Лекция 2. DNS TLS HTTP

Основны межсетевой передачи данных

Преподаватель:
Горев Данил Максимович
Разработчик образовательных проектов VK
(В том числе VK Education!)

Для связи - https://ketovx.t.me
Почта - danil.gorev@vk.team

Структура занятия

1. Фундамент — как пакеты находят дорогу
2. IP — протокол для «знакомств» между устройствами
3. DNS — телефонная книга интернета
4. BGP — тирлист сетей или «протокол дипломатов»
5. TCP vs UDP — не прошлый век, а вечный выбор
6. Эволюция HTTP: от 1.1 до HTTP/2
7. HTTP/3 и QUIC — рок-звезда современного интернета
8. TLS — секретный агент вашего соединения
9. Все вместе: как один запрос проходит через все слои

Как соединить два компьютера?

Телефонная сеть (коммутация каналов)	Интернет (коммутация пакетов)
✅ Предсказуемая задержка	❌ Задержка переменная
❌ Занята линия	✅ Линия общая для всех
❌ Нет отказоустойчивости	✅ Высокая отказоустойчивость
❌ Дорого (плата за время)	✅ Дешево (плата за трафик)

Аналогия:

- **Телефонный звонок** = Вы арендуете отдельную дорогу из точки А в точку Б. Больше никто ей пользоваться не может.
- **Интернет-запрос** = Вы отправляете письмо по общей почте. Оно едет в общих контейнерах с письмами других людей.

Вывод: Нужен протокол, который реализует идею "общей почты" для данных. Такой протокол — **IP (Internet Protocol)**.

IP — фундаментальная идея интернета

Основная задача: Доставить **пакет данных** от отправителя к получателю в условиях ненадежной сети.
Ключевые особенности:

1. **Пакетная коммутация:** Данные разбиваются на небольшие **IP-пакеты**.
2. **Негарантированная доставка:** IP "из коробки" не обещает, что пакет дойдет. Это задача протоколов верхнего уровня (например, TCP).
3. **Соединение не устанавливается:** Каждый пакет путешествует по сети независимо.

IP-адрес: что скрыто за цифрами?

IP-адрес — это не просто номер, это идентификатор сети и устройства.

Роли IP-адреса:

- **Идентификация хоста:** Уникальный "номер" устройства в сети.
- **Идентификация сети:** Определяет, к какой подсети принадлежит устройство (через маску подсети).

Аналогия с почтовым адресом:

- `ул. Ленина, д. 42, кв. 10` = `192.168.1.10`
- `ул. Ленина` = Сеть `192.168.1.0`
- `Дом и кв. 10` = Устройство с хост-частью `.10`

Формат IPv4:

- Четыре числа от 0 до 255, разделенные точками: `192.168.1.1`
- **Проблема:** Всего ~4.3 миллиарда адресов. Их не хватает.

IPv4 vs IPv6: смена эпох

IPv6 — это не просто больше адресов, это эволюция

IPv4	IPv6
192.168.1.1	2001:0db8:85a3::8a2e:0370:7334
~4.3 млрд	~340 секстиллионов (3.4×10^{38})
Универсальность, простота	Огромное пространство, встроенная безопасность (IPsec), автоконфигурация
Все еще доминирует, но адреса кончились	Активно внедряется, обязателен для мобильных сетей и IoT

Почему важен IPv6 для разработчика?

Скоро вам придется с ним работать! Многие облака и мобильные операторы уже используют IPv6. Ваши приложения должны быть готовы к его поддержке.

Итог: что делает IP?

IP — это про маршрутизацию, а не про надежность, IP закладывает основу, но не решает всех проблем.
Недостатки Internet Protocol:

- **Не гарантирует доставку** (пакеты могут теряться).
- **Не обеспечивает порядок** (пакеты могут приходить не по порядку).
- **Не управляет скоростью** (может вызвать "пробку" в сети).

Вывод: Для надежной связи поверх IP нужны другие протоколы.

Проблема: Цифры против людей

Зачем нужен DNS? Машинный язык vs человеческий

Проблема:

- Компьютеры находят друг друга по **IP-адресам** (например, `142.251.214.142`)
- Людям сложно запоминать и использовать цифровые адреса, они не подходят для рекламы
- IP-адрес не говорит ничего о своём владельце

Решение: Domain Name System (DNS) — «телефонная книга интернета»

Без DNS	С DNS
https://95.163.50.180	https://education.vk.company
Ненадежно: IP может измениться.	Надежно: имя остается постоянным.
Непонятно: что за сервис?	Понятно: бренд, сервис.

Вывод: DNS — это система преобразования *человекочитаемых имён* (education.vk.company) в *машинопонятные IP-адреса*.

Как работает DNS?

Что происходит, когда вы вводите адрес в браузере?

Ключевые роли:

- **Рекурсивный резолвер:** «Библиотекарь», который делает всю черновую работу по поиску
- **Корневой сервер:** «Указатель на отделы библиотеки» (например, `.com`, `.org`, `.ru`)
- **Авторитетный сервер:** «Официальный источник», где хранятся записи домена

Какие бывают DNS-записи? (Самое главное)

DNS — это не только про A-записи (IP). Это распределенная база данных о домене.

Какие бывают типы DNS-записей?

- A, AAAA (Address Records) - IPv4 и IPv6
- CNAME (Canonical Name) - Доменный алиас
- MX (Mail Exchange) - Почтовый сервер
- NS (Name Server) - Определяет Авторитетный сервер для нашего домена

TTL — «срок годности» записи

TTL (Time To Live) - Это число (в секундах), которое говорит резолверу, как долго **кешировать** ответ.

Пример: `TTL=300` — значит, следующие 5 минут все будут использовать старый IP-адрес.

На практике TTL - очень полезный инструмент, который позволяет контролировать кеширование в случае смены домена или уменьшить нагрузку на DNS-сервера (если мы большой сервис).

Проблемы с кешированием

Ситуация: Вы обновили DNS, но пользователи видят старую версию сайта.

Причина: Их локальный резолвер (провайдера) или ОС еще не «протухли» (TTL не истек).

Решение: Ждать или чистить кеш (`ipconfig /flushdns`, `sudo systemd-resolve --flush-caches`).

DNS Prefetching — упреждающее разрешение

Как работает: Браузер читает HTML, находит ссылки на другие домены и **заранее** резолвит их DNS, пока пользователь читает страницу.

```
<!-- Явно говорим браузеру предзагрузить DNS -->
<link rel="dns-prefetch" href="https://api.example.com">
```

Результат: Когда пользователь нажмет на кнопку, которая ведет на `api.example.com`, IP-адрес уже будет в кеше. Переход будет мгновенным.


ffee break .

BGP — «дипломатический протокол» интернета

IP протокол знает, *как* доставить пакет до соседа, но не знает, *какой путь* через всю паутину сетей выбрать.

Border Gateway Protocol (BGP):

- Это не протокол для передачи ваших данных, а **протокол для передачи маршрутов** между сетями.
- BGP — это язык, на котором большие сети объявляют друг другу: «**Через меня можно добраться до этих IP-адресов**».

Аналогия:

- **IP** = правила дорожного движения для одного письма.
- **BGP** = международные дипломатические соглашения между странами о том, через какие границы можно пересылать почту.

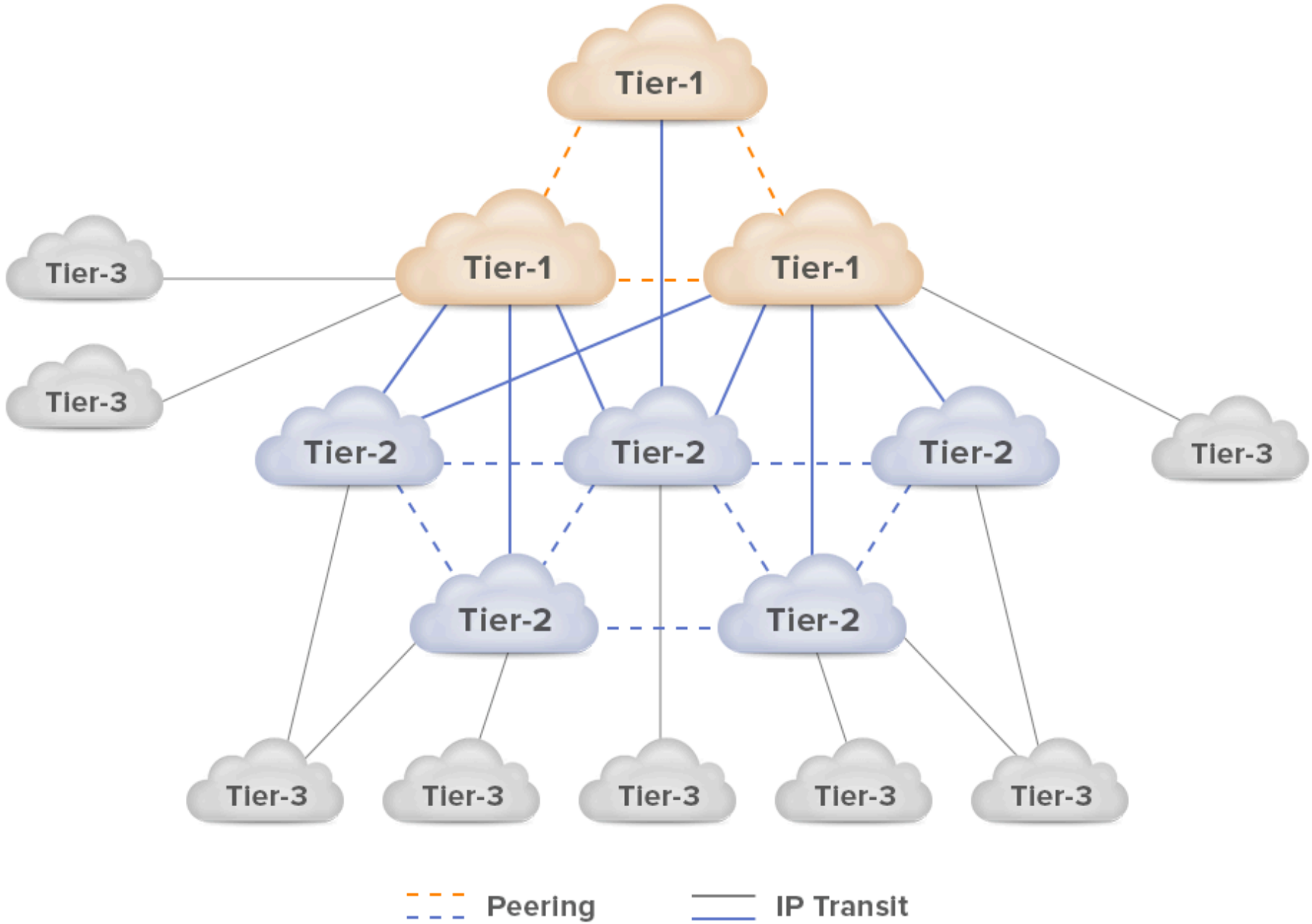
Ключевой принцип: BGP соединяет **Автономные Системы (AS)**.

Автономные Системы и тиры сетей

Автономная Система (AS) — это совокупность сетей под единым техническим управлением (провайдер, хостинг-компания, крупная корпорация). У каждой AS есть уникальный номер (ASN).

Проще всего представить AS в виде аналогии про страны - выше всех находятся сверхдержавы, следом идут автономные республики, следом города и самые маленькие сетевые "острова".

Уровень (Tier)	Роль (Аналогия)
Tier 1	**«Сверхдержавы»** Образуют ядро интернета. Договариваются о бесплатном обмене трафиком друг с другом (peering). Им не платит никто.
Tier 2	**«Региональные державы»** Покупают транзит у Tier 1, но договариваются между собой о прямых соединениях. Основа интернета в большинстве стран.
Tier 3	**«Локальные провайдеры»** Покупают транзит у Tier 2 и продают доступ конечным пользователям и бизнесу.


Tier-1
Tier-1
Tier-1
Tier-3
Tier-3
Tier-2
Tier-2
Tier-2
Tier-3
Tier-2
Tier-2
Tier-3
Tier-3
Tier-3
Tier-3
Tier-3
Peering
IP Transit

Как работает BGP? Объявление и выбор пути

1. **Объявление:** Хостинг-провайдер `AS-HOST` (Tier 3) объявляет своим провайдерам (Tier 2): «**Через меня можно добраться до сети 192.0.2.0/24**».
2. **Распространение:** Tier 2 провайдер `AS-T2` пересказывает это своим партнерам, в том числе Tier 1 провайдеру `AS-T1`: «**Через меня (а значит и через AS-HOST) можно добраться до 192.0.2.0/24**».
3. **Выбор пути:** Ваш провайдер получает несколько таких объявлений от разных партнеров. Он выбирает **самый короткий или самый выгодный** путь.

Почему BGP — ахиллесова пята интернета?

Когда падает BGP — падают Facebook, YouTube и целые страны.

- **Что было:** Провайдер «отзывает» свои BGP-объявления (намеренно или из-за сбоя).
- **Результат:** Его сеть (и все, кто к ней подключен) **пропадает из интернета**. Именно так падают Facebook и YouTube — их адреса перестают быть доступными для большей части мира.

Вывод: BGP — это гениальная, но хрупкая система, основанная на доверии. Его стабильность критически важна для работы

Что не смог решить IP?

IP доставил пакет, но что, если он потерялся, пришёл не вовремя или сломался?

Вспомним ограничения IP:

- ❌ **Не гарантирует доставку** (пакеты могут бесследно теряться).
- ❌ **Не сохраняет порядок** (пакет №2 может прийти раньше пакета №1).
- ❌ **Не контролирует перегрузку** (может устроить «пробку» в сети).

TCP — протокол контроля передачи

TCP (Transmission Control Protocol) — это «заказное письмо с уведомлением» поверх обычной почты (IP).

Ключевые механизмы TCP:

- Установление соединения («Рукопожатие»)
- Гарантированная доставка и порядок пакетов
- Контроль перегрузки

Для приложений (HTTP, SSH, БД): предоставляет абстракцию **надёжного байтового потока** между двумя процессами. «Как будто подключили трубу».

Порт — это «номер окна» в здании:

- **IP-адрес** = Адрес здания (например, `95.163.192.44`).
- **Порт** = Номер конкретной квартиры или окна в этом здании (например, `443` для HTTPS).
- TCP использует порты, чтобы направлять данные нужному приложению на сервере.

Где используется TCP?

Протокол/Сервис	Порт (пример)
HTTP/1.1, HTTP/2	80, 8080
HTTPS	443
SSH (Secure Shell)	22
Базы данных (PostgreSQL, MySQL)	5432, 3306
Почта (SMTP, IMAP)	25, 993

TCP — фундаментальный кирпич для 90% ваших приложений. Когда вы пишете код, который работает по сети, вы почти всегда неявно используете TCP.

UDP — минималистичный протокол

Если TCP — это заказное письмо, то UDP — это крик в толпу.

UDP (User Datagram Protocol) — это транспортный протокол, который предоставляет абсолютный минимум поверх IP.

Его философия:

- «Отправил и забыл».
- Максимальная скорость, минимальная задержка.
- Нет никаких гарантий.

Ключевое отличие: В UDP нет номеров пакетов, подтверждений, контроля перегрузки и установления соединения.

Осознанный выбор

TCP (Надёжный)	UDP (Быстрый)
✅ Переотправляет потерянные пакеты	❌ Отправил и забыл
✅ Пакеты в правильном порядке	❌ Могут прийти как угодно
✅ «Тормозит» при проблемах	❌ «Газ в пол», даже если сеть «легла»
✅ Three-way handshake	❌ Нет — сразу данные

Где царствует UDP? Сценарии применения

UDP незаменим там, где важна минимальная задержка (low latency)

1. DNS-запросы - Установка TCP-соединения для такого малого объёма данных заняла бы больше времени, чем сам запрос.
2. Голосовая связь и VoIP - Лучше услышать небольшой пропуск, чем задержку и буферизацию.
3. Видеотрансляции - Остановка и ожидание потерянных пакетов (как в TCP) приведут к «зависанию» картинки.
4. Онлайн-игры - Критически важна минимальная задержка

Эволюция HTTP: В погоне за скоростью

В 1999 году страница грузила 1-2 файла. В 2020-х — десятки JavaScript, CSS, изображений, шрифтов.

Чтобы оптимизировать загрузку контента, обезопасить пользователей, сделать соединение более надёжным и менее требовательным к оборудованию - был разработан прикладной протокол HTTP (Hyper Text Transfer Protocol).

Если коротко, то это швейцарский нож всего интернета.

Структура HTTP-запроса

Два основных "актора" при использовании HTTP - **клиент** и **сервер**. Важно понять, что и тем и другим может выступать практически любое устройство подключенное к сети (даже в рамках локальных сетей и даже в рамках одного устройства).

Любой HTTP-запрос состоит из трёх основных частей:

```
GET /api/users/123 HTTP/1.1          <- Стартовая строка
Host: example.com                    <- Заголовки (Headers)
User-Agent: Mozilla/5.0...
Accept: application/json
Authorization: Bearer token123
                                      <- Тело запроса (Body)
```

Методы запросов (Глаголы HTTP)

Метод описывает желаемое действие над указанным ресурсом.

Метод	Назначение
`HEAD`	**Получить** информацию о ресурсе
`GET`	**Получить** данные о ресурсе.
`POST`	**Создать** новый ресурс.
`PUT`	**Полностью обновить** существующий ресурс.
`PATCH`	**Частично обновить** ресурс.
`DELETE`	**Удалить** ресурс.

Пути, Заголовки и Тела запросов

Путь (Path или Endpoint) - Уникальный адрес ресурса на сервере. Например: https://api.shop.com/products/42

Заголовки (Headers) - Это «паспорт» и «инструкция» для запроса и ответа. Например: информация о клиенте (браузер/устройство), тип передаваемых данных, ключи для доступа к ресурсам и инструкции по кэшированию.

Тело запроса (Body) - Бинарная или текстовая структура, являющаяся частью `POST`, `PUT` и `PATCH` запросов. Например: JSON с данными, изображение или файл которые нужно сохранить на сервере.

Структура HTTP-ответа

Структура ответа для HTTP очень схоже со структурой запроса, что помогает как программам, так и разработчикам.

```
HTTP/1.1 200 OK                          <- Статусная строка
Content-Type: application/json           |
Date: Mon, 23 Jan 2023 10:00:00 GMT      |
Cache-Control: public, max-age=300       |<- Заголовки (Headers)
Server: nginx/1.18.0                     |

{"id": 123, "name": "John"}              <- Тело ответа (Body)
                                         (Данные, которые запрашивались)
```

Коды состояния HTTP (Status Codes)

Коды состояния ответов - трёхзначные числа, которые сообщают результат запроса.

- `2xx` Успех: `200 OK`, `201 Created`
- `3xx` Перенаправление: `301 Moved Permanently`, `304 Not Modified`
- `4xx` Ошибка клиента: `400 Bad Request`, `404 Not Found`, `403 Forbidden`
- `5xx` Ошибка сервера: `500 Internal Server Error`, `502 Bad Gateway`

От HTTP/1.1 до HTTP/2

Ограничения HTTP/1.1:

- **Очередь запросов (Head-of-Line Blocking):** В одном TCP-соединении запросы обрабатываются строго по очереди.
- **Множественные соединения:** Браузеры открывают 6-8 параллельных TCP-соединений к одному домену, чтобы обойти проблему. Это неэффективно.
- **Текстовый протокол:** Громоздкие заголовки, сложный для парсинга.

HTTP/1.1 — как одна касса в магазине с огромной очередью. Если один человек долго расплачивается, все остальные ждут.

HTTP/2: Революция под капотом

1. Бинарный протокол (Binary Framing Layer) - вместо текста протокол кодирует данные на стороне клиента и декодирует на стороне сервера, что позволяет разбить пакет на несколько отдельных отправок.

2. Мультиплексирование (Multiplexing) - Возможность в рамках одного TCP-соединения передавать множество параллельных потоков (streams). Загрузка картинок и CSS не заблокирует загрузку необходимого для работы сайта JS.

3. Принудительная загрузка (Server Push) - Сервер отдавая HTML может сразу передать список нужных для работы сайта CSS/JS файлов, чтобы клиент запросил их ещё до того, как спарсит HTML и отобразит контент.

4. Сжатие заголовков (HPACK) - Заголовки перед передачей сжимаются в компактный формат и не дублируются в рамках одного TCP-соединения (stream).

5. Приоритизация потоков - Сервер может указать порядок загрузки жизненно-необходимых компонентов сайта, которые клиент должен будет загрузить в первую очередь, игнорируя менее приоритетные ресурсы.

HTTP/2 vs HTTP/1.1

HTTP/1.1	HTTP/2
Множественные соединения, очередь	Одно соединение, мультиплексирование
Текстовый	Бинарный (кадры)
HOL на уровне протокола	HOL только на транспортном уровне
Нет сжатия заголовков	HPACK

Важное замечание: HTTP/2 **не меняет семантику** HTTP (методы, коды ответов). Это значит, что ваш API будет работать без изменений, но станет гораздо быстрее для клиентов.

Что это значит для разработчика?
Включите HTTP/2 на вашем веб-сервере (Nginx, Apache). Для большинства современных хостингов это уже стандарт. Вы сразу получите прирост производительности для ваших пользователей.

Проблемное наследие TCP

HTTP/2 решил много проблем, но одна осталась — транспортный уровень.

Нерешённая проблема HTTP/2:

- **Head-of-Line Blocking (HOL) на транспортном уровне.**
- HTTP/2 использует **одно** TCP-соединение с мультиплексированием.
- **Что это значит:** Если один потерянный TCP-пакет вызывает повторную передачу, это **блокирует все потоки** в этом соединении, даже те, чьи данные уже дошли!

Решение: QUIC — «TCP 2.0» поверх UDP

QUIC (Quick UDP Internet Connections) — это новый транспортный протокол от Google, который работает поверх UDP.

Почему UDP?

- UDP даёт свободу. Он не навязывает свои правила (доставка, порядок).
- QUIC может реализовать **собственные, более эффективные** механизмы надежности поверх UDP.

Что делает QUIC быстрым и надежным?

1. Решение HOL на транспортном уровне

- В QUIC есть **независимые потоки**. Данные в каждом потоке изолированы.
- **Результат:** Потеря пакета в одном потоке (например, при загрузке картинки) **не блокирует** доставку данных в других потоках (например, загрузку CSS или JS).

2. Встроенное шифрование

- Шифрование — не опция, а **обязательная часть** протокола.
- Заголовки и часть данных рукопожатия также зашифрованы.
- **Преимущество:** Больше безопасности, сложнее для цензуры и вмешательства провайдеров.

3. Ускоренное соединение (0-RTT)

- **Первое соединение:** Как в TLS 1.3, быстрое (1-RTT).
- **Повторное соединение:** Клиент может отправлять данные **вместе с первым пакетом** (0-RTT).
- **Аналогия:** Клиент может сказать: «Мне как обычно!».

Сравнение: Эволюция Handshake

TCP + TLS 1.2 (HTTP/1.1, HTTP/2):

```
Клиент: SYN ->
Сервер: SYN-ACK ->
Клиент: ACK -> TCP Connected!
Клиент: Client Hello ->
Сервер: Server Hello, Certificate ->
Клиент: ... -> TLS Connected!
```

QUIC (HTTP/3):

```
Клиент: Client Hello + Данные приложения ->
Сервер: Server Hello + Ответ на данные.
```

Как начать использовать HTTP/3 вам?

Проверьте поддержку у вашего хостинг-провайдера или CDN. Cloudflare: Включено по умолчанию для большинства тарифов.
Обновите ваше серверное ПО. Для самостоятельной настройки вам понадобятся последние версии nginx (с модулем `ngx_http_v3_module`) или Apache (с модулем `mod_http3`).

Вывод: HTTP/3 — это не будущее, а настоящее. Его внедрение — это самое простое и эффективное действие для ускорения вашего веб-приложения без изменения кода.

Почему HTTP недостаточно?

Потому что ваши данные идут открытым текстом.

Представьте:
Вы отправляете письмо по почте, написанное на **открытке**.
Каждый почтальон, сортировщик и просто любопытный человек может его прочитать.

HTTP = Отправка данных открытым текстом.

Ваши логины, пароли, сообщения, номера карт — всё видно как на ладони любому, кто перехватит трафик в кафе, офисе провайдера и т.д.

Решение: SSL/TLS - Это протокол, который превращает вашу «открытку» в **запечатанный бронированный конверт**, который может открыть только законный получатель.

Без TLS нет ни конфиденциальности, ни доверия к сайту.

Что даёт TLS? Три кита безопасности

Принцип	Объяснение
Шифрование	Преобразует данные в нечитаемый вид для всех, кроме отправителя и получателя.
Аутентификация	Гарантия, что вы общаетесь именно с тем сервером, с которым хотите (а не с мошенником).
Целостность данных	Гарантия, что данные не были изменены при передаче (злоумышленником или из-за ошибок).

SSL vs TLS: что использовать разработчику?

1. SSL 1.0, 2.0, 3.0 — разработаны Netscape в 90-х. **Полностью устарели и небезопасны.**
2. TLS 1.0, 1.1 — следующая версия, уже небезопасна по современным меркам.
3. TLS 1.2 (2008) — долгое время "золотой стандарт". Ещё широко используется.
4. TLS 1.3 (2018) — **современный стандарт**. Быстрее и безопаснее (сокращено рукопожатие, убраны уязвимые алгоритмы).

TLS для разработчика: Обязательные практики

1. Сертификаты и Let's Encrypt

- **Сертификат** — это тот самый «паспорт» для вашего сервера.
- **Let's Encrypt** — бесплатный и автоматизированный Центр Сертификации. Выдаёт доверенные сертификаты.
- **Практика:** Настроить автоматическое обновление сертификатов (например, с помощью `certbot`). Сертификаты имеют срок действия.

2. HSTS (HTTP Strict Transport Security)

- **Проблема:** Пользователь может вручную ввести `http://site.com`, и его данные уйдут незашифрованными.
- **Решение:** Заголовок `Strict-Transport-Security: max-age=31536000` + настройка Nginx/Apache для редиректа на HTTPS.
- **Что делает:** Браузер запоминает, что этот сайт нужно загружать ТОЛЬКО по `https://`, и автоматически исправляет `http` на `https`.

Вывод: В современном вебе HTTPS по умолчанию является базовым требованием, а не опциональным улучшением.

Инструменты для разработчика

Каждый протокол требует своего инструмента для анализа

Когда приложение работает медленно или с ошибками, нужно определить, на каком уровне происходит проблема. Эти инструменты — ваш набор для диагностики.

Уровень DNS и маршрутизации

`dig` / `nslookup` - Проверить, правильно ли разрешается доменное имя в IP-адрес. Отлаживать проблемы с DNS.

`traceroute` / (`tracert` **на Windows)** - Показать путь пакета до цели. Определить, на каком участке сети есть задержки или обрывы.

`whois` - Узнать информацию о владельце домена или IP-адреса.

Уровень транспорта и приложений (CLI)

`curl -v` - Отправить HTTP-запрос и посмотреть **все детали**: заголовки, тело, статус. Идеально для отладки API.

`telnet` / `nc` **(netcat)** - Проверить, доступен ли конкретный TCP-порт на сервере. «Постучаться» до сервера.

`openssl s_client` - Проверить SSL-сертификат, шифры и установить TLS-соединение.

Уровень приложений

Chrome DevTools (вкладка Network) — это ваш главный инструмент для отладки веба.

- **Что показывает:**
  - **Все сетевые запросы:** Видите каждый файл, API-вызов, изображение.
  - **Заголовки (Headers):** Точные заголовки запроса и ответа, куки.
  - **Время загрузки (Timing):** Где проводит время ваш запрос (DNS, TCP, TLS, Ожидание, Загрузка).

Вместо итога

Каждую из представленных технологий лучше понять на практике, попытаться "пощупать" её и увидеть в действии. Сухая теория не даст понимания, как протоколы и лежащие за ними технологии влияют на приложения.

Мини-пример напоследок

```
import socket

# Create a raw socket
s = socket.socket(socket.AF_INET, socket.SOCK_RAW, socket.IPPROTO_TCP)

# Craft IP and TCP headers (requires detailed knowledge of packet structure)
ip_header = b'...'
tcp_header = b'...'
packet = ip_header + tcp_header

# Send the packet to a specific IP address
s.sendto(packet, ('destination_ip_address', 0))
```

Knowledge deploy completed!

Преподаватель:
Горев Данил Максимович
Разработчик образовательных проектов VK

Для связи - https://ketovx.t.me
Почта - danil.gorev@vk.team