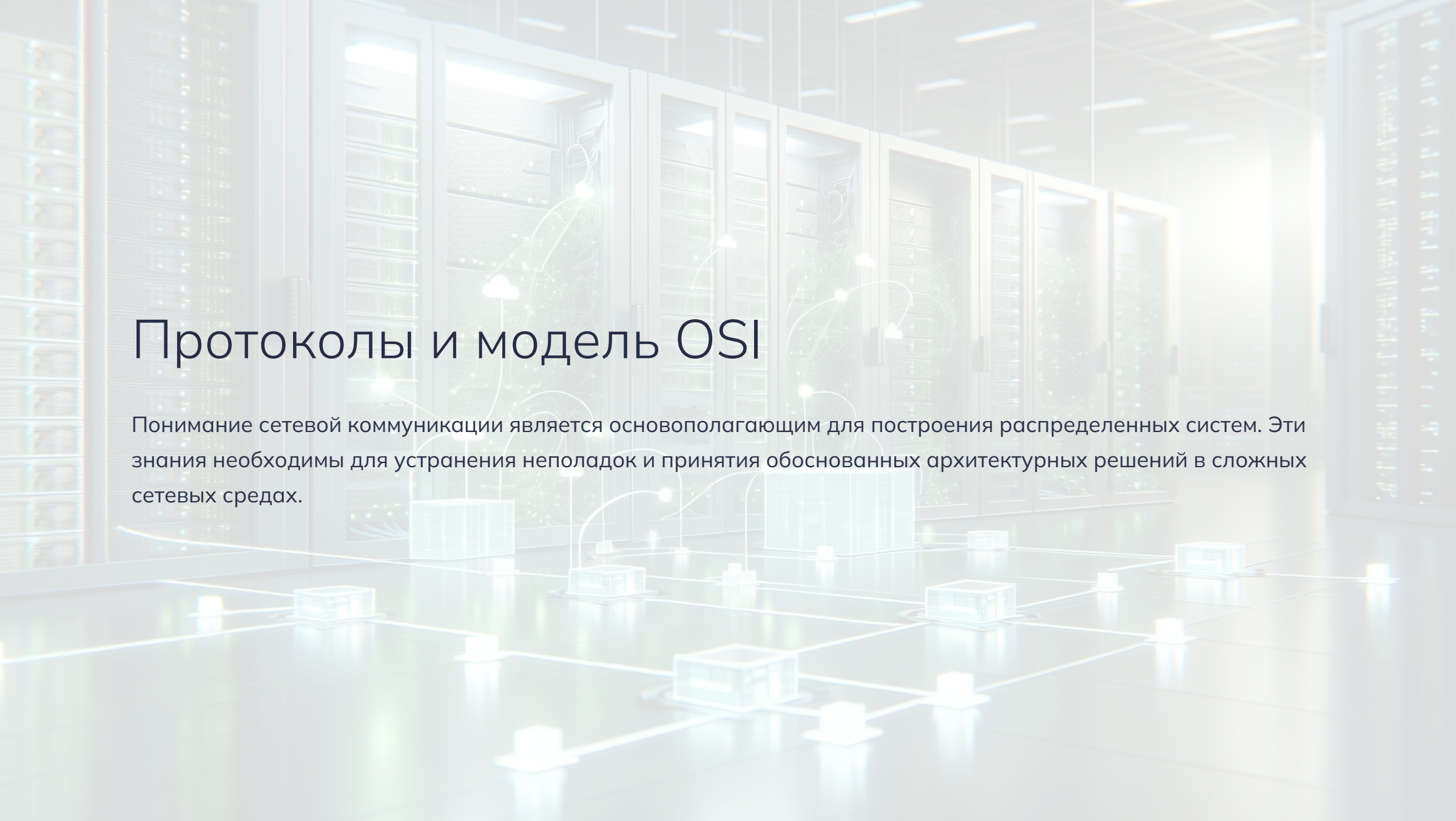


Протоколы и модель OSI

Понимание сетевой коммуникации является основополагающим для построения распределенных систем. Эти знания необходимы для устранения неполадок и принятия обоснованных архитектурных решений в сложных сетевых средах.

Понимание модели OSI

Модель OSI (Open Systems Interconnection) предоставляет стандартизированную структуру для сетевой коммуникации. Она состоит из семи уровней, каждый из которых имеет определенные обязанности и взаимодействует только с соседними уровнями.

Представьте модель OSI как процесс отправки письма. Вы пишете содержание (приложение), запечатываете в конверт с адресом (транспорт и сеть), передаете почтальону (канал данных), который физически доставляет его (физический уровень). На каждом этапе добавляется свой «слой» обработки, а получатель снимает эти слои в обратном порядке.

Семь уровней функционируют следующим образом:

Уровень 7 — Прикладной

Уровень 7 — Прикладной: Содержит протоколы, специфичные для приложений. Этот уровень предоставляет сетевые услуги непосредственно приложениям конечных пользователей.

Это то, с чем непосредственно работает пользователь или приложение. Когда вы открываете браузер и вводите адрес сайта — это прикладной уровень. Здесь «живут» программы, которые понимают, что означают ваши данные: это веб-страница, это email, это файл для скачивания. **Суть уровня: "Что именно мы хотим сделать в сети?"**

Примеры протоколов: HTTP/HTTPS (веб), SMTP/POP3/IMAP (электронная почта), FTP/SFTP (передача файлов), DNS (разрешение имен), SSH (удаленный доступ), Telnet, SNMP (управление сетью).

Уровень 6 — Презентационный

Уровень 6 — Презентационный: Обрабатывает перевод данных, шифрование, сжатие и преобразование форматов для обеспечения совместимости между различными системами.

Это «переводчик» и «упаковщик». Представьте, что вы отправляете документ другу, который говорит на другом языке и использует другую операционную систему. Презентационный уровень переводит данные в понятный формат, шифрует их для безопасности и сжимает для экономии места. **Суть уровня: "Как представить данные так, чтобы обе стороны их поняли?"** Например, когда видите "https://" в браузере — это презентационный уровень шифрует ваши данные через SSL/TLS.

Пример с HTTPS: HTTP это протокол высшего, прикладного, уровня, но, презентационный уровень может являться прослойкой для шифрования. В случае HTTPS, TLS шифрует трафик в нечитаемый набор символов, а сервер расшифровывает обратно. Также на этом же уровне можно сжать данные перед отправкой (gzip) и преобразовать кодировку текста, чтобы Windows-клиент и Linux-сервер поняли друг друга — данные остаются теми же по смыслу, но меняется их форма представления.

Примеры протоколов/технологий: SSL/TLS (шифрование), JPEG, GIF, PNG (форматы изображений), MPEG, MOV (видео), ASCII, EBCDIC, Unicode (кодировки текста), сжатие данных (GZIP, ZIP).

Уровень 5 — Сеансовый

Уровень 5 — Сеансовый: Управляет сеансами связи между приложениями, поддерживая состояние и контролируя диалог между сетевыми объектами.

Это «диспетчер долгого разговора». Сеансовый уровень важен, когда нужно **поддерживать постоянное соединение и состояние диалога** между приложениями, а не просто отправить запрос и получить ответ.

Суть уровня: "Как организовать и поддерживать непрерывный диалог, где обе стороны могут говорить в любой момент?"

Примеры использования:

- **WebSocket**: устанавливает постоянную сессию для чатов, онлайн-игр, уведомлений в реальном времени
- **gRPC с streaming**: поддерживает множественные потоки данных в одном соединении
- **Классические протоколы**: NetBIOS, RPC, телефония (VoIP)

Когда НЕ используется:

- **REST API / обычный HTTP**: каждый запрос независим (stateless), состояние управляется на прикладном уровне (cookies, JWT)
- **GraphQL через HTTP**: работает как REST — один запрос, один ответ

Уровень 4 — Транспорт

Уровень 4 — Транспорт: Это служба доставки, которая решает, КАК доставить данные. TCP — это как заказное письмо с уведомлением: каждый пакет подтверждается, гарантируется порядок и целостность. UDP — это как бросить листовку в почтовый ящик: быстро, но без гарантий доставки. **Суть уровня: "Как обеспечить передачу данных от приложения к приложению?"** Транспортный уровень разбивает большие данные на сегменты, нумерует их и отвечает за то, чтобы они дошли до адресата правильно. **Ключевое понятие: порты** — именно здесь используются порты (например, 80 для HTTP, 443 для HTTPS), чтобы определить, какому приложению предназначены данные.

Примеры протоколов: TCP (Transmission Control Protocol), UDP (User Datagram Protocol), SCTP (Stream Control Transmission Protocol), DCCP (Datagram Congestion Control Protocol).

Уровень 3 — Сеть

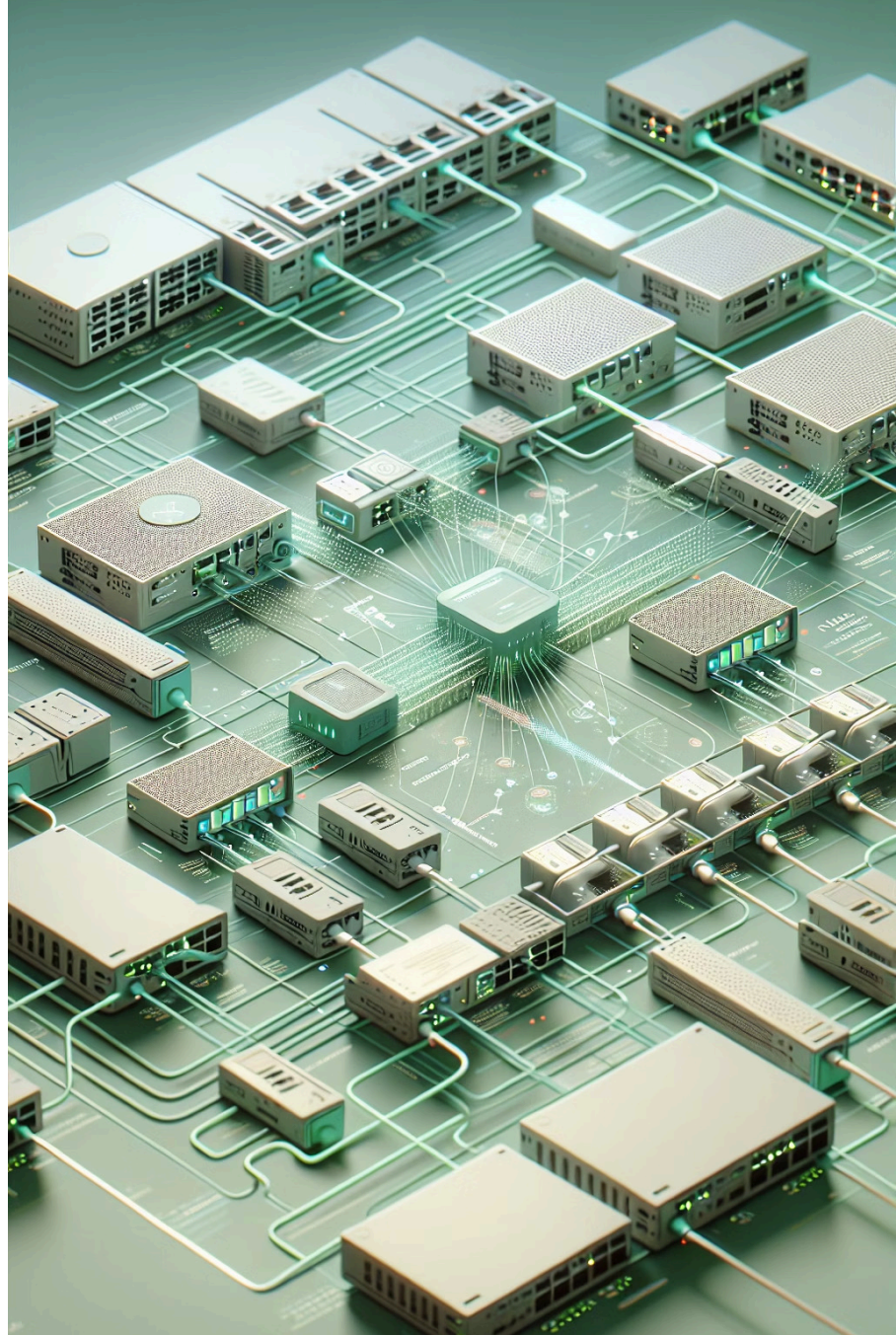
Уровень 3 — Сеть: Обрабатывает маршрутизацию и логическую адресацию. Это GPS и почтовая система интернета. Сетевой уровень использует IP-адреса (как почтовые адреса) для определения, КУДА отправлять данные, и выбирает оптимальный маршрут через множество сетей. **Суть уровня: "Как найти путь от источника к получателю через множество сетей?"** Когда вы отправляете пакет данных из Москвы в Сан-Франциско, сетевой уровень определяет, через какие маршрутизаторы его провести. **Важное отличие от уровня 4**: Уровень 3 работает с доставкой между устройствами (хост-хост), а уровень 4 — между приложениями (порт-порт).

Примеры протоколов: IPv4, IPv6, ICMP (ping, traceroute), IGMP (многоадресная рассылка), IPsec, ARP (Address Resolution Protocol), протоколы маршрутизации (RIP, OSPF, BGP).

Уровень 2 — канал передачи данных

Уровень 2 — канал передачи данных: управляет коммуникацией на уровне кадров между устройствами в одном сегменте сети, включая обнаружение и исправление ошибок.

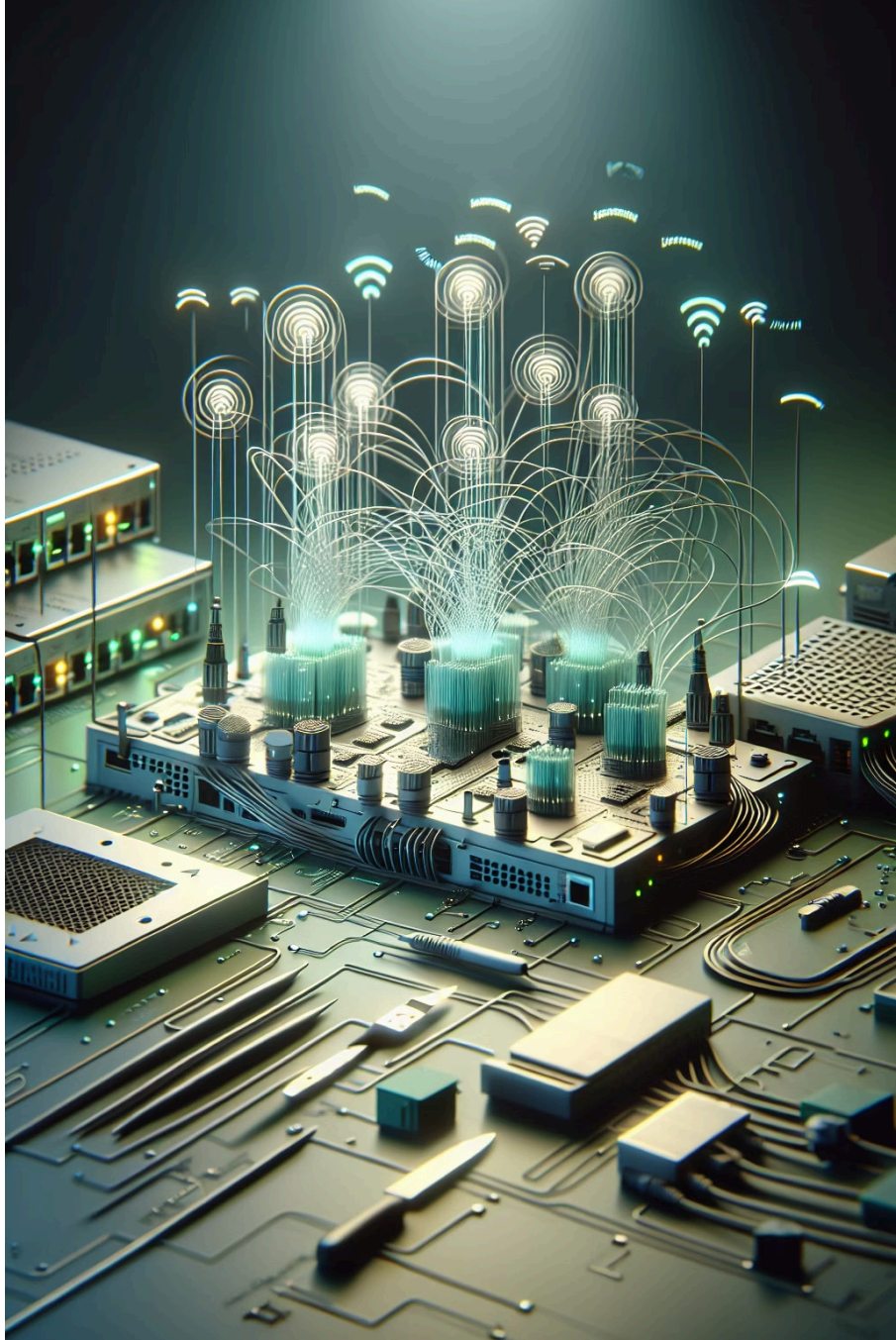
Это «локальный регулировщик трафика». Представьте, что сетевой уровень доставил пакет в ваш район (локальную сеть), а канал данных — это тот, кто находит конкретный дом на вашей улице. Он использует MAC-адреса (физические адреса сетевых карт) вместо IP-адресов. **Суть уровня: "Как организовать передачу данных между устройствами в одной локальной сети?"** Этот уровень берет пакеты с уровня 3 и упаковывает их в «кадры» (frames) для передачи по физической среде. **Важная роль**: Обнаружение коллизий (когда два устройства пытаются передавать одновременно) и обеспечение того, чтобы данные не потерялись на пути от роутера к вашему компьютеру.



Уровень 1 — физический

Уровень 1 — физический: Это чистая физика и «железо». Физический уровень не понимает, что такое IP-адрес или веб-страница — он просто передает биты (0 и 1) в виде электрических импульсов по медному проводу, световых импульсов по оптоволокну или радиоволн по воздуху. **Суть уровня: "Как преобразовать цифровые данные в физические сигналы и передать их по среде?"** Это напряжение в кабеле, частота сигнала, форма разъема. **Простая аналогия**: Если верхние уровни — это содержание и адрес письма, то физический уровень — это сам грузовик, который везет почту, асфальт, по которому он едет, и бензин в баке.

Примеры технологий: Витая пара (Ethernet-кабели: Cat5, Cat6), оптоволокно, коаксиальный кабель, радиоволны (Wi-Fi, Bluetooth, 4G/5G), USB, HDMI, хабы (hubs), репитеры, модемы, физические разъемы (RJ-45).

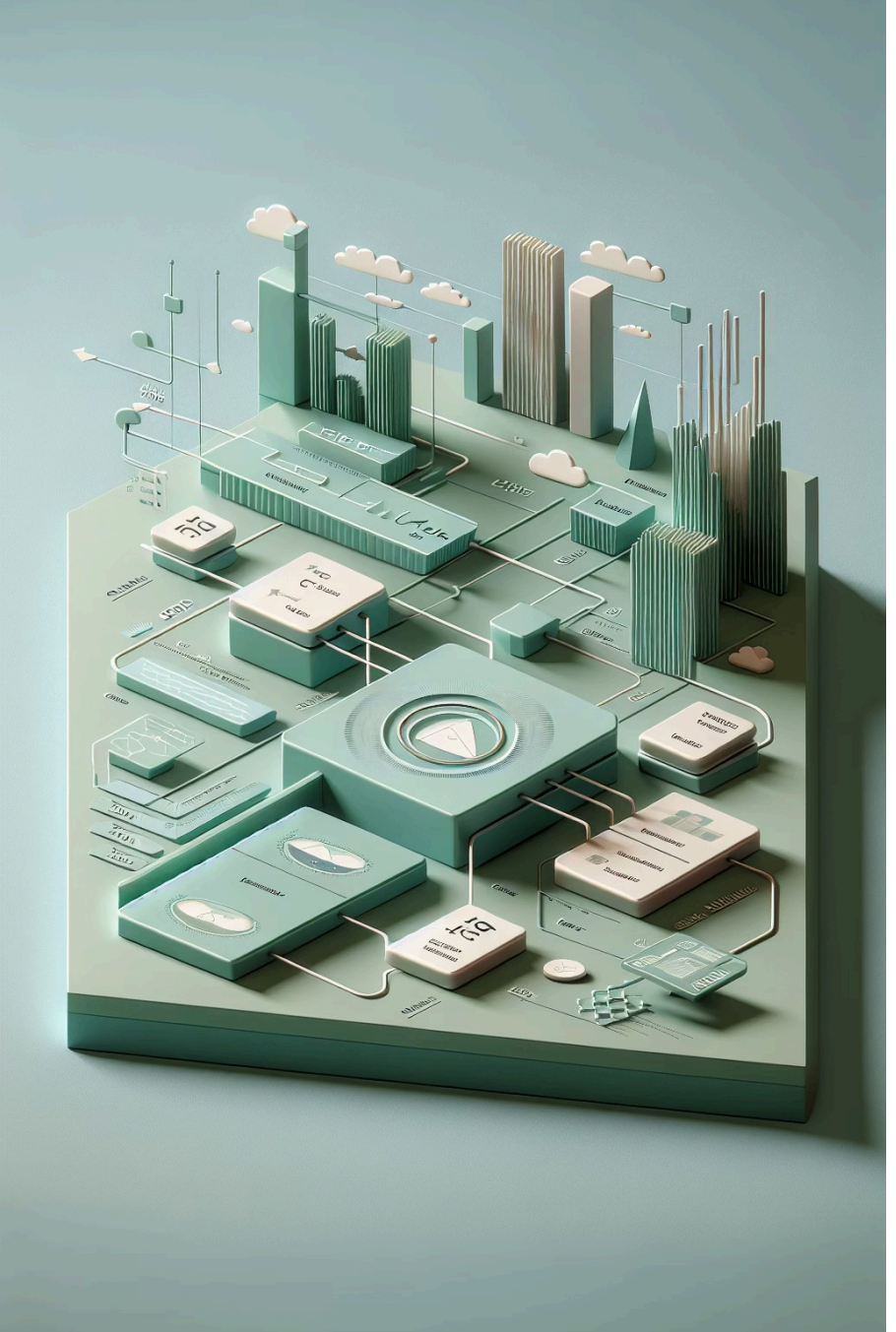


Реальность: модель TCP/IP

Хотя модель OSI обеспечивает отличное концептуальное понимание, в практической реализации сетей используется модель TCP/IP, которая более точно отражает принципы функционирования современных интернет-коммуникаций:

Уровень модели OSI	**Уровень модели TCP/IP**	**Что вы увидите на самом деле**
Приложение, презентация, сеанс	Приложение	HTTP, HTTPS, SMTP, FTP, DNS
Транспорт	Транспорт	TCP, UDP
Сеть	Сеть	IP, ICMP, протоколы маршрутизации
Канал передачи данных, физический	Доступ к сети	Ethernet, WiFi, сотовая связь

Разработка приложений в основном затрагивает прикладной и транспортный уровни модели TCP/IP.



TCP и UDP: выбор транспортного уровня

Выбор между TCP и UDP существенно влияет на проектирование системы и характеристики производительности. Каждый протокол предназначен для различных случаев использования в зависимости от требований к надежности и производительности.

TCP (протокол управления передачей) — надежная передача данных

TCP обеспечивает гарантированную упорядоченную доставку данных с комплексной обработкой ошибок:

Основные характеристики

- **Ориентированность на соединение**: устанавливает соединение перед отправкой данных
- **Надежная доставка**: гарантирует, что все данные будут доставлены в правильном порядке
- **Управление потоком**: предотвращает перегрузку получателя
- **Обнаружение и исправление ошибок**: автоматически повторно передает потерянные пакеты
- **Управление перегрузкой**: замедляет работу, когда сеть загружена

Подходящие случаи использования

- Просмотр веб-страниц (HTTP/HTTPS)
- Передача файлов (FTP, SFTP)
- Электронная почта (SMTP, IMAP)
- Подключение к базам данных
- Приложения, требующие гарантированной доставки данных

Компромиссы: Более высокая задержка из-за установления соединения, подтверждений и механизмов исправления ошибок.

UDP (протокол пользовательских дейтаграмм) — высокопроизводительная передача данных

UDP ставит скорость и эффективность выше гарантий надежности:

Ключевые особенности

- **Без соединения**: передает данные без установления соединений
- **Доставка по принципу «лучших усилий»**: нет гарантий доставки или упорядочения
- **Отсутствие управления потоком**: передача с максимально возможной скоростью
- **Минимальные накладные расходы**: легкий протокол с минимальными заголовками
- **Отсутствие управления перегрузкой**: не учитывает состояние сети

Подходящие случаи использования

- Потоковое видео (допустимая потеря пакетов)
- Онлайн-игры (приложения, чувствительные к задержкам)
- DNS-запросы (небольшие объемы данных, простая логика повторной попытки)
- Прямые трансляции
- Данные датчиков IoT (наиболее важные последние значения)

HTTP: основа Интернета

HTTP прошел несколько серьезных ревизий, каждая из которых была направлена на решение конкретных проблем производительности и совместимости. Понимание этих различий имеет решающее значение для принятия обоснованных решений о производительности и совместимости системы.

HTTP/1.1 остается наиболее широко распространенной версией в Интернете, хотя использование HTTP/2 продолжает расти, особенно среди веб-сайтов с высоким трафиком.

HTTP/1.0 — базовый протокол

HTTP/1.0, появившийся в середине 1990-х годов, заложил основу для веб-коммуникаций, но имел значительные ограничения:

- Каждый запрос требовал нового TCP-соединения
- Отсутствие механизмов повторного использования соединений
- Подходит для простых веб-страниц, но не подходит для сложных приложений

HTTP/1.1 — производственный стандарт

Выпущенный в 1997 году, HTTP/1.1 внес значительные улучшения, которые по-прежнему широко используются:

Постоянные соединения

одно соединение обрабатывает несколько запросов

Конвейеризация

несколько запросов могут быть отправлены без ожидания ответов (ограниченная поддержка браузерами)

Заголовки хоста

виртуальный хостинг позволяет размещать несколько веб-сайтов на одном IP-адресе

Кодирование с разбиением на фрагменты

передача данных без заранее определенной длины содержимого

HTTP/1.1 служит основой для большинства современных веб-приложений.

HTTP/2 — повышение производительности

Введен в 2015 году для устранения ограничений производительности HTTP/1.1:

01
10

Двоичный протокол

более эффективен, чем текстовый HTTP/1.1

Мультиплексирование

одновременная передача запросов по одному соединению

Серверный push

проактивная доставка ресурсов до запроса клиента

Сжатие заголовков

уменьшение накладных расходов на повторяющиеся заголовки

Крупные платформы, включая Google, Facebook и Netflix, широко внедрили HTTP/2.

Мультиплексирование HTTP/2: решение проблемы блокировки начала очереди

Ограничение HTTP/1.1: HTTP/1.1 обрабатывает запросы последовательно даже при постоянных соединениях. Длительные запросы (например, загрузка больших файлов) блокируют все последующие запросы по тому же соединению, создавая блокировку начала очереди.

> **Соединение HTTP/1.1:**
> Запрос 1 (медленный) ----[ожидание 5 секунд]----> Ответ 1
> Запрос 2 (быстрый) ----[блокирован, ожидание]----> Ответ 2
> Запрос 3 (быстрый) ----[блокирован, ожидание]----> Ответ 3

Решение HTTP/2: HTTP/2 реализует мультиплексирование, позволяя одновременную передачу запросов по одиночным соединениям с доставкой ответов в любом порядке:

> **HTTP/2 Connection:**
> Request 1 (slow) ----[processing]----> Response 1 (arrives last)
> Request 2 (fast) ----> Response 2 (arrives first)
> Запрос 3 (быстрый) ----> Ответ 3 (приходит вторым)

Преимущества производительности

- **Ускоренная загрузка страниц**: параллельная загрузка CSS, JavaScript и изображений
- **Улучшенное использование ресурсов**: устраняет необходимость в нескольких соединениях
- **Улучшенная производительность на мобильных устройствах**: критически важно для мобильных сетей с высокой задержкой

Техническое резюме: Мультиплексирование HTTP/2 позволяет одновременно передавать запросы по одному TCP-соединению, устраняя блокировку начала очереди, присущую последовательной обработке HTTP/1.1. Это значительно сокращает время загрузки страниц, особенно для сайтов с большим количеством небольших ресурсов, таких как изображения, CSS и файлы JavaScript.

HTTP/3 — протокол следующего поколения

Построен на протоколе QUIC (транспортный уровень на основе UDP):

Сокращение времени установки соединения

минимальная задержка первоначального установления соединения

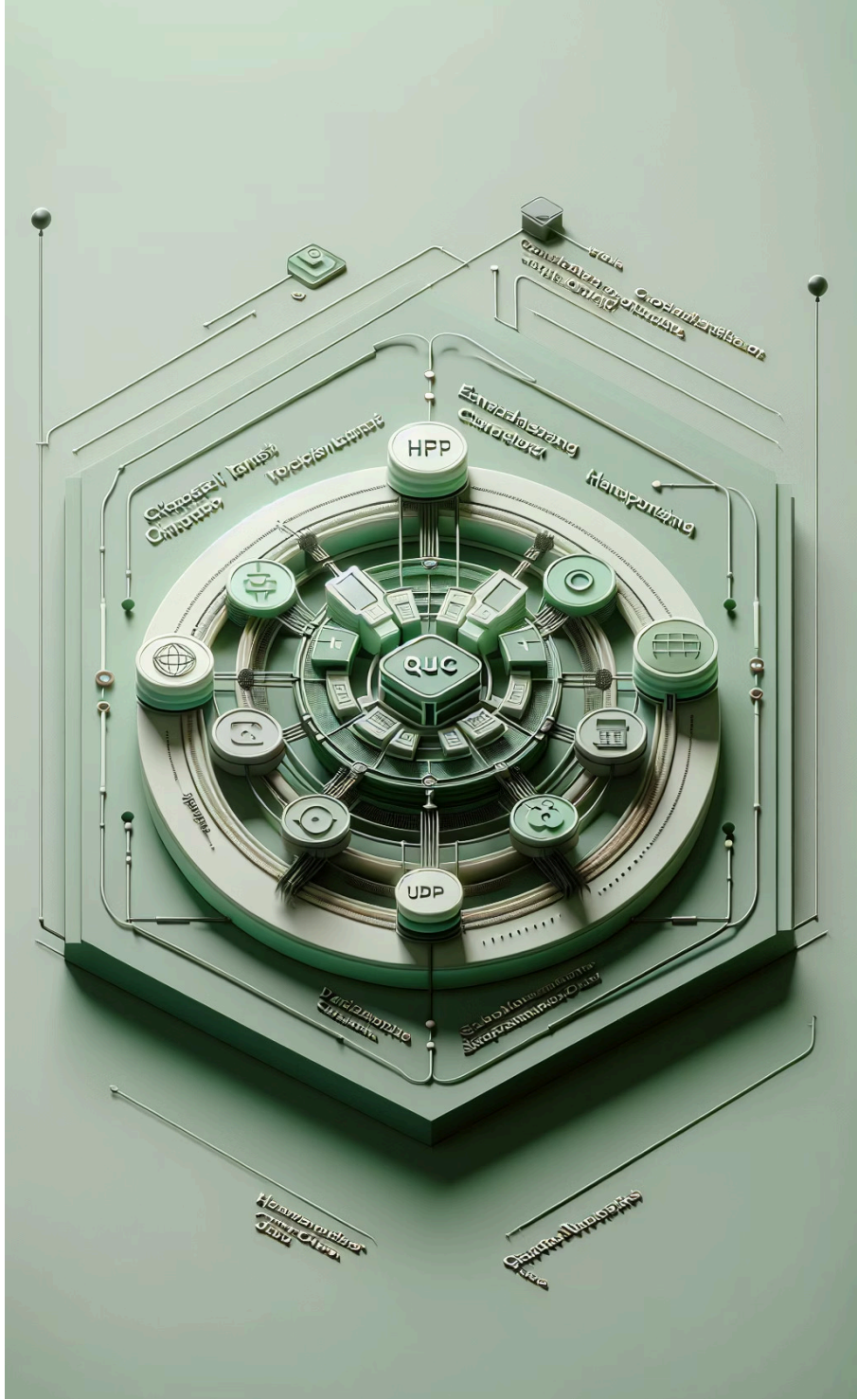
Повышенная отказоустойчивость сети

улучшенная обработка потери пакетов

Миграция соединений

поддержание соединений при переходе между сетями (с WiFi на сотовую)

HTTP/3 продолжает распространяться, демонстрируя особые преимущества для мобильных приложений.



Защита HTTP-коммуникаций

Индикатор безопасности браузера представляет собой сложный криптографический процесс, обеспечивающий безопасную коммуникацию:

SSL vs TLS — эволюция протоколов

SSL (Secure Sockets Layer)

исходный протокол безопасности, устаревший из-за уязвимостей в безопасности

TLS (Transport Layer Security)

современный безопасный протокол, заменивший SSL, хотя часто все еще называемый «SSL»

Современные ссылки на «SSL-сертификаты» на самом деле означают TLS-сертификаты и протоколы.

Как работает TLS

Рукопожатие TLS устанавливает безопасную связь посредством следующего процесса:

01

Client Hello

клиент объявляет поддерживаемые алгоритмы шифрования и наборы шифров

02

Server Hello

сервер выбирает метод шифрования и предоставляет цифровой сертификат

03

Проверка сертификата

Клиент проверяет подлинность сертификата сервера и цепочку доверия

04

Обмен ключами

обе стороны устанавливают общие ключи шифрования с использованием согласованных алгоритмов

05

Начало безопасного сеанса

вся последующая связь использует установленное шифрование

Взаимный TLS (mTLS) — двунаправленная аутентификация

В отличие от стандартного TLS, где только серверы аутентифицируются перед клиентами, mTLS требует взаимной аутентификации:

- Сервер проверяет действительность и идентичность сертификата клиента
- Клиент проверяет действительность и идентичность сертификата сервера
- Обычно реализуется в архитектурах микросервисов, требующих доверия между службами
- Требует комплексного управления жизненным циклом сертификатов и процедур ротации

Когда использовать mTLS, а когда обычный TLS

Выбор между mTLS и стандартным TLS представляет собой важное архитектурное решение, имеющее значительные последствия для безопасности и эксплуатации:

Обычный TLS — стандартная реализация

Подходит для:

Публичные API

службы, к которым обращаются внешние клиенты (мобильные приложения, веб-браузеры, сторонние интеграции)

Неконтролируемые клиентские среды

сценарии, в которых доверие клиента не может быть заранее определено

Простота эксплуатации

ситуации, требующие минимальных затрат на управление сертификатами

Оптимизация производительности

приложения, в которых критически важно сократить задержку соединения

Сценарии реализации:

- REST API, поддерживающие мобильные приложения
- Общедоступные веб-сайты и веб-приложения
- SaaS-платформы, обслуживающие внешних клиентов
- Балансировщики нагрузки, обеспечивающие терминацию SSL

mTLS — реализация с нулевым доверием

Подходит для:

Межсервисная коммуникация

архитектуры микросервисов, требующие взаимной проверки

Среды с нулевым доверием

сети, в которых доверие не может быть предположено на любом уровне

Соблюдение нормативных требований

отрасли, требующие взаимной аутентификации (финансовые услуги, здравоохранение)

Среды с высоким уровнем безопасности

государственные системы и среды с конфиденциальными данными

Контролируемые конечные точки

среды, в которых управляются обе конечные точки связи

Сценарии реализации:

- Микросервисы в кластерах Kubernetes
- Подключения к базам данных с серверов приложений
- Шлюз API для связи с бэкэнд-сервисами
- Реализации service mesh (Istio, Linkerd)

Компромиссы

Преимущества mTLS

- **Повышенная безопасность**: двунаправленная проверка идентичности
- **Защита на уровне сети**: работает независимо от аутентификации на уровне приложений
- **Соответствие нормативным требованиям**: соответствует строгим отраслевым требованиям
- **Неотрицаемость**: криптографическое доказательство участников связи

Недостатки mTLS

- **Управление жизненным циклом сертификатов**: сложные процессы предоставления, ротации и отзыва
- **Эксплуатационные расходы**: повышенная сложность системы и количество точек отказа
- **Влияние на производительность**: дополнительные процедуры установления соединения и шаги проверки
- **Сложность устранения неполадок**: криптографические ошибки могут быть сложны для диагностики

Схема принятия решений: Обычный TLS подходит для общедоступных сервисов с неконтролируемыми клиентами, где приоритетом является простота эксплуатации. mTLS подходит для межсервисного взаимодействия в архитектурах микросервисов, требующих надежной взаимной аутентификации и управляемого жизненного цикла сертификатов. Основной компромисс заключается в балансе между требованиями безопасности и сложностью эксплуатации.

Модель зрелости REST

Модель зрелости Леонарда Ричардсона демонстрирует, что большинство API, претендующих на соответствие REST, не полностью реализуют принципы REST:

Уровень 0 — RPC через HTTP

```
POST /api/endpoint
{
  "action": "getUserById",
  "userId": 123
}
```

Представляет удаленные вызовы процедур, реализованные через HTTP без принципов REST.

Уровень 1 — несколько ресурсов

```
POST /api/users/123/getProfile
POST /api/orders/456/cancel
```

Вводятся URL-адреса на основе ресурсов, но сохраняются операции в стиле RPC.

Уровень 2 — HTTP-глаголы и коды статуса

```
GET /api/users/123 (возвращает 200 OK)
DELETE /api/orders/456 (возвращает 204 No Content)
```

Большинство современных «RESTful» API работают на этом уровне, используя соответствующие методы HTTP и значимые коды статуса.

Уровень 3 — гипермедиа (HATEOAS)

```
{
  "userId": 123,
  "name": "John Doe",
  "_links": {
    "self": { "href": "/api/users/123" },
    "orders": { "href": "/api/users/123/orders" },
    "edit": { "href": "/api/users/123", "method": "PUT" }
  }
}
```

API предоставляет клиентам обнаруживаемые действия и возможности навигации. Немногие производственные API реализуют этот полный уровень REST.

Резюме

Сетевые протоколы служат фундаментальными строительными блоками архитектуры распределенных систем. TCP обеспечивает гарантии надежности, необходимые для целостности данных, а UDP предлагает преимущества производительности для приложений, чувствительных к задержкам. HTTP/2 устраняет ограничения производительности за счет мультиплексирования и сжатия. Шифрование TLS должно быть реализовано повсеместно, а mTLS подходит для внутренней коммуникации служб. Большинство практических REST API эффективно функционируют на уровне зрелости 2 по шкале Ричардсона.