность данных и конфиденциальность запросов.



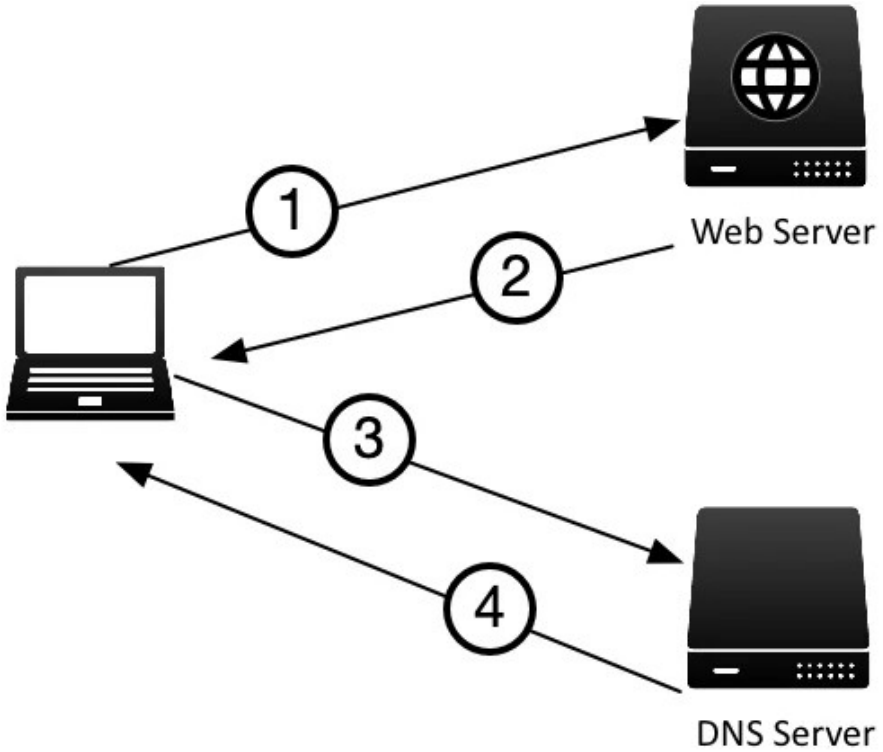
DNS-based Authentication of Named Entities (DANE)

1 The client browser connects to https://www.example.com

2 The web server replies with its certificate

3 The client asks its local DNS server for the TLSA record of www.example.com

4 The DNS server performs a normal DNS lookup for www.example.com TLSA record, uses **DNSSEC** to validate the response that came from the example.com authoritative name servers

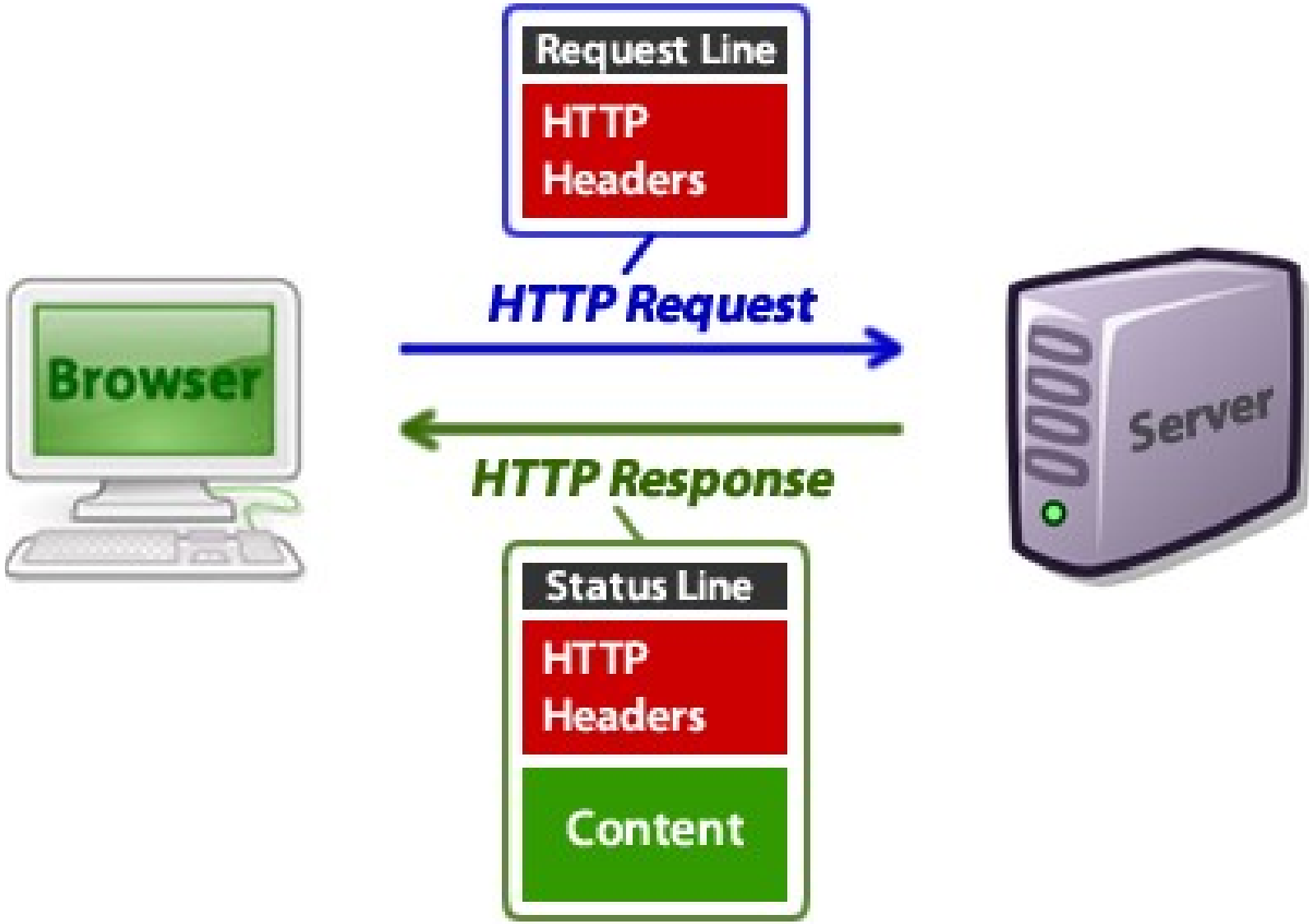


HTTP

- HyperText Transfer Protocol — «протокол передачи гипертекста» — протокол прикладного уровня передачи данных (изначально — в виде гипертекстовых документов в формате HTML, в настоящий момент используется для передачи произвольных данных). Основой HTTP является технология «клиент-сервер», то есть предполагается существование потребителей (клиентов), которые инициируют соединение и посылают запрос, и поставщиков (серверов), которые ожидают соединения для получения запроса, производят необходимые действия и возвращают обратно сообщение с результатом

HTTP


Cloud
Computing


Request Line
HTTP
Headers
HTTP Request
Browser
Server
HTTP Response
Status Line
HTTP
Headers
Content

HTTP – история

- HTTP/0.9 **Март 1991г.** Тим Бернерс-Ли, работавший в CERN, предложил механизм для доступа к документам в Интернете и облегчения навигации посредством использования гипертекста. Самая ранняя версия протокола HTTP/0.9 была впервые опубликована в январе 1992 г..
- HTTP/1.0 **Май 1996 г.** Выпущен RFC 1945, что послужило основой для реализации большинства компонентов HTTP/1.0.
- HTTP/1.1 **Июнь 1999г.** Введён режим «постоянного соединения»: TCP-соединение может оставаться открытым, что позволяет посылать несколько запросов за одно соединение. Клиент теперь обязан посылать информацию об имени хоста, к которому он обращается, что сделало возможной более простую организацию виртуального хостинга.
- HTTP/2 **Май 2015г.** опубликован RFC 7540. Протокол HTTP/2 является бинарным! Введено: мультиплексирование запросов, расстановка приоритетов для запросов, сжатия заголовков, загрузка нескольких элементов параллельно, посредством одного TCP соединения, поддержка проактивных push-уведомлений со стороны сервера. По данным W3Techs на февраль 2021 года, 50,2% из 10 млн самых популярных интернет-сайтов поддерживают протокол HTTP/2

HTTP. Структура

- Каждое HTTP-сообщение состоит из трёх частей, которые передаются в указанном порядке:
  1) Стартовая строка (англ. Starting line) — определяет тип сообщения;
  2) Заголовки (англ. Headers) — характеризуют тело сообщения, параметры передачи и прочие сведения;
  3) Тело сообщения (англ. Message Body) — непосредственно данные сообщения. Обязательно должно отделяться от заголовков пустой строкой.
- С 1 версии протокола стартовая строка является обязательным элементом, так как указывает на тип запроса/ответа.
- Для версии протокола 1.1 сообщение запроса обязательно должно содержать заголовок Host.

HTTP. Структура. Стартовая строка *запроса*

- *<Метод> <URI>* HTTP/*<Версия>*

- **Метод** (англ. Method) — название запроса, одно слово заглавными буквами
- **URI** определяет путь к запрашиваемому документу
- **Версия** (англ. Version) — пара разделенных точкой цифр. Например: 1.0

» GET /wiki/HTTP HTTP/1.0

HTTP. Структура. Стартовая строка *ответа*

- HTTP/<*Версия*> <*КодСостояния*> <*Пояснение*>

- **Версия** — пара разделенных точкой цифр как в запросе.
- **КодСостояния** (англ. Status Code) — три цифры. По коду состояния определяется дальнейшее содержимое сообщения и поведение клиента.
- **Пояснение** (англ. Reason Phrase) — текстовое короткое пояснение к коду ответа для пользователя. Никак не влияет на сообщение и является необязательным.

» HTTP/1.0 200 OK

HTTP. Структура


```
GET / HTTP/1.1
Host: xbb.uz
User-Agent: Mozilla/5.0  ...
Accept: text/html,applic ...
Accept-Language: ru-ru,r ...
Accept-Encoding: gzip,de ...
...
```

Запрос

Ответ

```
HTTP/1.0 200 OK
Server: nginx/0.7.67
Date: Tue, 08 Feb 2011 08: ...
Content-Type: text/html; c ...
X-Powered-By: PHP/5.2.12
Expires: Thu, 19 Nov 1981  ...
...
```

HTTP. Коды состояния

1xx Informational («Информационный»)

- **100** Continue («продолжай»)
- **101** Switching Protocols («переключение протоколов»)
- **102** Processing («идёт обработка»)

- В этот класс выделены коды, информирующие о процессе передачи. В HTTP/1.0 сообщения с такими кодами должны игнорироваться. В HTTP/1.1 клиент должен быть готов принять этот класс сообщений как обычный ответ, но ничего отправлять серверу не нужно. Сами сообщения от сервера содержат только стартовую строку ответа и, если требуется, несколько специфичных для ответа полей заголовка. Прокси-серверы подобные сообщения должны отправлять дальше от сервера к клиенту.

HTTP. Коды состояния

2xx Success («Успех»)

- **200** OK («хорошо»)
- **201** Created («создано»)
- **202** Accepted («принято»)
- **203** Non-Authoritative Information («информация не авторитетна»)
- **204** No Content («нет содержимого»)
- **205** Reset Content («сбросить содержимое»)
- **206** Partial Content («частичное содержимое»)
- **207** Multi-Status («многостатусный»)

- Сообщения данного класса информируют о случаях успешного принятия и обработки запроса клиента. В зависимости от статуса сервер может ещё передать заголовки и тело сообщения.

HTTP. Коды состояния

3xx Redirection («Перенаправление»)

- **300** Multiple Choices («множество выборов»)
- **301** Moved Permanently («перемещено навсегда»)
- **302** Moved Temporarily («перемещено временно»)
- **302** Found («найдено»)
- **303** See Other (смотреть другое)
- **304** Not Modified (не изменялось)
- **305** Use Proxy («использовать прокси»)
- **306** — зарезервировано (использовался в ранних спецификациях)
- **307** Temporary Redirect («временное перенаправление»)

- Коды класса 3xx сообщают клиенту что для успешного выполнения операции необходимо сделать другой запрос. Адрес, по которому клиенту следует произвести запрос, сервер указывает в заголовке Location. Допускается использование фрагментов целевого URI.

HTTP. Коды состояния

4xx Client Error («Ошибка клиента»)

- **400** Bad Request («плохой, неверный запрос»)
- **401** Unauthorized («не авторизован»)
- **402** Payment Required («необходима оплата»)
- **403** Forbidden («запрещено»)
- **404** Not Found («не найдено»)
- **405** Method Not Allowed («метод не поддерживается»)
- **406** Not Acceptable («неприемлемо»)
- **407** Proxy Authentication Required («необходима аутентификация прокси»)
- **408** Request Timeout («истекло время ожидания»)
- **409** Conflict («конфликт»)
- **410** Gone («удалён»)
- **411** Length Required («необходима длина»)
- **412** Precondition Failed («условие ложно»)

- Класс кодов 4xx предназначен для указания ошибок со стороны клиента. При использовании всех методов, кроме HEAD, сервер должен вернуть в теле сообщения гипертекстовое пояснение для пользователя.

HTTP. Коды состояния

4xx Client Error («Ошибка клиента»)

- **413** Request Entity Too Large («размер запроса слишком велик»)
- **414** Request-URI Too Large («запрашиваемый URI слишком длинный»)
- **415** Unsupported Media Type («неподдерживаемый тип данных»)
- **416** Requested Range Not Satisfiable («запрашиваемый диапазон не достижим»)
- **417** Expectation Failed («ожидаемое неприемлемо»)
- **422** Unprocessable Entity («необрабатываемый экземпляр»)
- **423** Locked («заблокировано»)
- **424** Failed Dependency («невыполненная зависимость»)
- **425** Unordered Collection («неупорядоченный набор»)
- **426** Upgrade Required («необходимо обновление»)
- **428** Precondition Required («необходимо предусловие»)
- **429** Too Many Requests («слишком много запросов»)
- **431** Request Header Fields Too Large («поля заголовка запроса слишком большие»)
- **434** Requested host unavailable. («Запрашиваемый адрес недоступен»)
- **444** Закрывает соединение без передачи заголовка ответа. Нестандартный код
- **449** Retry With («повторить с»)
- **451** Unavailable For Legal Reasons («недоступно по юридическим причинам»)

HTTP. Коды состояния

5xx Server Error («Ошибка сервера»)

- **500** Internal Server Error («внутренняя ошибка сервера»)
- **501** Not Implemented («не реализовано»)
- **502** Bad Gateway («плохой, ошибочный шлюз»)
- **503** Service Unavailable («сервис недоступен»)
- **504** Gateway Timeout («шлюз не отвечает»)
- **505** HTTP Version Not Supported («версия HTTP не поддерживается»)
- **506** Variant Also Negotiates («вариант тоже проводит согласование»)
- **507** Insufficient Storage («переполнение хранилища»)
- **508** Loop Detected («обнаружена петля»)
- **509** Bandwidth Limit Exceeded («исчерпана пропускная ширина канала»)
- **510** Not Extended («не расширено»)
- **511** Network Authentication Required («требуется сетевая аутентификация»)

- Коды 5xx выделены под случаи неудачного выполнения операции по вине сервера. Для всех ситуаций, кроме использования метода HEAD, сервер должен включать в тело сообщения объяснение, которое клиент отобразит пользователю.

HTTP. Методы

- OPTIONS
- GET
- HEAD
- POST
- PUT
- PATCH
- DELETE
- TRACE
- CONNECT

Каждый сервер обязан поддерживать как минимум методы GET и HEAD, также часто применяется метод POST. Если сервер не распознал указанный клиентом метод, то он должен вернуть статус 501 (Not Implemented). Если серверу метод известен, но он неприменим к конкретному ресурсу, то возвращается сообщение с кодом 405 (Method Not Allowed). В обоих случаях серверу следует включить в сообщение ответа заголовок Allow со списком поддерживаемых методов.

HTTP. Метод OPTIONS

- Используется для определения возможностей веб-сервера или параметров соединения для конкретного ресурса. В ответ серверу следует включить заголовок Allow со списком поддерживаемых методов. Также в заголовке ответа может включаться информация о поддерживаемых расширениях.
- Результат выполнения этого метода не **кэшируется**

HTTP. Метод GET

- Используется для запроса содержимого указанного ресурса. С помощью метода GET можно также начать какой-либо процесс. В этом случае в тело ответного сообщения следует включить информацию о ходе выполнения процесса.
- Клиент может передавать параметры выполнения запроса в URI целевого ресурса после символа «?»:
- Согласно стандарту HTTP, запросы типа GET считаются идемпотентными (*технически он вообще не должен менять состояние сервера*)

» GET /path/resource?param1=v1&param2=v2 HTTP/1.1

HTTP. Метод HEAD

- Аналогичен методу GET, за исключением того, что в ответе сервера отсутствует тело. Запрос HEAD обычно применяется для извлечения метаданных, проверки наличия ресурса (валидация URL) и чтобы узнать, не изменился ли он с момента последнего обращения.
- Заголовки ответа могут кэшироваться. При несовпадении метаданных ресурса с соответствующей информацией в кэше копия ресурса помечается как устаревшая

HTTP. Метод POST

- Применяется для передачи пользовательских данных заданному ресурсу. Передаваемые данные (например — значения в форме) включаются в тело запроса. С помощью метода POST могут загружаться файлы на сервер.
- В отличие от метода GET, метод POST не считается идемпотентным, то есть многократное повторение одних и тех же запросов POST может возвращать разные результаты.
- При результате выполнения 200 (Ok) в тело ответа следует включить сообщение об итоге выполнения запроса. Если был создан ресурс, то серверу следует вернуть ответ 201 (Created) с указанием URI нового ресурса в заголовке Location.
- Ответа сервера на выполнение метода POST **не кэшируется**

HTTP. Метод PUT

- Применяется для загрузки содержимого запроса на указанный в запросе URI. Если по заданному URI не существует ресурс, то сервер создаёт его и возвращает статус 201 (Created). Если же был изменён ресурс, то сервер возвращает 200 (Ok) или 204 (No Content). Сервер не должен игнорировать некорректные заголовки Content-*, передаваемые клиентом вместе с сообщением. Если заголовок не не допустим, то необходимо вернуть код ошибки 501 (Not Implemented).
- Методы POST и PUT различаются в понимании предназначения URI ресурсов. POST предполагает, что по указанному URI будет производиться *обработка* передаваемого содержимого. PUT предполагает что передаваемое содержимое соответствует находящемуся по данному URI ресурсу.
- Сообщения ответов сервера на метод PUT **не кэшируются**

HTTP. Метод PATCH

Аналогично PUT, но применяется только к фрагменту ресурса

- PATCH — это метод, который не объявляется ни безопасным, ни идемпотентным, и позволяет производить полное или частичное обновление, возможно с побочным эффектом на смежные ресурсы.

```
PATCH /echo/patch/json HTTP/1.1
Host: reqbin.com
Accept: application/json
Content-Type: application/json
Content-Length: 80

{
  "Id": 12345,
  "Customer": "John Smith",
  "Quantity": 1,
  "Price": 10.00
}
```

```
HTTP/1.1 200 OK
Date: Wed, 01 Jul 2020 13:29:24 GMT
Content-Type: application/json
Content-Length: 19
Connection: keep-alive
Set-Cookie:
__cfduid=d2fe29096c830a8faa499def718bda3931593610164;
expires=Fri, 31-Jul-20 13:29:24 GMT; path=/;
domain=.reqbin.com; HttpOnly; SameSite=Lax; Secure
CF-Cache-Status: DYNAMIC
cf-request-id: 03ac2a72a90000eff5fe81e200000001
Expect-CT: max-age=604800, report-uri="https://report-
uri.cloudflare.com/cdn-cgi/beacon/expect-ct"
Server: cloudflare
CF-RAY: 5ac079caa82feff5-EWR

{"success":"true"}
```

HTTP. Метод DELETE

- Удаляет указанный ресурс

```
DELETE /echo/delete/json HTTP/1.1
Authorization: Bearer
mt0dgHmLJMVQhvjpNXDyA83vA_PxH23Y
Accept: application/json
Content-Type: application/json
Content-Length: 19
Host: reqbin.com
```

```
HTTP/1.1 200 OK
Content-Length: 19
Content-Type: application/json

{"success":"true"}
```

HTTP. Метод TRACE

Возвращает полученный запрос так, что клиент может увидеть, какую информацию промежуточные серверы добавляют или изменяют в запросе

```
TRACE / HTTP/1.1
Host: www.tutorialspoint.com
User-Agent: Mozilla/4.0 (compatible;
MSIE5.01; Windows NT)
```

```
HTTP/1.1 200 OK
Date: Mon, 27 Jul 2009 12:28:53 GMT
Server: Apache/2.2.14 (Win32)
Connection: close
Content-Type: message/http
Content-Length: 39

TRACE / HTTP/1.1
Host: www.tutorialspoint.com
User-Agent: Mozilla/4.0 (compatible;
MSIE5.01; Windows NT)
```

HTTP. Метод CONNECT

Преобразует соединение запроса в прозрачный TCP/IP-туннель, обычно чтобы содействовать установлению защищённого SSL-соединения через нешифрованный прокси

```
CONNECT www.tutorialspoint.com HTTP/1.1
User-Agent: Mozilla/4.0 (compatible; MSIE5.01; Windows NT)
```

```
HTTP/1.1 200 Connection established
Date: Mon, 27 Jul 2009 12:28:53 GMT
Server: Apache/2.2.14 (Win32)
```

Пример: клиент запрашивает прокси-сервер HTTP для перенаправления TCP-соединения в желаемое место назначения. Затем сервер переходит к установлению соединения от имени клиента. После того как сервер установил соединение, прокси-сервер продолжает проксировать поток TCP к клиенту и от него. Только начальный запрос на подключение — HTTP — после этого сервер просто проксирует установленное TCP-соединение.

HTTP Cookies

- HTTP Cookie (куки) – небольшая порция текстовых данных, отправляемая веб-сервером и хранящаяся в браузере клиента. Браузер всякий раз при открытии страницы соответствующего сайта пересылает сохранённый фрагмент данных обратно веб-серверу через HTTP-заголовки.
- Куки используются для:
  - аутентификации пользователя;
  - хранения персональных предпочтений и настроек пользователя;
  - отслеживания состояния сеанса доступа пользователя;
  - ведения статистики о пользователях.

HTTP Cookies

- Для установки cookie в заголовке HTTP-ответа веб-сервера может содержаться указание браузеру:

```
HTTP/1.1 200 OK
Content-Type: text/html
Set-Cookie: name=value

Содержимое страницы
```

- Строка Set-Cookie отправляется только тогда, когда сервер желает, чтобы браузер сохранил куки. В этом случае браузер запомнит строку name=value и отправит её обратно серверу с каждым последующим запросом.

```
GET /spec.html HTTP/1.1
Host: www.example.org
Cookie: name=value
```

- Значение Cookie может быть изменено повторной отправкой сервером «Set-Cookie».

HTTPS

- HyperText Transfer Protocol Secure — расширение протокола HTTP, поддерживающее шифрование. Данные, передаваемые по протоколу HTTPS, «упаковываются» в криптографический протокол SSL или TLS.
- Вместо 80 порта в HTTP для HTTPS по умолчанию используется TCP-порт 443
- Для работы веб-сервера по протоколу HTTPS необходимо получить и установить в систему сертификат для этого веб-сервера. Сертификат состоит из 2 частей (2 ключей) — public и private:
  - public-часть сертификата используется для зашифровывания трафика от клиента к серверу в защищённом соединении,
  - private-часть — для расшифровывания полученного от клиента зашифрованного трафика на сервере.
  - на основе публичного ключа формируется запрос на сертификат в Центр сертификации, в ответ на который ЦС высылает подписанный сертификат. ЦС позволяет гарантировать, что держатель сертификата является тем, за кого себя выдаёт.

TLS

1) клиент подключается к серверу, поддерживающему TLS, и запрашивает защищённое соединение;
2) клиент предоставляет список поддерживаемых алгоритмов шифрования и хеш-функций;
3) сервер выбирает из списка, предоставленного клиентом, алгоритм и сообщает о своём выборе клиенту;
4) сервер отправляет клиенту цифровой сертификат для собственной аутентификации. Обычно цифровой сертификат содержит имя сервера, имя удостоверяющего центра сертификации и открытый ключ сервера;
5) клиент проверяет валидность серверного сертификата, относительно корневых сертификатов удостоверяющих центров (центров сертификации), а также проверяет не отозван ли серверный сертификат, связавшись с удостоверяющим центром;
6) для шифрования сессии используется сеансовый ключ. Получение общего секретного сеансового ключа клиентом и сервером проводится по протоколу Диффи-Хеллмана.

HTTPS

- В недавном историческом прошлом на одном IP-адресе мог работать только один HTTPS сайт.
- HTTPS не является отдельным протоколом. Это обычный HTTP, работающий через шифрованные транспортные механизмы SSL и TLS. Он обеспечивает защиту от атак, основанных на прослушивании сетевого соединения — от снифферских атак и атак типа man-in-the-middle, при условии, что будут использоваться шифрующие средства и сертификат сервера проверен и ему доверяют
- Для работы нескольких HTTPS-сайтов с различными сертификатами применяется расширение TLS под названием Server Name Indication (SNI)

HTML история

1991	HTML first published
1995	HTML 2.0
1997	HTML 3.2
1999	HTML 4.01
2000	XHTML 1.0
2002 - 2009	XHTML 2.0
2012	HTML5

После релиза HTML 4.01, акцент сместился к XHTML и его строгим стандартам.

XHTML 2.0 имел еще более строгие стандарты, чем 1.0, отклоняя страницы не были составлены правильно. Его популярность упала и от него практически отказались в 2009.

HTML5 на много более толерантен и может поддерживать разметку всех предыдущих версий.

HTML5 был официально представлен в 2012 и разрабатывался с 2004.

HTML. Структура документа


```
<HTML>
  <HEAD>
      Служебная информация
      <TITLE>
          информация об имени страницы
      </TITLE>
  </HEAD>
  <BODY>
          основная часть документа
  </BODY>
</HTML>
```

HTML. Ссылки

Адрес ссылки может быть как абсолютным, так и относительным. Абсолютные адреса должны начинаться с указания протокола (обычно http://) и содержать имя сайта. Относительные ссылки ведут отсчет от корня сайта или текущего документа.

Относительные ссылки

- Файлы в одной папке
- Ссылка

Файлы в разных папках

- Ссылка
- Ссылка
- Ссылка

Ссылка относительно корня сайта

- Курсы
- **Абсолютные ссылки**
- <a href="http://htmlbook.ru">

HTML5. Новые элементы

<article>	<figcaption>	<progress>
<aside>	<footer>	<section>
<audio>	<header>	<source>
<canvas>	<hgroup>	<svg>
<datalist>	<mark>	<time>
<figure>	<nav>	<video>

Это лишь некоторые из новых элементов в HTML5.

HTML. AJAX

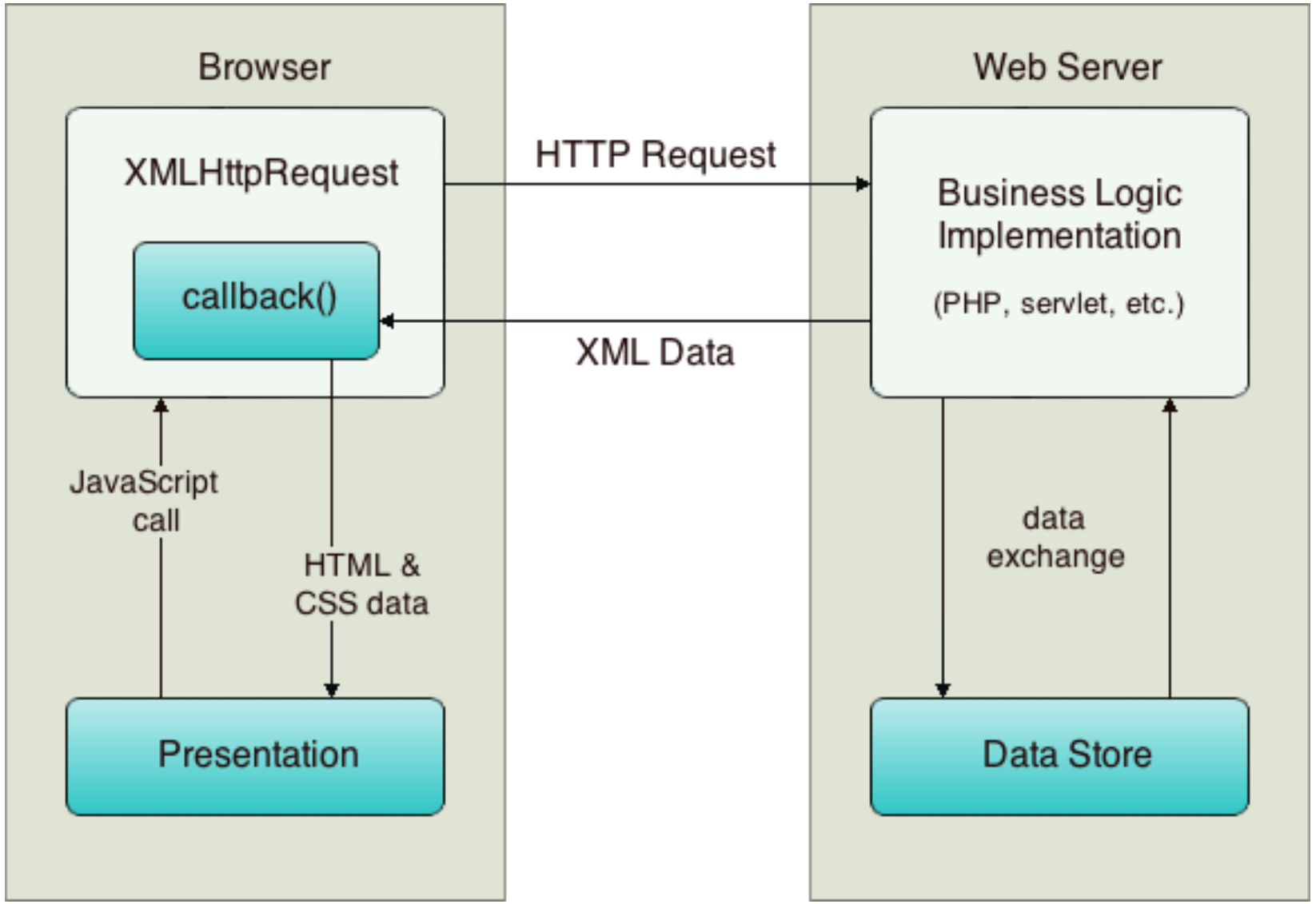
AJAX, Ajax (Asynchronous Javascript and XML — «асинхронный JavaScript и XML») — подход к построению интерактивных пользовательских интерфейсов веб-приложений, заключающийся в «фоновом» обмене данными браузера с веб-сервером.

AJAX — не самостоятельная технология, а концепция использования нескольких смежных технологий:

- технология динамического обращения к серверу «на лету», без перезагрузки всей страницы полностью, может осуществляться например с использованием XMLHttpRequest;
- в качестве формата передачи данных могут использоваться фрагменты простого текста, HTML-кода, JSON или XML.
- Изменение содержимого страницы производится согласно подходу DHTML с использованием скриптового языка JavaScript, CSS (каскадных таблиц стилей) и DOM (объектной модели документа).

HTML. AJAX


Cloud
Computing


Browser
XMLHttpRequest
callback()
HTTP Request
XML Data
Web Server
Business Logic
Implementation
(PHP, servlet, etc.)
JavaScript
call
HTML &
CSS data
data
exchange
Presentation
Data Store

REST api

REST API Methods

REST API

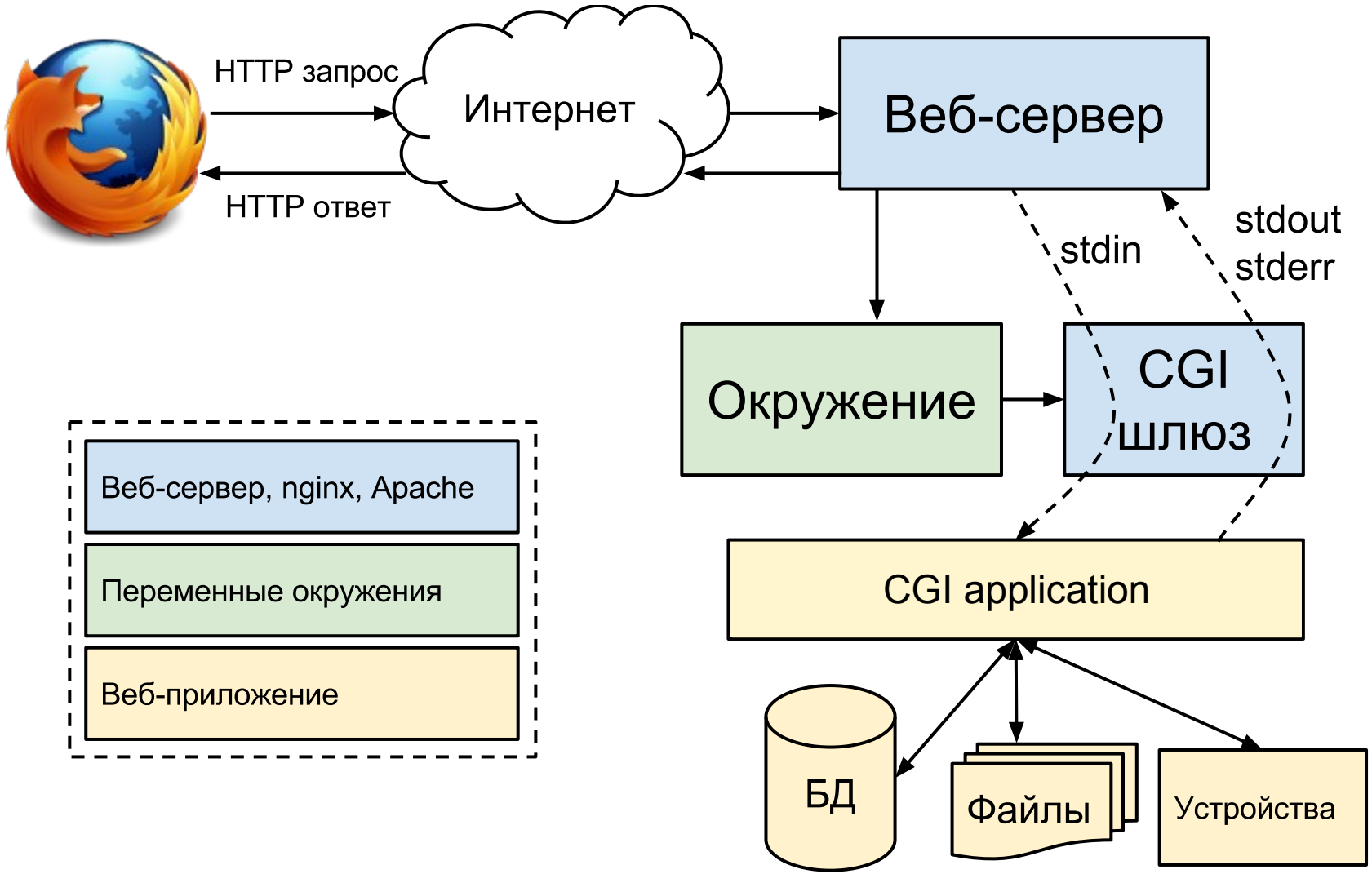
GET
Receive information about an API resource

POST
Create an API resource

PUT
Update an API resource

DELETE
Delete an API resource

CGI
Common Gateway Interface



Как работает CGI?

- Обобщенный алгоритм CGI:

1. Клиент запрашивает CGI-приложение по его URI.
2. Веб-сервер принимает запрос и устанавливает переменные окружения, через них приложению передаются данные и служебная информация.
3. Веб-сервер перенаправляет запросы через стандартный поток ввода (stdin) на вход вызываемой программы.
4. CGI-приложение выполняет все необходимые операции и формирует результаты в виде HTML.
5. Сформированный гипертекст возвращается веб-серверу через стандартный поток вывода (stdout). Сообщения об ошибках передаются через stderr.
6. Веб-сервер передает результаты запроса клиенту.

Переменные CGI-окружения

- **CONTENT_LENGTH** – величина данных, переданных методом POST и подлежащих считыванию в стандартное устройство ввода.
- **DOCUMENT_ROOT** – абсолютный путь до директории Web-сервера, откуда выполняется CGI-сценарий.
- **HTTP_REFERER** – путь URL, откуда пришёл пользователь, запустив CGI-сценарий.
- **HTTP_USER_AGENT** – имя и версия клиента, используемого пользователем.
- **QUERY_STRING** – строка запроса, часть строки адреса после знака “?”. По сути данные, переданные методом GET.
- **REMOTE_ADDR** – IP-адрес клиента.
- **REQUEST_METHOD** – метод, с помощью которого клиент передаёт данные.
- **SCRIPT_NAME** – имя CGI-сценария, который выполняется в данный момент.
- **SERVER_NAME** – доменное имя или IP-адрес сервера, на котором выполняется CGI-сценарий.
- **SERVER_SOFTWARE** – тип сервера, на котором выполняется CGI-сценарий.

Cookie в CGI

- Получение Cookie в среде CGI происходит с помощью переменной окружения HTTP_COOKIE, которая в точности повторяет HTTP-заголовок клиента «Cookie».
- Формат Cookie имеет следующий вид:
- name=value; name2=value2

CGI. Пример программы

```
#include <iostream>

using namespace std;

int main()
{
    cout << "Content-Type: text/html; charset=utf-8"  << endl << endl;
    cout << "<p>Привет!</p>" << endl;
    cout << "<p>Ваш IP-адрес: "<< getenv("REMOTE_ADDR")<< ".</p>" << endl;
    cout << "<p>Ваш браузер: "<< getenv("HTTP_USER_AGENT") <<
                                                ".</p>" << endl;

    return -1;
}
```

CGI. Вывод программы

pavel.mati.su/test.cgi

Привет!

Ваш IP-адрес: 217.9.88.22.

Ваш браузер: Mozilla/5.0 (Windows NT 6.1; WOW64; rv:68.0) Gecko/20100101 Firefox/68.0.

Преимущества и недостатки CGI

Преимущества

1. Процесс CGI скрипта не зависит от Веб-сервера и, в случае падения, никак не отразится на работе последнего
2. Может быть написан на любом языке программирования
3. Поддерживается большинством Веб-серверов

Недостатки

- Малая производительность CGI. Основная причина в том, что при **каждом** обращении к серверу для работы CGI – программы создается отдельный процесс, что требует большого количества системных ресурсов.
- Если приложение написано с ошибками, то возможна ситуация, когда браузер прервет соединение по истечении тайм-аута, а на серверной стороне процесс будет продолжаться, пока администратор не снимет его принудительно.
- CGI-программа – это готовый к исполнению файл, что препятствует миграции, обновлению, масштабированию и т.д..

FastCGI

CGI

Веб-сервер
(python3 -m http.server --cgi 8000)

Запрос Ответ Запрос Ответ Запрос Ответ

CGI (hello.**rb**) | CGI (hello.**py**) | ... | CGI (hello.**cgi**)

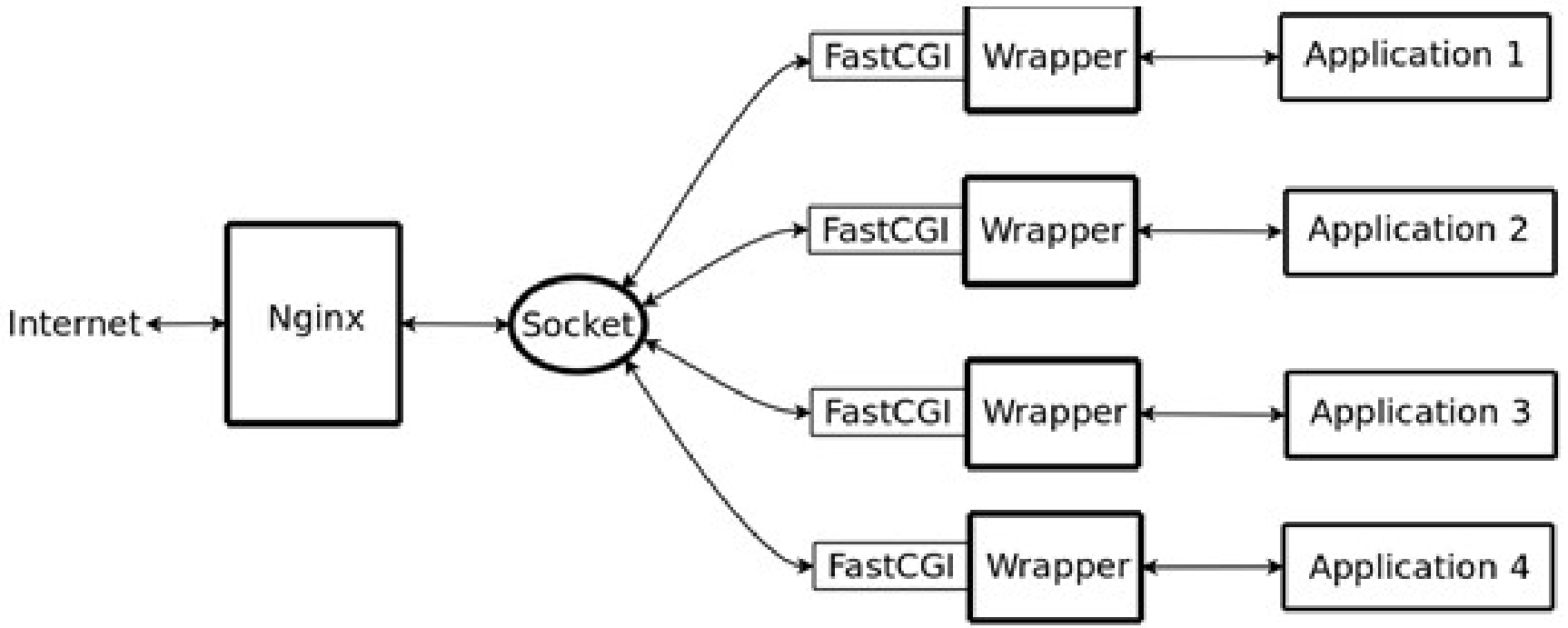
FastCGI

Веб-сервер
(Nginx)

Запрос Ответ

FastCGI
(spawn-fcgi)

FastCGI



FastCGI в двух словах - это CGI-программа, запущенная в цикле. Если обычная CGI-программа заново запускается для каждого нового запроса, то в FastCGI-программе используется очередь запросов, которые обрабатываются последовательно.

Как работает FastCGI

Обобщённый алгоритм:

- FastCGI программа единожды загружается в память в качестве демона (независимо от HTTP-сервера), а затем входит в цикл обработки запросов от HTTP-сервера.
- Один и тот же процесс обрабатывает несколько различных запросов один за другим, что отличается от работы в CGI-режиме, когда на каждый запрос создается отдельный процесс, "умирающий" после окончания обработки.
- Взаимодействие между FastCGI программой и HTTP сервером происходит через **сокет** (либо TCP, либо Unix)
- Одному типу запросов может быть сопоставлено несколько FastCGI демонов, это позволяет автоматически распределять нагрузку.

Спасибо за внимание!