Network: Часть 7. Прикладной уровень

24.10.2026 09:58 edited 24.10.2026 10:48

Эпоха смыслов

В декабре 1990 года в Европейской организации по ядерным исследованиям (CERN) под Женевой заработал компьютер NeXT с приклеенной от руки табличкой: «Эта машина — сервер. НЕ ВЫКЛЮЧАТЬ!». Его владелец Тим Бернерс-Ли разместил на нем первый в мире веб-сайт — одну страницу, объяснявшую, что такое World Wide Web. Первые месяцы Веб существовал внутри стен института: анонс в телеконференции alt.hypertext Бернерс-Ли опубликовал только 6 августа 1991 года, а к концу 1993-го в мире насчитывалось около шестисот сайтов. Тридцать пять лет спустя прикладной уровень — то, ради чего, собственно, и строились все остальные, — выглядит так. По состоянию на октябрь 2026 года в мире зарегистрировано 401,6 миллиона доменных имен (Verisign, второй квартал 2026-го), ежедневно отправляется и принимается около 390 миллиардов электронных писем (Radicati Group), к сети подключены примерно 6 миллиардов человек — 74% населения планеты (МСЭ, 2025), свыше 95% загрузок страниц в браузере Chrome идут по шифрованному HTTPS, медианная веб-страница весит 2,4 мегабайта и требует около сотни запросов (HTTP Archive, 2025), на видео приходится около двух третей всего мирового трафика, а HTTP/3 — протокол, стандартизованный лишь в 2022 году, — поддерживает уже 41% веб-сайтов (W3Techs, октябрь 2026-го).

Парадокс прикладного уровня в том, что это единственный уровень сетевой модели, который пользователь видит своими глазами. Все, что ниже, — сигналы в проводе, кадры Ethernet, пакеты IP, сегменты TCP, — существует исключительно ради того, чтобы протоколы этого уровня могли обменяться сообщениями. Физический уровень говорит о битах, транспортный — о надежной доставке байтов, а прикладной — о страницах, письмах, видео и вызовах функций. Именно здесь «доставка» превращается в «смысл»: сеть, которая умела лишь пересылать байты между адресами, становится почтой, библиотекой, кинотеатром, телефоном и торговой площадкой. При этом прикладной уровень — самая широкая часть песочных часов Интернета: узкая талия IP за сорок пять лет почти не изменилась, транспортный уровень дорабатывали, а не перестраивали, а сверху выросли сотни тысяч протоколов, значительная часть которых стандартизована в тысячах RFC.

Эта архитектура порождает ряд критических вопросов:

1. **Вопрос именования.** Люди и программы работают с именами вроде google.com, сеть — с числами вроде 142.250.74.14. Как распределенная база данных без единого центра, реплицирующая себя кэшированием, выдерживает триллионы запросов в сутки, не

становясь при этом единой точкой отказа, — и почему ответ на этот вопрос стоит искать в 1983 году, когда всю адресную книгу Интернета пришлось заменить деревом?

2. **Вопрос состояния.** HTTP намеренно ничего не помнит о клиенте между запросами. Из этой простоты выросла и масштабируемость Веба, и целая надстройка — cookies, сессии, токены, — которая из механизма брошенной корзины покупок превратилась в фундамент аутентификации и в главный инструмент слежки за пользователями.
3. **Вопрос эволюции.** От протокола с одним методом и без заголовков (1991) до HTTP/3 поверх QUIC (2022) Веб четырежды перестраивал собственный фундамент, сохраняя совместимость со страницами тридцатилетней давности. Почему HTTP/1.1 господствовал двадцать лет, почему мультиплексирование HTTP/2 упиралось в TCP, и почему новый транспорт пришлось прятать внутри старого UDP?
4. **Вопрос масштаба.** Один сервер физически неспособен обслужить миллионы зрителей одновременно. Кэширование, CDN, одноранговые сети — за тридцать лет Интернет превратился из схемы «клиент спрашивает сервер» в распределенную систему доставки контента, где большинство байтов отдается не автором, а копией копии, лежащей в двух шагах от пользователя.
5. **Вопрос времени.** Видео требует мегабайтов в секунду, а телефонный разговор — задержки меньше 150 миллисекунд, для первого приемлема буферизация, для второго — смерть. Как одни и те же приложения уживаются с каналами самого разного качества — и почему битрейт выбирает клиент, не спрашивая сеть?
6. **Вопрос доверия.** Прикладные протоколы проектировались в эпоху всеобщего доверия: SMTP не проверяет отправителя, DNS не подписывает ответы, HTTP ничего не шифрует. Сорок лет индустрия латает эти дыры на ходу — SPF, DKIM и DMARC для почты, DNSSEC и DoH для имен, TLS для всего остального, — и латает до сих пор не до конца.
7. **Вопрос приватности.** Прикладной уровень стал главным театром борьбы за приватность: сторонние cookies отслеживают пользователя между сайтами, DNS-запросы раскрывают всю историю просмотров провайдеру, а зашифрованный HTTPS скрывает содержимое, но не метаданные. Битва браузеров, регуляторов и рекламного бизнеса переписывает протоколы прямо сейчас.

Дальше мы пройдем весь прикладной уровень: от первых приложений ARPANET и архитектур клиент-сервер и P2P — через DNS как систему и как протокол, HTTP со всеми его версиями, cookies, кэшированием и прокси — к загрузке страницы, потоковому видео, CDN, электронной почте с ее store-and-forward-архитектурой, современным API (WebSocket, REST, GraphQL, gRPC), ветеранам вроде FTP и Gopher, рекламе и трекингу, безопасности. В финале проследим путь одного запроса — от адресной строки браузера до пикселей на экране и письма-подтверждения в почтовом ящике. Материал опирается на стандарты IETF и W3C, исторические документы и статистику 2025–2026 годов.

Рождение прикладного уровня

«LO»: приложения появились раньше, чем для них построили сеть

Первым сообщением в истории Интернета было сообщение прикладного уровня. Вечером 29 октября 1969 года студент UCLA Чарли Клайн, сидя за терминалом, попытался передать слово LOGIN на компьютер Стэнфордского исследовательского института: сеть успела доставить буквы L и O, после чего удаленная система рухнула. Историческое «LO» примечательно не курьезом, а сутью: уже в первый день сеть понадобилась не для абстрактной «передачи данных», а для конкретного приложения — удаленного входа на чужой компьютер. Telnet-подобный доступ, затем пересылка файлов, затем почта — вот три кита, на которых ARPANET стоял еще до того, как обзавелся привычным стеком протоколов.

Хронология здесь важнее, чем кажется. Спецификация протокола передачи файлов FTP появилась в RFC 114 16 апреля 1971 года — за три с лишним года до статьи Серфа и Кана о TCP и за двенадцать лет до того, как TCP/IP стал стандартом сети. Первые приложения работали поверх NCP (Network Control Program) и не знали ни IP-адресов, ни сегментов. Слои сетевой модели не строились снизу вверх, как в учебниках: сначала сообщество научилось тому, зачем нужна сеть, — и лишь потом построило под эти задачи доставку. Прикладной уровень — старейший в Интернете, и он же определяет облик всех остальных.

Электронная почта: первое киллер-приложение

В конце 1971 года Рэй Томлинсон, инженер компании BBN, строившей процессоры ARPANET, догадался скрестить две существовавшие программы: локальную SNDMSG, оставлявшую записки в почтовом ящике на том же компьютере, и CPYNET, пересылавшую файлы по сети. Получилась первая сетевая почта — сообщение, способное перелететь с машины на машину. Символ @ Томлинсон выбрал из соображений редкости: он не встречался в именах пользователей и разделял имя получателя от имени хоста — ray@bbn-tenexa. Содержание первых писем автор не запомнил, по его словам, это была бессмыслица вроде «QWERTYUIOP».

Зато эффект запомнили все. К 1973 году, по оценкам современников, на электронную почту приходилось около трех четвертей всего трафика ARPANET: сеть, задуманная для разделения вычислительных ресурсов, оказалась в первую очередь средством общения. Почта стала первым киллер-приложением (killer app) — сервисом, ради которого люди и организации подключались к сети, — и задала шаблон на полвека вперед: приложения тянут инфраструктуру, а не наоборот. Уже в мае 1978 года по ARPANET разлетелся первый спам — рекламная рассылка DEC, о которой речь впереди, в почтовой главе.

Клиент-сервер: модель из эпохи мейнфреймов

Архитектуру сетевых приложений предопределили вычислительные машины 1960–1970-х. Мейнфрейм с пакетной обработкой и «глупые» терминалы, печатающие символы и показывающие результат, — это и есть прообраз клиент-серверной модели: асимметричной, где одна сторона (сервер) постоянно работает, хранит данные и обслуживает запросы, а другая (клиент) эти запросы порождает. Разделение труда, придуманное для разделения дорогого процессорного времени между сотрудниками одного института, перенесли на разделение труда между машинами в сети — и оно оказалось настолько удачным, что на нем до сих пор стоит почти весь Интернет: HTTP, DNS, SMTP, IMAP.

Стандарты: фабрика RFC

Прикладной уровень стандартизуется той же фабрикой RFC, что и остальные, — но производит на порядок больше продукции: большинство из почти десяти тысяч документов IETF описывает именно приложения. Старейшая действующая спецификация уровня — SMTP, протокол передачи почты, выпущенный Джоном Постелом в августе 1982 года (RFC 821 и RFC 822, современная редакция — RFC 5321, 2008): ему сорок пять лет, и по нему до сих пор ходит большинство мировых писем. Срок жизни прикладного протокола измеряется десятилетиями — зашитые в спецификацию решения переживают авторов и эпоху. Цена долголетия — консерватизм: ошибки проектирования SMTP и DNS исправляют не заменой протоколов, а надстройками поверх них.

Задачи и архитектуры прикладного уровня

Где кончается доставка и начинается смысл

Границу уровня проще всего увидеть на цепочке инкапсуляции — той же, что разбиралась для транспорта и сети, с еще одним слоем сверху:

```
сообщение приложения → TCP-сегмент или UDP-датаграмма → IP-пакет → кадр Ethernet → биты в среде
```

Каждый уровень добавляет к данным свой заголовок и решает свою задачу. Канальный уровень отвечает за передачу кадра между соседями, сетевой — за доставку пакета между хостами через множество сетей, транспортный — за доставку потока байтов между процессами. Прикладной уровень создает само сообщение: его формат, его смысл и правила разговора. HTTP не думает о маршрутах и потерях — этим заняты IP и TCP, HTTP думает о том, какой ресурс запрашивают, в каком представлении его вернуть и что означает полученный ответ. Нижние уровни — это доставка, прикладной — смысл общения. Отсюда и практическое следствие, знакомое каждому, кто открывал браузерные инструменты разработчика: проблемы доставки лечатся настройкой сети, проблемы смысла — только исправлением приложений и протоколов.

Четыре задачи уровня

Из этого положения следуют четыре задачи, которые решает любой прикладной протокол.

- **Формат данных.** Как сообщение выглядит на проводе: HTTP — человекочитаемый текст с заголовками, DNS — плотный бинарный формат с указателями сжатия, почта — текст с MIME-расширениями, gRPC — бинарные сообщения protobuf. Формат определяет эффективность, отлаживаемость и совместимость — выбор между ними вечный инженерный торг.
- **Логика взаимодействия.** Кто начинает разговор, кто отвечает, сколько шагов в диалоге. Модель «запрос — ответ» (HTTP) — лишь одна из многих: многошаговые диалоги (SMTP, IMAP), постоянные двусторонние каналы (WebSocket), асинхронная передача с накоплением (почта), равный обмен без выделенных серверов (BitTorrent).
- **Именование.** Люди работают с осмысленными именами — google.com, user@example.com, — а сеть оперирует числами. Индирекция — разрешение имени в адрес через отдельный слой — один из самых мощных инструментов информатики, и в Интернете его реализует DNS: без него не было бы ни переносимых ссылок, ни миграций между хостингами, ни балансировки нагрузки.
- **Семантика.** Что сообщение означает и что позволено с ним делать: «GET /index.html» — «дай страницу», код 404 — «не найдено». Семантика включает контракт о повторимости: безопасные операции (чтение) не меняют состояние сервера, идемпотентные (удаление, PUT) дают одинаковый результат при повторе — на этих свойствах держатся кэширование, ретраи и вся надежность веба поверх ненадежной сети.

Клиент-серверная архитектура

Основная модель наследует мейнфреймам: клиент инициирует соединение и задает вопросы, сервер слушает на известном порту, отвечает и хранит состояние. Сервер — постоянно работающий процесс с публично адресуемой машиной (или кластером за балансировщиком), клиент — любое устройство, от телефона до датчика. Асимметрия проста в управлении, но упирается в производительность сервера. Отсюда три пути масштабирования: вертикальный (мощное железо), горизонтальный (кластеры и балансировщики) и географический (копии контента ближе к пользователям — CDN).

Одноранговая архитектура: равенство, которое не победило

Альтернатива клиент-серверу — одноранговая сеть (peer-to-peer, P2P), где каждый узел одновременно и клиент, и сервер, а центра нет вовсе. Главный козырь — масштабируемость: мощность системы растет вместе с числом участников, ведь каждый новый потребитель приносит и ресурсы. История P2P — три поколения. Napster (Шон Фаннинг, июнь 1999-го) был гибридом: файлы ходили напрямую между пользователями, но центральный сервер хранил индекс, за два года сервис собрал десятки миллионов пользователей и погиб в судах звукозаписывающих компаний (июль 2001-го). Gnutella (март 2000-го) убрала центр совсем — и утонула в собственной неэффективности: запросы

разливались лавиной, каждый узел ретранслировал чужие поиски. BitTorrent (Брэм Коэн, 2001) решил обе проблемы: куски файлов, обмен по принципу «ты — мне, я — тебе» (tit-for-tat), приоритет редких кусков (rarest first), координация трекером, позже замененным распределенной хеш-таблицей. Skype (2003) построил на суперузлах глобальную телефонию.

Но у равенства обнаружились системные минусы: большинство пользователей за NAT не принимает входящие соединения, узлы приходят и уходят (churn), предсказуемость обслуживания нулевая, монетизация невозможна, а правообладатели и провайдеры ведут с раздачами войну. Итог: доля P2P в мировом трафике упала с примерно шестидесяти процентов в середине 2000-х до единиц процентов сегодня. Победила гибридная модель: управление — через серверы, данные — откуда угодно, включая соседние узлы, так устроены обновления игр, доставка дистрибутивов, блокчейн-сети. Чистое P2P осталось нишей — зато доказало, что масштабируемая доставка контента не обязана быть централизованной, и подготовило почву для CDN.

Протокол как API

И главное обобщение уровня: прикладной протокол — это сетевой API, контракт не между машинами, а между программами. URI (RFC 3986, 2005) задает универсальный синтаксис имен — схема://хост/путь?запрос, — а номер порта указывает, какому приложению на хосте адресовано сообщение. Все остальное — соглашение двух сторон: формат сообщений, допустимые действия, возможные ответы. В этом источник и скорости инноваций (любой может изобрести протокол и опубликовать RFC, не спрашивая ничьего разрешения), и проблем (никто не проверит новый протокол на безопасность). Прикладной уровень — самая широкая часть песочных часов: снизу узкая талия IP, сверху сотни тысяч приложений, и граница между «протоколом» и «приложением» условна: REST, WebSocket и gRPC — одновременно соглашения о сообщениях и способы организации программных систем.

DNS: имена вместо чисел

Кризис одного файла и ответ Мокапетриса

DNS легко принять за «телефонную книгу Интернета» — таблицу «имя → адрес». Это упрощение упускает суть: DNS — распределенная, иерархическая, кэширующая система именования общего назначения, и до 1983 года никакой системы не было вовсе. Был файл. Хост-таблицу HOSTS.TXT вела Служба сетевого информационного центра (SRI-NIC) под руководством Элизабет Файнлер: администратор каждого нового узла ARPANET слал в центр письмо с именем и адресом своей машины, сотрудники NIC вручную правили файл и дважды в неделю рассылали его всем хостам. Схема работала, пока хостов были десятки. В начале 1980-х число машин стало удваиваться примерно каждые девяносто дней, рассылка съедала заметную долю каналов, имена начали конфликтовать (несколько

организаций независимо выбирали одинаковые — вроде «fubar»), ручное администрирование перестало успевать за ростом. Стало ясно: единая точка ведения имен не масштабируется.

Требований было два: глобально уникальные имена и децентрализованное администрирование. Решение спроектировал Пол Мокапетрис из ISI: в ноябре 1983 года вышли RFC 882 и RFC 883, а сам Мокапетрис написал первую реализацию — JEEVES. В 1984–1985 годах четверо студентов Калифорнийского университета в Беркли (Дуглас Терри, Марк Пейнтер, Дэвид Риггл и Чжоу Суннянь) создали BIND — первую широко разошедшуюся реализацию, которую сегодня поддерживает консорциум ISC. В 1987-м спецификацию переработали в действующие RFC 1034 и RFC 1035. Ключевых идей было четыре: единое дерево имен, делегирование ответственности за поддеревья, кэширование с временем жизни и общий формат данных — DNS хранит не только адреса, но и произвольные типизированные записи.

Дерево и делегирование

Пространство имен организовано как перевернутое дерево. Корень — пустая метка, от него отходят домены верхнего уровня (TLD): национальные (.ru, .de, .uk), общего назначения (.com, .org, .net) и «новые», появившиеся после программы ICANN 2012 года (.app, .dev, .museum — всего около полутора тысяч TLD). Ниже — домены второго уровня (google.com), третьего (mail.google.com) и так далее, полное имя читается справа налево, от общего к частному, а точка в конце (google.com.) — тот самый корень, который браузеры из вежливости прячут.

Управляют деревом так же иерархично. Корневую зону поддерживает IANA (функция ICANN), 1 октября 2016 года управление ею перешло от правительства США к глобальному многостороннему сообществу. Корневые серверы — тринадцать логических имен от a.root-servers.net до m.root-servers.net, за которыми стоят двенадцать организаций (Verisign, ISC, RIPE NCC, ICANN, WIDE и другие). Тринадцать — ограничение протокола 1980-х (в UDP-ответ помещались адреса ровно тринадцати серверов), а не физических машин: благодаря anycast те же имена обслуживают более двух тысяч экземпляров по всему миру (root-servers.org, 2026). Записи каждого домена хранят его авторитетные серверы, делегирование — запись NS у родителя — передает ответственность за поддерево владельцу домена. Так дерево масштабируется: никто не хранит все записи, каждый хранит свои и ссылки на чужие. По данным Verisign на 30 июня 2026 года, в зонах .com и .net зарегистрировано 179,1 миллиона имен (в .com — 166,6 миллиона), а всего во всех TLD — 401,6 миллиона.

Путь запроса: рекурсивный и итеративный

Когда браузер спрашивает адрес example.com, путь выглядит так. Сначала проверяются локальные кэши: браузера, операционной системы (в Windows — служба DNS Client, в Linux — systemd-resolved), файл /etc/hosts. Если ответа нет, запрос уходит рекурсивному резолверу — DNS-серверу провайдера или публичному (Google Public DNS 8.8.8.8, запущен

в декабре 2009-го, Cloudflare 1.1.1.1 — с апреля 2018-го). Клиент ставит флаг RD (Recursion Desired) — «сделай всю работу за меня», резолвер отвечает флагом RA, если рекурсию поддерживает.

Дальше резолвер обходит дерево итеративно — каждый сервер возвращает не ответ, а «подсказку» (referral). К корневому — «где зона .com?» — тот отвечает списком NS зоны .com и их адресами (glue-записи). К серверам .com — «где example.com?» — ответ: NS домена. К авторитетным серверам example.com — «каков адрес?» — ответ: A-запись с TTL. Резолвер возвращает ответ клиенту и кэширует его на время TTL, кэшируются и подсказки. Именно кэширование делает систему живой: до корневых серверов доходят доли процента обращений — иначе тринадцать имен не выдержали бы и дня. Масштаб: только Google Public DNS отвечает более чем на триллион запросов в сутки.

Два типа запросов стоит различать твердо: рекурсивный («найди и дай ответ целиком») идет от клиента к его резолверу, итеративный («скажи, кого спросить дальше») — от резолвера к корневым, TLD и авторитетным серверам. Клиент почти никогда не ходит по дереву сам, резолвер почти никогда не требует рекурсии от других.

Типы записей: DNS — не только адреса

DNS — типизированная база данных. Основные типы: A (адрес IPv4) и AAAA (адрес IPv6) — имя в число, CNAME — «псевдоним» (www.example.com → example.com), заставляющий резолвер повторить запрос к другому имени, NS — авторитетные серверы зоны, SOA — параметры зоны (серийный номер, тайминги, отрицательный TTL), MX — почтовые серверы домена с приоритетами (RFC 974, 1986) — без MX не работает электронная почта, TXT — произвольный текст, который практика превратила в служебный канал: там лежат политики SPF, ключи DKIM, подтверждения владения доменом, параметры DMARC, PTR — обратное соответствие «адрес → имя», SRV (RFC 2782) — адрес и порт сервиса (SIP, XMPP). Новейшее поколение — записи SVCB и HTTPS (RFC 9460): по ним браузер до соединения узнает, поддерживает ли сервер HTTP/3, — DNS все глубже встраивается в настройку приложений.

TTL и кэширование: скорость ценой устаревания

Каждая запись отдается с временем жизни (TTL): резолверы и клиенты вправе держать ее в кэше не дольше этого срока. TTL — главный рычаг управления системой: большой (часы и сутки) снижает нагрузку на авторитетные серверы, маленький (секунды и минуты) позволяет быстро менять адреса — ценой роста трафика. Отсюда практические следствия: смена IP при переезде «распространяется» не мгновенно, а в течение TTL — плюс кэши, которые TTL уважать не обязаны (исторически многие резолверы держали записи дольше предписанного), перед миграцией TTL заранее снижают до минут, а диагноз «у меня работает, у пользователя нет» в половине случаев означает расхождение кэшей. Существует и отрицательное кэширование (RFC 2308): ответы «такого имени нет» (NXDOMAIN) тоже запоминаются — иначе опечатки грузили бы авторитетные серверы повторно.

DNS за пределами именования: глобальный план управления

Поскольку DNS стоит перед каждым соединением, он давно стал планом управления (control plane) Интернета. Один домен возвращает несколько адресов, и порядок ответа можно менять — простейшая балансировка нагрузки (round robin). Авторитетному серверу виден адрес резолвера — а часто и подсеть клиента, — поэтому разным пользователям он отвечает по-разному: ближайшим узлом CDN (геомаршрутизация), наименее нагруженным дата-центром, запасным адресом при аварии (failover с проверками здоровья). Системы класса GSLB (global server load balancing) превращают DNS в диспетчера планетарного масштаба — именно так Cloudflare, Akamai и Amazon направляют запросы в нужные точки. Обратная сторона — грубость инструмента: DNS не видит реальной загрузки пользователя, а кэширование делает переключения медленными, поэтому DNS-диспетчеризацию дополняют anycast (один адрес из многих точек — выбор делает сама маршрутизация) и балансировкой внутри дата-центров. Есть и обратное разрешение: зоны in-addr.arpa для IPv4 и ip6.arpa для IPv6, где адреса записаны задом наперед (14.74.250.142.in-addr.arpa). PTR-записи используют логирование, почтовые фильтры (отсутствие обратного имени — дурной знак для отправителя) и инструменты безопасности.

Интернет технически способен работать без DNS — curl с явным адресом соединится нормально. Но имя — единственная точка входа: нет имени — нет сайта, нет почты (MX!), нет API. DNS — критическая инфраструктура в буквальном смысле: его отказ делает ресурсы «несуществующими», даже когда серверы живы. Эта роль — при отсутствии встроенной аутентификации в оригинальном протоколе — сделала DNS постоянной мишенью.

DNS как протокол: на проводе

UDP, TCP и порт 53

DNS ходит по порту 53 — исторически через UDP, в современности все чаще через TCP и шифрованные надстройки. Выбор UDP очевиден: типичный запрос — несколько десятков байт, ответ — сотни, датаграмма не требует рукопожатия и экономит RTT. TCP подключается в трех случаях. Первый: ответ не помещается в датаграмму — сервер ставит в заголовке флаг TC (truncated), и клиент повторяет запрос по TCP. Исторический лимит UDP-ответа — 512 байт из RFC 1035, расширение EDNS0 (RFC 2671, 1999, действующая редакция RFC 6891, 2013) позволило объявлять больший буфер — обычно 1232 или 4096 байт. Второй: пересылка зон (zone transfer, AXFR/IXFR) с первичного сервера на вторичные. Третий: DNSSEC — подписи раздувают сообщения в разы. Отдельная дата — 1 февраля 2019 года, «DNS Flag Day»: крупнейшие операторы (Google, Cloudflare, Cisco, Akamai, ISC) одновременно перестали обходить ошибки старых реализаций, не понимающих EDNS, — редчайший случай принудительной модернизации децентрализованного протокола.

Пять секций сообщения

Запрос и ответ DNS устроены одинаково — пять секций подряд. HEADER: идентификатор запроса (16 бит) для сопоставления ответов, флаги — QR (запрос или ответ), RD/RA (рекурсия желательна/доступна), AA (ответ авторитетный), TC (усечено), код результата RCODE (NOERROR, NXDOMAIN — «имя не существует», SERVFAIL), счетчики четырех оставшихся секций. QUESTION: что спрашиваем — имя (QNAME), тип записи (QTYPE: A, AAAA, MX...) и класс (QCLASS, почти всегда IN — Интернет). ANSWER: найденные записи, каждая с именем, типом, классом, TTL и данными. AUTHORITY: NS-записи зоны — подсказка «кого спросить дальше». ADDITIONAL: дополнительные данные, экономящие запросы, — например, адреса тех самых NS-серверов (glue-записи).

Имена кодируются последовательностью меток («www», «example», «com», корень) с длиной перед каждой и сжимаются двухбайтовыми указателями на первые вхождения. Сжатие 1983 года позволило уложить ответы с десятками записей в одну датаграмму — и оно же стало источником уязвимостей (циклические указатели в злых ответах).

Оптимизации, заложенные в дизайн

Три механизма делают систему быстрой: кэширование на всех уровнях, благодаря которому до корневых серверов доходит ничтожая доля запросов, секция ADDITIONAL с glue-записями — адреса NS-серверов приходят вместе с подсказкой, не тратя лишний запрос, отрицательное кэширование NXDOMAIN. У последнего есть темная сторона: злоумышленники генерируют миллионы случайных поддоменов, чтобы нагрузить авторитетные серверы, — атаки класса «water torture».

Уязвимости: спуфинг, отравление кэша и атака Каминского

Оригинальный DNS не аутентифицирует ничего: ответ признается настоящим, если совпали идентификатор запроса, адрес и порт отправителя — а все три значения в реализациях 1980-х были предсказуемы. Отсюда два приема. Спуфинг (spoofing) — подсунуть жертве поддельный ответ быстрее настоящего, отравление кэша (cache poisoning) — тот же ответ, скормленный резолверу: одна удачная подделка перенаправляет к атакующему всех его клиентов на весь TTL, а через подделку делегирований — и дольше (bailiwick-атаки). В июле 2008 года Дэн Каминский показал: слабость глубже — перебором идентификаторов кэш отравляется, даже не угадывая, какое имя запрашивают. 8 июля 2008-го вышел беспрецедентный согласованный патч крупнейших вендоров, меры закрепил RFC 5452 (2009): случайные исходные порты, стойкие идентификаторы, а также трюк Пола Викси «0x20» — случайное чередование регистра букв в имени как дополнительный бит энтропии.

DNSSEC: подписи вместо доверия

Радикальное решение — криптография. DNSSEC (RFC 4033–4035, март 2005 года) подписывает зоны: подписи RRSIG, ключи DNSKEY, хеши отсутствующих имен NSEC/NSEC3, доверие — цепочкой от корневого ключа вниз через DS-записи делегирования. Проверивший подпись резолвер знает: ответ действительно от владельца зоны и не

изменен в пути. Корневая зона подписана 15 июля 2010 года, .com — 1 апреля 2011-го, сегодня подписаны подавляющее большинство TLD. Но «последняя миля» буксует: подписанных доменов в .com — считанные проценты (Verisign, 2023), валидирующих резолверов — порядка трети мирового трафика. Причины типичны: подписи раздувают ответы, управление ключами операционно сложно, а видимой пользы для пользователя нет — пока работает HTTPS, подделанный DNS-ответ упрется в невалидный сертификат. И главное: DNSSEC защищает целостность и происхождение данных, но не конфиденциальность.

DNS как цель атак

6 февраля 2007 года скоординированная DDoS-атака обрушила запросы на все тринадцать корневых серверов сразу и длилась около двенадцати часов, интернет в целом не заметил — спасли кэширование и anycast. В марте 2013 года спор антиспамеров Spamhaus с хостингом Cyberbunker вылился в атаку усиления (DNS amplification) мощностью около 300 Гбит/с — на тот момент крупнейшую в истории: крошечные запросы с отраженным адресом жертвы возвращались с открытых резолверов ответами в десятки раз большего размера. 21 октября 2016 года ботнет Mirai из зараженных IoT-устройств тремя волнами ударил по DNS-провайдеру Dyn: на восточном побережье США на часы легли Twitter, GitHub, Reddit, Spotify, PayPal. Урок Dyn: упал не сайт — упал провайдер имен, и тысячи независимых сервисов исчезли вместе с ним. Отсюда практики: несколько DNS-провайдеров на зону, короткие TTL для критичных записей, anycast, фильтрация поддельного трафика (BCP 38).

Приватность и зашифрованный DNS

Обычный DNS — последний крупный незашифрованный протокол массового Интернета: провайдер и любой наблюдатель по пути видит каждое запрошенное имя — фактически список всех посещенных сайтов. Ответ — шифрование. DoT (DNS over TLS, RFC 7858, 2016) — DNS в TLS-туннеле на порту 853, DoH (DNS over HTTPS, RFC 8484, 2018) — DNS-запросы как HTTPS-вызовы на порту 443, неотличимые для фильтров от веб-трафика, Oblivious DoH (RFC 9230, 2022) — схема, где прокси не видит запрос, а резолвер не видит клиента. Mozilla включила DoH по умолчанию для американских пользователей Firefox в феврале 2020-го, Chrome с конца 2019-го автоматически поднимает до DoH поддерживающие его резолверы. Развертывание спорно: корпоративные сети теряют фильтрацию, провайдеры — телеметрию, государства — рычаг цензуры, спор «это ломает родительский контроль» против «провайдер продает историю моих посещений» — политический, а не технический. Большая часть запросов от устройств до первого резолвера до сих пор ходит открытым UDP — но тренд очевиден: DNS движется к шифрованию по умолчанию.

HTTP: протокол, который стал Интернетом

«Расплывчато, но захватывающе»

В марте 1989 года Тим Бернерс-Ли, сотрудник CERN, подал руководству предложение «Information Management: A Proposal»: система гипертекстовых документов, связанных ссылками и доступных через сеть. Задача была прозаической: в институте терялась документация, знания уезжали вместе с уволившимися. На титульном листе сохранилась резолюция начальника Майка Сендалла: «Расплывчато, но захватывающе» (Vague, but exciting). За полтора года Бернерс-Ли написал браузер-редактор WorldWideWeb, первый сервер на своем NeXT и сам протокол HTTP, сайт info.cern.ch заработал 20 декабря 1990-го.

Первый HTTP, задним числом названный версией 0.9, состоял из одной команды: строка «GET /путь», в ответ — HTML-документ, после чего соединение закрывалось. Ни заголовков, ни кодов состояния, ни других методов. Дальнейшее определяли рынок и сообщество: в 1993 году Марк Андриссен и Эрик Бина из NCSA выпустили Mosaic — браузер, показывавший картинки внутри текста, к концу 1993 года в мире было около шестисот сайтов, к 1995-му — десятки тысяч. 30 апреля 1993 года CERN передал технологии Веба в общественное достояние — решение, без которого ни о каком Интернете в нынешнем виде речи бы не шло (оно же похоронило конкурента — Gopher). Андриссен ушел в бизнес: в 1994-м вместе с Джимом Кларком основал Netscape, IPO 9 августа 1995-го — акции взлетели вдвое в первый день торгов — дало старт эпохе доткомов. Microsoft ответила Internet Explorer, раздаваемым бесплатно вместе с Windows, — война браузеров закончилась антимонопольным делом против Microsoft (1998–2001) и открытием кода Netscape (январь 1998-го, из него выросли Mozilla и Firefox).

Стандартизация поспевала за практикой: HTTP/1.0 описали в RFC 1945 (июнь 1996, Бернерс-Ли, Филдинг и Фристык) — документ зафиксировал то, что браузеры и серверы уже умели: заголовки, коды состояния, методы POST и HEAD, типы содержимого из MIME. Тогда HTTP стал универсальным транспортом для любых данных, а не только гипертекста.

Анатомия запроса и ответа

HTTP — текстовый протокол обмена ресурсами по модели «клиент — сервер». Текст здесь буквальный: запрос можно набрать руками, и простейшая отладка до сих пор выглядит как telnet на порт 80 и четыре строки:

```
GET /index.html HTTP/1.1
Host: example.com
User-Agent: Mozilla/5.0
Accept: text/html
```

Стартовая строка задает метод (GET), путь к ресурсу (/index.html) и версию протокола. Далее заголовки — пары «имя: значение»: кто спрашивает (User-Agent), какой хост имеется в виду (Host — обязателен с HTTP/1.1 и сделал возможным виртуальный хостинг: десятки сайтов на одном IP-адресе), какой формат предпочтителен (Accept). Пустая строка

отделяет заголовки от тела: в GET его обычно нет, в POST и PUT там лежат данные. Ответ зеркален: статусная строка «HTTP/1.1 200 OK», заголовки (Content-Type, Content-Length, Cache-Control), пустая строка, тело. В HTTP/1.1 добавился chunked-режим: тело передается порциями без заранее известной длины — основа стриминга поверх HTTP. Главная архитектурная удача формата — расширяемость заголовками: тридцать лет протокол наращивает возможности, не меняя грамматику, а старые посредники прозрачно пропускают незнакомые им заголовки.

Методы и статус-коды: семантика как контракт

Методы задают смысл действия. Набор описан в RFC 9110 (июнь 2022): GET — получить ресурс, HEAD — то же без тела, POST — отправить данные/создать, PUT — заменить ресурс целиком, DELETE — удалить, OPTIONS, CONNECT (туннель для TLS) и TRACE (практически запрещен) — служебные, PATCH (частичное обновление) добавлен отдельно — RFC 5789 (2010). За методами стоят два свойства, на которых держится надежность веба: безопасность (GET не меняет состояние сервера — его можно кэшировать, повторять, предзагружать) и идемпотентность (повтор PUT или DELETE дает тот же результат — клиент вправе ретранслировать запрос при обрыве, POST не идемпотентен, и нажатие «обновить» на странице оплаты — баг пользователя, а не браузера).

Статус-коды — язык ответов, пятиклассная система. 1xx — служебные (101 Switching Protocols — смена протокола, понадобится WebSocket'у), 2xx — успех (200 OK, 201 Created, 204 No Content, 206 Partial Content — частичное содержимое, на котором держатся докачка файлов и видео по диапазонам байт), 3xx — перенаправления (301 навсегда, 302 временно, 304 Not Modified — «кэш свеж», 307/308 — то же без смены метода), 4xx — ошибка клиента (400 Bad Request, 401 Unauthorized — «представься», 403 Forbidden — «знаю, кто ты, и не пущу», 404 Not Found, 429 Too Many Requests), 5xx — ошибка сервера (500 Internal, 502 Bad Gateway и 504 Gateway Timeout — типичные ответы прокси на смерть бэкенда, 503 Service Unavailable). Протокол не чужд самоиронии: код 418 I'm a teapot родился в первоапрельском RFC 2324 (1998) о протоколе управления кофеваркой и стал любимой шуткой индустрии, код 451 Unavailable For Legal Reasons (RFC 7725, 2016) назван в честь «451 градуса по Фаренгейту».

Stateless и cookies: история одного компромисса

HTTP принципиально не хранит состояние (stateless): сервер не помнит ни предыдущих запросов, ни кто их задавал. Решение сознательное: любой сервер кластера может обслужить любой запрос (масштабируемость), падение сервера не теряет «сессию» (отказоустойчивость), посредники свободно кэшируют ответы. Расплата — невозможность «узнать» пользователя: ни входа между страницами, ни корзины покупок.

Выход нашли в Netscape летом 1994 года: Лу Монулли решал задачу для первого коммерческого применения браузера — виртуальной корзины покупок. Идея была заимствована из Unix — «магические печенья» (magic cookies): небольшие данные, которые программа выдает и получает обратно неизменными. Так родился механизм

cookies: сервер отправляет заголовок Set-Cookie с идентификатором, браузер сохраняет его и автоматически прикладывает заголовок Cookie к последующим запросам к тому же домену. Состояние осталось на сервере, а протокол формально остался stateless.

Стандартизация шла мучительно: от RFC 2109 (1997), фиксировавшего расходящиеся версии Netscape и Microsoft, через почти мертвый RFC 2965 (2000) — к RFC 6265 (2011), описавшему реальную практику. Cookie бывают сессионные (умирают с браузером) и постоянные (Expires/Max-Age), с областью действия (Domain/Path) и атрибутами безопасности, добавлявшимися по мере атак: Secure — только по HTTPS, HttpOnly — не отдавать скриптам (защита от кражи сессии через XSS), SameSite — не прикладывать к межсайтовым запросам (защита от CSRF, с февраля 2020 года Chrome по умолчанию считает cookie «Lax»). По сути session cookie — bearer-токен: предъявил — значит владелец. Отсюда атаки на сессии (перехват по незашифрованному HTTP, кража через XSS, фиксация сессии) и практика: чувствительные cookie — только Secure+HttpOnly+SameSite, короткие сроки жизни, ротация идентификаторов.

Согласование содержимого: один URL — много представлений

Ресурс и его представление в HTTP разведены: по одному адресу сервер может отдать HTML браузеру и JSON программе. Механизм — согласование содержимого (content negotiation): клиент заявляет предпочтения заголовками Accept (тип данных с весами качества), Accept-Language (язык — en, ru, de), Accept-Encoding (сжатие — gzip, br), сервер выбирает представление и предупреждает кэши заголовком Vary: «ответ разный для разных Accept-Language, не отдай немцу английскую копию». Та же механика через Client Hints постепенно заменяет строку User-Agent — хаос самопрезентаций браузеров: строка современного Chrome до сих пор содержит «Mozilla/5.0 ... like Gecko ... Safari».

Эволюция HTTP: тридцать лет борьбы с задержкой

HTTP/1.0: цена каждого соединения

До HTTP/1.1 каждое соединение обслуживало один запрос: открыть TCP-соединение (рукопожатие — целый RTT), передать запрос, получить ответ, закрыть. Для страницы с пятьюдесятью картинками это катастрофа: пятьдесят рукопожатий и медленных стартов — TCP разгоняет окно постепенно, короткое соединение не успевает выйти на скорость. Браузеры выкручивались, открывая по два параллельных соединения на хост.

HTTP/1.1: постоянные соединения и конвейер, который не взлетел

HTTP/1.1 (RFC 2068, январь 1997, пересмотр RFC 2616, июнь 1999) сделал соединения постоянными по умолчанию (keep-alive): одно TCP-соединение обслуживает последовательность запросов-ответов, добавились chunked-передача, диапазоны байт (Range/206), строгое кэширование (Cache-Control). Спецификация прожила двадцать пять лет: в 2014-м ее рассекли на RFC 7230–7235 (Филдинг и Решке), а в 2022-м переиздали как RFC 9110–9112 — без изменения смысла.

Главная неудача HTTP/1.1 — конвейеризация (pipelining): спецификация разрешала отправить несколько запросов подряд, не дожидаясь ответов, но ответы возвращались строго по порядку. Одна медленная генерация в начале очереди задерживала все последующие — блокировка во главе очереди (head-of-line blocking), прокси массово ломали конвейер, и браузеры его отключили. Практика пошла другим путем: шесть параллельных соединений на хост, «шардирование доменов» (img1...img4.example.com), укрупнение файлов — спрайты иконок, склейка CSS и JavaScript. Вся фронтенд-оптимизация 2000-х — памятник борьбе с ограничениями HTTP/1.1.

HTTP/2: из SPDY в стандарт

Google решил проблему иначе: в ноябре 2009 года компания представила SPDY — экспериментальный протокол поверх TLS, где одно соединение несло множество параллельных потоков с приоритетами, заголовки сжимались, а сервер мог сам предлагать ресурсы (push), замеры показывали ускорение загрузки до полутора раз. В 2012 году IETF взяла SPDY за основу стандарта, и в мае 2015-го вышел RFC 7540 — HTTP/2 (переиздан как RFC 9113 в 2022-м).

Ключевые отличия: бинарное кадрирование вместо текста — сообщение режется на кадры, принадлежащие потокам, семантика HTTP/1.1 не изменилась, мультиплексирование — сотни запросов-ответов по одному TCP-соединению параллельно, что снимает и лимит шести соединений, и шардирование доменов, HPACK (RFC 7541) — сжатие заголовков со статическим словарем и динамической таблицей соединения, приоритеты потоков. Но две проблемы остались. Первая — TCP-блокировка во главе очереди: все потоки HTTP/2 делят один надежный байтовый поток, и потеря сегмента останавливает их все — на сетях с потерями HTTP/2 местами проигрывал HTTP/1.1. Вторая — server push: сервер не знал, что уже есть в кэше клиента, Chrome удалил поддержку в 2022 году. Сегодня HTTP/2 — рабочий стандарт крупных сайтов и CDN: по методике W3Techs его используют 35% сайтов (октябрь 2026).

HTTP/3 и QUIC: транспорт переезжает в протокол

Ответ на TCP-блокировку лежал на поверхности: заменить транспорт. QUIC (RFC 9000, май 2021, подробности — в статье о транспортном уровне) работает поверх UDP: каждый поток надежен сам по себе, рукопожатие TLS 1.3 встроено в транспортное (одно RTT на все, ноль — при повторном соединении), а идентификатор соединения позволяет мигрировать между сетями (Wi-Fi → LTE) без разрыва. В июне 2022 года вышел RFC 9114 — HTTP/3: та же семантика HTTP, сжатие заголовков QPACK (RFC 9204), но поверх QUIC вместо TCP.

Обнаружение работает через заголовок Alt-Svc («ресурс доступен и по h3») и DNS-записи HTTPS/SVCB, браузер пробует QUIC, а при неудаче (UDP заблокирован в 3–7% сетей) откатывается на TCP. Темпы внедрения беспрецедентны: за три года после RFC поддержку HTTP/3 заявляют 41,1% веб-сайтов (W3Techs, октябрь 2026), а фактическая доля запросов

по h3 в сети Cloudflare перевалила за пятую часть (Cloudflare Radar, III квартал 2026): мобильный Интернет с потерями и переключениями сетей — среда, где выигрыш QUIC максимален.

Из чего складывается задержка

Тридцатилетняя гонка версий — война за round-trip'ы, а они упираются в физику. Свет в оптоволокне идет со скоростью около 200 тысяч км/с — примерно 5 мс на тысячу километров в одну сторону, круговой рейс Нью-Йорк — Лондон составляет 70–80 мс, Сидней — Лондон — под 300 мс, и никакой протокол это не отменит. Бюджет рукопожатий до первого байта при холодном старте: DNS — от нуля RTT (запись в кэше) до двух-трех (полный обход дерева), TCP — 1 RTT, TLS 1.2 — 2 RTT (TLS 1.3 — 1 RTT, QUIC — 0–1 RTT вместе с транспортом), затем сам запрос-ответ — еще минимум 1 RTT. Итого три-пять круговых рейсов «впустую» — вот почему объединенное рукопожатие QUIC и кэши DNS дают выигрыш в сотни миллисекунд на трансконтинентальных соединениях. Дополнительные механизмы экономят рейсы точечно: TCP Fast Open (RFC 7413), 0-RTT в TLS 1.3, статус 103 Early Hints (RFC 8297, 2017) — промежуточный ответ с подсказками «начни качать эти CSS и шрифты», пока основной сервер генерирует страницу.

Загрузка страницы: от URL до пикселей

Конвейер из семи шагов

Соединим уровни в один сценарий — ввод адреса в браузере. Шаг 1: разрешение имени — кэши браузера и ОС, затем рекурсивный резолвер, при необходимости полный обход дерева. Шаг 2: установка соединения — TCP-рукопожатие (или QUIC, если известен h3). Шаг 3: TLS-рукопожатие — проверка сертификата, обмен ключами. Шаг 4: HTTP-запрос — с cookies и кэш-заголовками. Шаг 5: ответ — HTML. Шаг 6: парсинг и подзапросы — браузер порождает десятки новых запросов к CSS, JavaScript, шрифтам, картинкам — возможно, к другим доменам, со своим DNS и соединениями. Шаг 7: рендеринг. До первого байта при холодном старте проходит два-четыре RTT плюс время генерации на сервере, полная загрузка современной страницы занимает секунды даже на быстром канале.

Внутренности браузера: DOM, CSSOM и критический путь

Получив HTML, парсер строит DOM — дерево объектов документа, из CSS строится CSSOM — дерево стилей, их слияние дает render tree, основу для расчета геометрии (layout), рисования (paint) и сборки слоев (composite). Два ресурса ведут себя по-особому. CSS блокирует рендеринг: браузер не покажет ничего, пока не построит CSSOM, отсюда прием «инлайновый критический CSS». JavaScript блокирует парсинг: тег `script` без атрибутов останавливает разбор HTML до загрузки и выполнения скрипта, лечится атрибутами defer (выполнить после парсинга, по порядку) и async (выполнить сразу, независимо). Пока парсер заблокирован, работает preload scanner — предпросмотр текста вперед, вылавливающий URL ресурсов для раннего старта загрузки.

Страница как граф ресурсов

Современная страница — не документ, а граф зависимостей, и статистика HTTP Archive (Web Almanac 2025) это фиксирует: медианная страница весит около 2,4 МБ на десктопе и 2,1 МБ на мобильных, медианная главная — 2,86 МБ, за десятилетие вес вырос примерно втрое. Состав медианной десктопной главной: изображения около 1,06 МБ, JavaScript около 0,7 МБ, шрифты 0,139 МБ, CSS 0,082 МБ и сам HTML — 22 килобайта. Запросов — порядка сотни. Зависимости образуют каскад: HTML → CSS → шрифты, HTML → JS → вызов API → данные → новые ресурсы, каждое колено добавляет RTT к «водопаду» загрузки. Отсюда заповедь производительности: сокращать глубину графа (меньше последовательных зависимостей) и ширину (меньше запросов, крупнее кэш) — и помнить, что ответ бэкенда задерживает все, что от него зависит.

Метрики и оптимизация

Пользовательское восприятие скорости формализовано в Core Web Vitals (Google, май 2020): LCP — время отрисовки самого крупного элемента (цель — менее 2,5 с), CLS — сдвиг макета (менее 0,1), INP — отзывчивость на взаимодействия, с марта 2024 года заменивший FID (менее 200 мс), с 2021-го метрики учитываются в поисковой выдаче. Инструментарий оптимизации — следствие устройства конвейера: CDN приближает байты к пользователю, кэширование убирает запросы вовсе, сжатие (gzip, brotli — RFC 7932) режет тексты на 70–80%, форматы изображений WebP и AVIF режут размер на 30–50% против JPEG, ленивая загрузка откладывает ресурсы вне первого экрана, минификация и бандлинг уменьшают число файлов — с оглядкой на HTTP/2, где кэш одного огромного бандла инвалидируется целиком, resource hints (dns-prefetch, preconnect, preload) разрешают браузеру начать работу заранее. Универсальный вывод для бэкенда: каждый запрос — это латентность, а самый быстрый запрос тот, который не пришлось делать.

Кэширование и прокси: инфраструктура доставки

Иерархия кэшей

Если доставка байта через океан стоит десятки миллисекунд, лучший способ ее ускорить — не доставлять байт вовсе. Кэширование — центральный механизм веба, стандартизованный в RFC 9111 (июнь 2022). Кэши делятся на приватные (браузер — на одного пользователя) и разделяемые (shared — промежуточные). Иерархия: браузер → прокси → CDN → исходный сервер (origin). Каждый уровень поглощает свою долю запросов: у хорошо настроенного статического сайта CDN отдает из кэша более 90% обращений — это и есть механизм, позволяющий одному серверу обслуживать планету.

Cache-Control, ETag и 304 Not Modified

У кэширования два режима. Первый — кэш свежести (strong caching): ответ с «Cache-Control: max-age=3600» можно целый час отдавать, вообще не спрашивая сервер. Второй — валидация: кэш хранит копию с меткой (ETag — отпечаток версии, Last-Modified — дата изменения) и перед выдачей задает условный запрос (If-None-Match / If-Modified-Since),

если ресурс не изменился, сервер отвечает 304 Not Modified — пустым телом. Байты сэкономлены, но RTT потрачен — поэтому свежие ответы валидации не требуют. Директивы Cache-Control тонко управляют политикой: public против private (только браузеру — для персональных данных), no-store (не кэшировать никогда), no-cache (кэшировать, но валидировать каждый раз), s-maxage (отдельный TTL для разделяемых кэшей), stale-while-revalidate (RFC 5861) — отдать устаревшее немедленно, обновляя копию фоном. Заголовок Expires с абсолютной датой — наследие HTTP/1.0, от него отказались из-за рассинхронизации часов, типовой паттерн: HTML — короткий max-age или no-cache, статика с хешем в имени — max-age=31536000, immutable.

Инвалидация: сложная задача

Филу Карлтону приписывают афоризм: «В информатике есть две сложные задачи: инвалидация кэшей и именование сущностей» (шутники добавляют третью — ошибки на единицу). Кэш, отдающий устаревшее, — источник бесконечных аварий: «пользователи видят старую версию», «откатили деплой, а оно висит». Средства те же, что в DNS: TTL, версионирование URL (новый контент — новое имя файла с хешем), принудительная очистка (purge-запросы к CDN). Конфликт «скорость против актуальности» неразрешим в принципе — он лишь настраивается под класс контента.

Прямые и обратные прокси

Прокси — посредник, выполняющий запросы от имени другой стороны. Прямой прокси (forward proxy) стоит на стороне клиента: весь трафик организации идет через него, что дает контроль доступа (блокировка сайтов в офисе и в цензурных сетях), кэширование, частичную анонимность, классическая реализация — Squid (1996). Обратный прокси (reverse proxy) стоит на стороне сервера и для клиента неотличим от самого сервера: балансирует запросы по бэкендам, терминирует TLS (шифрование разбирается на прокси, внутрь идет обычный HTTP), кэширует статику, фильтрует атаки (WAF — web application firewall). Самый известный — nginx Игоря Сысоева: начат в 2002-м как ответ на проблему C10K (десять тысяч одновременных соединений), выпущен в октябре 2004-го, рядом — HAProxy (2000), Envoy (Lyft, 2016, основа сервисных сетей), API-шлюзы. Типовой путь запроса сегодня: клиент → CDN → обратный прокси → пул бэкендов.

Потоковое видео: как Интернет научился показывать кино

От RealAudio к инаугурации Обамы

Передать HTML-страницу просто, передать видео трудно: это мегабайты в секунду, чувствительные ко времени доставки. Приложений два класса — буферизуемое вещание (YouTube, Netflix: можно накопить запас) и интерактив в реальном времени (звонки: задержка критична, запас невозможен). Первыми пришли специализированные стеки: RealAudio компании RealNetworks (апрель 1995) с собственными серверами и портами, затем IETF-стандарты RTP и RTSP (RFC 1889 и RFC 2326, 1996–1998) и эра Flash-вещания по RTMP. Проблема «особых» протоколов одна: файрволы и NAT их режут, инфраструктура

кэшей их не понимает, серверы стоят дорого. Перелом случился в 2007–2009 годах: YouTube (основан в феврале 2005-го) доказал массовость онлайн-видео, Netflix запустил стриминг в январе 2007-го, а инаугурация Барака Обамы 20 января 2009 года — более миллиона одновременных зрителей, рекорд того времени — прошла через Akamai по обычному HTTP. Индустрия поняла: стриминг = обычный HTTP-трафик, и ему достаются бесплатно кэши, CDN, прохождение через файрволы и масштабирование.

Чанки, манифесты и адаптивный битрейт

Современная схема: видео нарезается на короткие файлы-сегменты (2–10 секунд), а их список лежит в плейлисте-манифесте. Apple представила HLS (HTTP Live Streaming) в июне 2009 года (стандартизован в RFC 8216, 2017): файл .m3u8 со списком сегментов .ts, позже — фрагментированный MP4. Конкурент — MPEG-DASH (ISO/IEC 23009-1, 2012): XML-манифест .mpd, независимость от кодека, его использует YouTube. В 2018-м усилия Apple и Microsoft родили CMAF — общий контейнер фрагментированного MP4, чтобы один набор файлов обслуживал и HLS, и DASH. Клиент скачивает манифест, затем сегменты — обычными GET-запросами, — складывая их в буфер на 5–30 секунд воспроизведения, при опустошении буфера зритель видит «колесико».

Ключевая идея поверх этого — адаптивный битрейт (ABR). Сервер хранит видео в нескольких качествах («лестница» от 240p до 4K), манифест перечисляет варианты, а клиент сам, измеряя скорость загрузки сегментов и уровень буфера, выбирает качество для следующих кусков — динамически, по ходу просмотра, без участия сети. Это триумф философии end-to-end: приложение адаптируется к каналу, а не канал подстраивается под приложение. Алгоритмы выбора эволюционировали от наивных «по последней скорости» (осцилляции качества) к буфер-ориентированным (BOLA, 2016) и гибридным (MPC), а в 2017-м MIT показал Pensieve — обучение с подкреплением. Целевая метрика — QoE: время старта, частота и длительность «залипаний» (rebuffering), качество и плавность переключений, исследования Netflix легли в канон: для зрителя зависание хуже низкого разрешения — отсюда агрессивная буферизация и осторожный апгрейд качества. Кодеки сменились трижды: H.264 → VP9 → AV1 (альянс AOMedia, 2018), каждый давал 30–50% экономии битрейта.

Реальное время: WebRTC и градации задержки

Буферизация и разговор несовместимы: для диалога приемлема односторонняя задержка до 150 мс (рекомендация ITU-T G.114), 400 мс — уже тяжело. Здесь царит RTP поверх UDP: потерянные пакеты не переотправляют (просроченный кадр бесполезен), потери скрывают интерполяцией, джиттер сглаживают маленьким буфером (механику разбирала статья о транспортном уровне). Вокруг этого ядра вырос WebRTC: Google купил шведскую Global IP Solutions в 2011-м, открыл стек, и в январе 2021 года W3C принял спецификацию 1.0 — звонки браузер-браузер без плагинов, с медиа по SRTP, обходом NAT через ICE/STUN/TURN и групповыми серверами SFU. Так сложилась лестница задержек: VoD (десятки секунд буфера) → live-стриминг (6–45 с) → low-latency live (LL-HLS/LL-DASH, Twitch — около 2 с) → интерактив WebRTC (доли секунды) → облачный гейминг (менее 100 мс). Чем ниже

ступень, тем меньше запас прочности: треугольник «задержка — качество — устойчивость» неразрешим целиком, и каждое приложение выбирает его угол. Новые стандарты (MoQ — Media over QUIC, WebTransport) пытаются раздвинуть границы, перенося доставку медиа на QUIC.

Экономика видеотрафика

Масштабы определяют архитектуру. Видео — около двух третей всего интернет-трафика (Sandvine, данные 2023 года), средний пользователь скачивает за день порядка 5,6 ГБ (AppLogic Networks, 2025), несколько технологических гигантов — Alphabet, Meta, а также Netflix, Disney и TikTok — дают свыше половины мирового веб-трафика. Netflix с его 300+ миллионами подписчиков решает задачу по-своему: программа Open Connect (2012) бесплатно размещает серверы непосредственно в сетях провайдеров, и около 95% трафика компании отдается «изнутри» ISP. Именно трафик таких гигантов разжег споры о сетевой нейтральности: конфликт Comcast и Level 3 (2010) и платный пиринг Netflix с Comcast (2014) поставили вопрос «кто платит за последний километр».

CDN и P2P: доставка контента в масштабе планеты

Akamai: из математической задачи — в индустрию

Индустрия доставки контента родилась из академической задачи. В 1995 году профессор MIT Том Лейтон и его аспирант Дэнни Левин занялись вопросом: как разложить содержимое сайта по тысячам серверов так, чтобы пользователь автоматически попадал на ближайший и наименее загруженный? Ответом стали консистентное хеширование (статья Каргера и соавторов, 1997) и умная DNS-маршрутизация. В августе 1998-го Лейтон и Левин основали Akamai, в 1999-м компания провела IPO на пике дотком-бума — и пережила крах бума, продавая «лопаты для золотой лихорадки». Дэнни Левин погиб 11 сентября 2001 года в рейсе American Airlines 11, ему был 31. Сама сеть в те дни выдержала беспрецедентный всплеск трафика именно благодаря распределенной модели — кэширующие края оказались прочнее центральных серверов новостей, падавших один за другим.

Дальше индустрия росла лавинообразно: Limelight (2001), Amazon CloudFront (2008), Fastly (2011) и Cloudflare — запущенный 27 сентября 2010 года Мэттью Принсом, Мишель Зетлин и Ли Холлоуэвым проект, выросший из анти-спам-сервиса Project Honey Pot и превратившийся в платформу безопасности и периферийных (edge) вычислений для заметной доли Веба. Netflix в 2012-м пошел особым путем Open Connect. К 2026 году Akamai объявляет о более чем 4 400 точках присутствия в 130+ странах с пропускной способностью свыше петабита в секунду, Cloudflare — о присутствии в 330+ городах, через CDN проходит преобладающая доля потребительского веб-трафика.

Как это работает: DNS, anycast, кэш

Механика CDN складывается из трех элементов. Контент: копии объектов лежат на краевых (edge) серверах, при промахе кэша (cache miss) край запрашивает объект у origin-сервера — часто через промежуточный щит (origin shield), чтобы сотни краев не дублировали промахи, — кэширует и отдает следующему пользователю. Маршрутизация: домен сайта записью CNAME делегируется в DNS провайдера CDN, а тот, видя адрес резолвера (и подсеть клиента через EDNS Client Subnet, RFC 7871), возвращает адрес ближайшего здорового края, крупные сети дополнительно объявляют один anycast-адрес из многих точек. Управление: TTL, принудительная очистка (purge), правила кэширования. Эффект четырехкратный: меньше задержка (сервер в двух шагах), меньше нагрузки на origin (в десятки раз), выше надежность доставки (короткие пути и собственные магистрали CDN между точками — Cloudflare Argo, Akamai SureRoute), выше отказоустойчивость. По сути CDN — гигантский распределенный кэш, ставший средой обитания веб-контента.

P2P как альтернатива — и почему она уступила

Идеологически противоположный подход — раздавать контент силами самих зрителей. BitTorrent (Брэм Коэн, июль 2001-го) довел его до совершенства: файл режется на куски, пиры обмениваются недостающими частями по принципу взаимности (tit-for-tat: раздаешь — получаешь), редкие куски приоритетны (rarest first — устойчивость роя), а с 2005 года координацию ведет распределенная хеш-таблица (DHT), сделавшая магнит-ссылки полностью децентрализованными. Пропускная способность системы растет с каждым новым зрителем — мечта для доставки обновлений: так раздают дистрибутивы Linux, игровые патчи Blizzard, архивы Internet Archive, Facebook в 2010-х использовал BitTorrent для развертывания конфигураций в своих дата-центрах.

Но коммерческий стриминг P2P не взяло: большинство пользователей за NAT не принимает входящие соединения, пиры непредсказуемо уходят (churn), гарантий качества нет, провайдеры режут P2P-трафик, а правообладатели ведут юридическую войну с файлообменом. Когда цены на полосу упали, а CDN выросли, экономика склонилась в пользу централизованных краев. Гибриды остались: mesh-звонки WebRTC между несколькими участниками, блокчейн-сети. Урок Napster — BitTorrent — Open Connect прост: децентрализация решает проблему емкости, но проигрывает управляемость и предсказуемость — и рынок платит за второе.

Edge-вычисления и обратная сторона централизации

Доставка статики стала commodity, и CDN превратились в вычислительные платформы: Cloudflare Workers (сентябрь 2017, изоляты V8 вместо контейнеров), AWS Lambda@Edge (2017), Fastly Compute@Edge позволяют исполнять код приложения в сотнях точек планеты — запрос обрабатывается там же, где лежит кэш. «Бэкенд» становится глобальной функцией, плата — сложность состояния (его приходится выносить в распределенные хранилища типа KV) и привязка к платформе.

Обратная сторона эффективности — концентрация риска. Одна ошибка конфигурации у крупного провайдера роняет заметную часть Интернета: 8 июня 2021 года часовая авария Fastly (изменение конфигурации клиента задело скрытый баг) уложила Reddit, Amazon, сайты правительств Великобритании и США, 22 июля 2021-го сбой EdgeDNS Akamai на 45 минут отключил банки (Lloyds, Virgin Money) и авиакомпании (American, United, Southwest, Delta), 2 июля 2019-го регулярное выражение в новой конфигурации WAF Cloudflare сожрало процессоры по всей сети, в январе 2026-го утечка маршрутов (тип 3/4 по классификации RFC 7908) на 25 минут нарушила доступность сервисов Cloudflare в части регионов. «Единых точек отказа» в архитектуре Интернета нет, зато они есть в его экономике — и мульти-CDN, мульти-DNS и чужие постмортемы стали частью инженерной гигиены.

Электронная почта: store-and-forward архитектура

Протокол старше Интернета в его нынешнем виде

Веб — не самое старое массовое приложение, почта старше. Ее путь от локальной программы SNDMSG до планетарной системы занял десятилетие: в августе 1982 года Джон Постел выпустил RFC 821 (протокол передачи SMTP) и RFC 822 (формат сообщения) — отделив почту от FTP, чьими командами MAIL и MLFL она пользовалась до того. Действующая редакция — RFC 5321 (2008), «почтовик» sendmail Эрика Оллмана, написанный в Беркли в начале восьмидесятых, дожил до наших дней. У SMTP был серьезный конкурент: стандарт X.400 от ITU — «взрослая» корпоративная почта OSI, которую Европа внедряла госзаказами. Победил простецкий текстовый SMTP, работавший поверх чего угодно: к середине девяностых X.400 сошел со сцены — первый и не последний случай, когда «правильный» стандарт проиграл «достаточно хорошему».

Путь письма: цепочка независимых прыжков

Почта — store-and-forward система: «сохрани и передай». Участники: MUA (почтовый клиент), MSA (сервер отправки), MTA (агенты передачи — серверы, пересылающие письмо между собой), MDA (агент доставки в ящик). Письмо идет прыжками: клиент → свой сервер отправки (порт 587, с аутентификацией) → MTA получателя (порт 25) → ящик → клиент получателя забирает его позже. Каждый прыжок — полноценное сохранение: сервер принимает сообщение целиком, кладет в очередь и пытается передать дальше, временная ошибка (код 4xx) означает «повтори позже» — попытки длятся днями, постоянная (5xx) — возврат отправителю (bounce). В этом фундаментальное отличие от HTTP: веб-запрос синхронен и сквозной, а письмо асинхронно и многошагово, недоступность получателя не отменяет отправку, а лишь откладывает ее. Именно поэтому почта пережила все сбои Интернета: архитектура очередей абсорбирует отказы — по самой природе сети, построенной на best effort с повторами.

Куда именно нести письмо, MTA узнает у DNS: MX-запись домена получателя (RFC 974, 1986) перечисляет почтовые серверы с приоритетами. Без DNS почта невозможна в принципе, поломка MX-записи — классическая причина «письма к нам не доходят», невидимая из веб-интерфейса.

SMTP-диалог и его порты

SMTP — текстовый протокол, буквально «чат между серверами»: отправитель представляется (EHLO — расширенное приветствие ESMTP, в ответ сервер перечисляет поддерживаемые расширения), объявляет конверт (MAIL FROM — обратный адрес, RCPT TO — получатель), командой DATA передает содержимое до строки с одинокой точкой, завершает сеанс (QUIT). Портов три: 25 — межсерверная передача (между чужими MTA аутентификации нет исторически — доверяли имени отправителя, и это доверие стало главной дырой почты), 587 — отправка от клиента к своему серверу (submission, RFC 6409: аутентификация обязательна, иначе сервер станет открытым ретранслятором спама), 465 — устаревший «implicit TLS», возвращенный практикой (RFC 8314). Шифрование между серверами добавляет STARTTLS (RFC 3207) — оппортунистическое, уязвимое к атакам снятия шифрования посредником (stripping), поэтому поверх придумали MTA-STS (RFC 8461) и DANE (RFC 7672), привязывающие политику к DNS. Более 90% входящего трафика Gmail сегодня идет по TLS (данные Google) — но сертификат сервера проверяется, а подлинность отправителя — нет.

POP3 и IMAP: две философии получения

Доставленное в ящик письмо клиент забирает сам — pull-модель, породившая два протокола с противоположными философиями. POP3 (RFC 1939, 1996) — «почта живет на моем компьютере»: клиент авторизуется (USER/PASS), спрашивает список (LIST), скачивает письма (RETR) и по умолчанию удаляет их с сервера (DELE). Модель честная для одного устройства и безнадежная для нескольких: ни папок, ни флагов, ни синхронизации. IMAP (Марк Криспин, Стэнфорд, 1986, RFC 3501, 2003, действующая версия IMAP4rev2 — RFC 9051, 2021) — «почта живет на сервере»: папки, флаги, поиск на сервере, частичная загрузка больших писем, расширение IDLE (RFC 2177) — долгоживущее соединение, через которое сервер сам сообщает о новых письмах. IMAP победил: мультиустройственность — норма жизни. Поверх него современность надстроила push-уведомления мобильных платформ (APNs у Apple, FCM у Google) и веб-почту по HTTPS: Hotmail (4 июля 1996-го — первая массовая веб-почта, проданная Microsoft примерно за 400 миллионов долларов) и Gmail (1 апреля 2004-го — гигабайт ящика и приглашения, мир всерьез обсуждал, не первоапрельская ли это шутка). Наследник IMAP — JMAP (RFC 8620, 2019, детище Fastmail): JSON поверх HTTP, приживается медленно — почта консервативнее веба.

Формат письма и MIME

Само письмо — текст с заголовками по RFC 822 (ныне RFC 5322): From, To, Subject, Date, Message-ID и цепочка Received — каждый сервер пути штампует свою строку, образуя встроенный протокол перемещений, по которому криминалисты восстанавливают

маршрут письма. Тело — изначально строго 7-битный ASCII: ни вложений, ни кириллицы, ни картинок. Расширение MIME (Натаниэль Боренштейн и Нед Фрид, первые RFC — 1992, действующие 2045–2049 — ноябрь 1996-го) решило все разом заголовками Content-Type и Transfer-Encoding: multipart-структуры склеивают в одном письме текст, HTML-версию и вложения, бинарные данные кодируются base64. Первое MIME-письмо Боренштейн отправил в 1992 году коллегам по Bellcore — с фотографией выступления своего вокального ансамбля. Так протокол, «умеющий только текст», научился всему, что умеет почта сегодня, — включая HTML-письма, которые клиенты рендерят с легендарной кривизной (Outlook до сих пор верстает их движком Microsoft Word). Полная интернационализация (адреса и заголовки в UTF-8, RFC 6530–6532, 2012) не взлетела — почта остается 7-битной в душе.

Спам: экономика пятидесятилетней войны

Первый сетевой спам — рассылка Тьюрка 1978 года. Первый скандальный — 12 апреля 1994-го: супруги-юристы Кантер и Зигель разослали рекламу «лотереи грин-карт» в шесть тысяч групп Usenet, назвали это маркетингом и собрали, по собственным словам, сто тысяч долларов. Сам термин — из скетча «Монти Пайтона», где слово spam повторяется, пока не заглушает разговор: точная метафора. Экономическая природа спама неустранима: стоимость отправки стремится к нулю, значит, прибыльна конверсия в доли промилле — рациональная чума. К 2008–2010 годам спам составлял 85–90% мирового почтового трафика, его гнали ботнеты (Rustock, Srizbi, Mega-D) с «пуленепробиваемых» хостингов. Показательна операция против McColo: 11–12 ноября 2008 года вышестоящие провайдеры отключили калифорнийского хостера — и мировой спам за сутки упал примерно вдвое.

Борьба со спамом и аутентификация отправителя: SPF, DKIM, DMARC

Защита строилась слоями: DNSBL — репутационные черные списки, распространяемые через сам DNS (первым был MAPS Пола Викси, 1997), байесовские фильтры содержимого (эссе Пола Грэма «A Plan for Spam», 2002), серые списки (greylisting, 2003) — временный отказ с ожиданием повтора: нормальный MTA повторит, спам-бот нет. Но главная дыра — отсутствие аутентификации отправителя: SMTP верит полю MAIL FROM на слово, и подделать «письмо от вашего банка» тривиально. Закрыли ее тремя надстройками. SPF (идея 2003 года, Менг Венг Вонг, RFC 7208, 2014): владелец домена публикует в TXT-записи список IP, которым разрешено слать его почту, — получатель сверяет адрес соединения. DKIM (RFC 4871, 2007, слияние Yahoo DomainKeys и Cisco Identified Internet Mail): домен криптографически подписывает письмо, подпись проверяется публичным ключом из DNS и переживает пересылки. DMARC (RFC 7489, 2015, начат PayPal с Google, Microsoft и Yahoo): политика — что делать, если проверки не пройдены (принять/карантин/отклонить), плюс отчеты владельцу домена. Финальный аккорд — 1 февраля 2024 года: Google и Yahoo сделали для массовых отправителей обязательными SPF или DKIM, выравнивание DMARC, отписку в один клик и долю жалоб ниже 0,3% — сорок два года

спустя после RFC 821 аутентификация отправителя стала де-факто обязательной. Война не окончена: спам держится на уровне около 45% трафика (2025), а вместо него расцвели фишинг и деловые мошенничества (BEC) — атаки на людей, а не на протокол.

Почему почта не умирает

Почту хоронят каждое десятилетие — и каждое она переживает хоронящих. Причины структурные: федеративность (ни владельца, ни единого сервиса — любой может поднять свой сервер), асинхронность (не требует присутствия обеих сторон), открытые стандарты и сорокапятилетний сетевой эффект. Плюс роль, которую никто не планировал: адрес почты стал универсальным идентификатором человека в Интернете — на нем держатся регистрация, восстановление паролей и двухфакторная аутентификация всего остального веба. Цифры 2026 года: около 390 миллиардов писем в сутки и более 4,5 миллиарда пользователей (Radicati Group) — старейший прикладной протокол Интернета остается одним из самых нагруженных.

API современности: WebSocket, REST, GraphQL, gRPC

Сервер не может заговорить первым

Классический HTTP — протокол строгой очередности: говорит клиент, сервер отвечает, инициатива всегда у клиента. Для страниц и файлов этого достаточно, но чаты, уведомления, биржевые котировки, мультиплеер и совместное редактирование требуют обратного — серверного push. Пока стандарта не было, веб изворачивался: периодический опрос (polling) жег трафик впустую, длинный опрос (long polling, обобщенный под именем Comet в 2005–2006 годах) держал запрос открытым, пока не появятся данные, — полумера, где каждый «push» стоил нового соединения.

WebSocket: полный дуплекс на порту 80

Стандарт появился в декабре 2011 года — RFC 6455, выросший из экспериментов Майкла Картера (Kaazing, 2008) и спецификации, доведенной до ума Иэном Хиксоном. Устроено изящно: соединение начинается как обычный HTTP-запрос с заголовком «Upgrade: websocket», сервер отвечает статусом 101 Switching Protocols — и TCP-соединение перестает быть HTTP: стороны обмениваются двоичными кадрами (текстовыми или бинарными, со служебными ping/pong/close) в обоих направлениях одновременно. WebSocket остается на портах 80/443 и проходит сквозь прокси и файрволы — та же маскировка под HTTP, которую позже применит QUIC. Клиентские кадры обязательно маскируются случайным ключом — защита от атак на кэширующие прокси, из-за которых в 2010 году пришлось отозвать ранние реализации протокола. Цена дуплекса — состояние: долгоживущие соединения требуют балансировки с «липкими» сессиями, pub/sub-слоя между серверами и аккуратной работы с обрывами. Там, где нужен поток только «сервер → клиент», проще SSE (Server-Sent Events), новое поколение — WebTransport поверх HTTP/3.

REST: архитектурный стиль, а не протокол

В 2000 году Рой Филдинг — один из авторов HTTP/1.1 — защитил диссертацию «Архитектурные стили и дизайн сетевых программных архитектур», где описал и назвал REST (Representational State Transfer). Это не протокол, а набор ограничений на архитектуру: клиент-серверное разделение, отсутствие состояния на сервере, кэшируемость ответов, единообразный интерфейс — ресурсы идентифицируются URI и манипулируются представлениями (JSON, XML) через стандартные методы, — и слоистость системы. На практике «REST API» означает HTTP + JSON + «существительные» в URL (/users/42/orders) + глаголы-методы + осмысленные статус-коды. JSON совершил собственную революцию: формат, описанный Дугласом Крокфордом в 2001-м и стандартизованный в RFC 8259 (2017), вытеснил XML из веб-API — компактность и «родность» браузерам победили схемы и пространства имен. REST выиграл эпоху публичных API, не изобретая нового протокола: семантика HTTP — кэш, методы, коды — стала конструктором.

GraphQL: клиент заказывает форму ответа

У REST обнаружились системные издержки: overfetching — ответ несет больше полей, чем нужно клиенту, и underfetching — связанные данные требуют серии запросов («за профилем — запрос за друзьями, за друзьями — за их постами»). Ответ придумали в Facebook, где в 2012 году переписывали мобильное приложение: GraphQL (открыт вместе со спецификацией в сентябре 2015-го) переворачивает отношения — клиент сам описывает нужную форму данных запросом-деревом, а сервер собирает ответ ровно такой формы, из одного эндпоинта, по строгой типизированной схеме с самоинтроспекцией. Знаковое принятие — GitHub API v4 (май 2017). Расплата: HTTP-кэш почти перестает работать (все запросы — POST в один URL), резолверы порождают проблему N+1 к базе, а злонамеренно глубокий запрос становится вектором DoS. GraphQL не заменил REST — он занял нишу сложных клиентских приложений с сильным фронтендом.

gRPC: вызов функции между дата-центрами

История RPC — удаленного вызова процедур — началась со статьи Биррела и Нельсона из Xerox PARC (1984): «вызов должен выглядеть как локальный». Sun RPC (RFC 1050, 1988) породил NFS, CORBA (1991) — эпоху тяжеловесного объектного middleware, SOAP и XML-RPC (рубеж 2000-х) — корпоративную интеграцию поверх XML, от которой устали все, что и расчистило дорогу REST. Но внутри дата-центров RPC никуда не исчезал: Google с начала 2000-х использовала внутреннюю систему Stubby с бинарной сериализацией protobuf (открыта в 2008-м) — к середине 2010-х через нее проходили триллионы вызовов в неделю. В марте 2015 года Google анонсировал gRPC — открытое переосмысление Stubby: контракты сервисов описываются в .proto-файлах, кодогенераторы делают клиентские и серверные классы для десятка языков, транспорт — HTTP/2, четыре режима работы: унарный вызов и три вида стриминга — серверный, клиентский и двунаправленный (структурированный аналог WebSocket, но со схемой и типами). gRPC стал стандартом микросервисной коммуникации и фундаментом сервисных сетей (Istio, Envoy), браузерам

доступен через gRPC-Web и прокси. Выбор между REST и gRPC диктуется границей системы: наружу — читаемый и кэшируемый HTTP+JSON, внутрь, между своими сервисами, — быстрый и типизированный gRPC.

Три модели взаимодействия

Все многообразие прикладных протоколов сводится к трем моделям. «Запрос — ответ» (HTTP, REST, GraphQL): синхронно, клиент ведет, ответы кэшируются, так устроено большинство веба. «Поток» (WebSocket, SSE, gRPC-стриминги, WebRTC): долгоживущий канал, сервер говорит сам, так устроены чаты, котировки, совместная работа. «Сохранение с пересылкой» (почта, очереди сообщений): асинхронно, участники не обязаны быть онлайн одновременно, отказ промежуточного узла не теряет данные. Реальные системы комбинируют модели — REST для данных, WebSocket для живых обновлений, очереди для фоновых задач, — и выбор модели есть архитектурное решение первого порядка.

Ветераны и павшие: FTP, telnet, Gopher

FTP: протокол старше TCP

Протокол передачи файлов старше стека, в котором работает: RFC 114 вышел 16 апреля 1971 года (Абхай Бхушан, MIT) — в эпоху NCP, за двенадцать лет до появления TCP/IP, действующая спецификация RFC 959 датируется 1985-м. Архитектурная особенность FTP — два соединения: управляющее (порт 21, команды и ответы) и отдельное для данных (порт 20 в активном режиме — сервер сам подключается к клиенту! — или согласованный диапазон в пассивном). Активный режим сломался о NAT задолго до того, как это стало проблемой веба. В 1980–1990-е анонимный FTP был главным механизмом распространения ПО — архивные серверы держали зеркала всего софта, а первая поисковая система Archie (1990, Университет Макгилла) индексировала списки FTP-каталогов. Безопасность FTP нулевая — логин, пароль и данные идут открытым текстом, надстройка FTPS (RFC 4217, 2005) спасла ситуацию лишь отчасти, и административную нишу занял SFTP — протокол поверх SSH, при похожей аббревиатуре не имеющий к FTP почти никакого отношения. Агония завершилась в 2021-м, когда Chrome (версия 88) и Firefox (версия 90) удалили поддержку FTP из браузеров, протокол жив в легаси-интеграциях и хостинге.

Telnet и SSH: цена открытого пароля

Telnet (первые реализации — 1969 год, стандарт RFC 854 — 1983-й) давал удаленный терминал в открытом тексте: команды и пароли летели через весь Интернет в виде, доступном для чтения любому посреднику. Весной 1995 года атака сниффера на сеть Хельсинкского технологического университета собрала пароли сотен пользователей — и Тату Илонен за неделю написал SSH: шифрованный терминал, порт 22, публикация в июле того же года. SSH-2 (RFC 4251–4254, 2006) и свободная реализация OpenSSH (1999)

вытеснили telnet, rlogin и rsh из администрирования повсеместно. Сегодня SSH — универсальный швейцарский нож: терминал, туннели, проброс портов, транспорт для git и SFTP.

Gopher: дорога, которую не выбрали

В апреле 1991 года команда Университета Миннесоты (Марк Маккахилл и коллеги) выпустила Gopher — систему меню-ориентированных документов: простую в развертывании, иерархическую, с поиском через Veronica. Годы Gopher и Веба совпали, и университеты сначала охотнее ставили именно Gopher. Развязка наступила в феврале 1993-го: Миннесота объявила о лицензионных сборах за серверные реализации — а 30 апреля 1993-го CERN объявил технологии WWW бесплатными для всех. Веб победил не только архитектурой (гиперссылки против меню), но и решением не брать денег. Gopher дожил до сегодняшнего дня в крошечной субкультуре энтузиастов — учебный пример того, как в сетях с сетевым эффектом ценообразование на протокол равносильно самоубийству.

Прочие долгожители

Рядом с Вебом тихо работают протоколы-старожилы: NTP (Дэвид Миллс, RFC 958, 1985) — синхронизация часов как прикладной сервис, whois (RFC 812, 1982), IRC (Яркко Ойкаринен, 1988), NNTP — транспорт Usenet. Прикладной уровень — одновременно музей и лаборатория: протокол 1971 года соседствует в одном стеке с протоколом прошлого года.

Реклама, трекинг и приватность

Куки-экономика

Бесплатный веб оплачен рекламой, а реклама — отслеживанием. Механика проста: на страницу встраивается ресурс стороннего домена — скрипт рекламной сети или пиксель аналитики, — и этот домен ставит (или читает) собственный cookie, тот же домен на других сайтах узнает того же пользователя — так складывается поведенческий профиль, а показ каждого объявления уходит на аукцион в реальном времени. DoubleClick, основанная в 1995-м, построила на этом бизнес, проданный Google в 2008-м за 3,1 миллиарда долларов. Масштаб: хотя бы один трекер присутствует на 75% десктопных и 74% мобильных страниц (HTTP Archive, 2025).

Битва за трекинг

Первые сражения отгремели в 1990-х: разоблачения прессы, слушания FTC, провалившаяся саморегуляция через протокол P3P (W3C, 2002). Затем пришло право: Директива ePrivacy (2002/2009) породила европейские баннеры согласия, GDPR (применяется с 25 мая 2018-го) сделал обработку персональных данных юридически рискованной. Затем — браузеры: Safari с Intelligent Tracking Prevention (2017) и Firefox с Enhanced Tracking Protection (2019) заблокировали сторонние cookies по умолчанию, Chrome обещал то же в 2020-м (Privacy Sandbox), откладывал четыре года, в июле 2024-го

отказался от блокировки, а в 2025-м ограничился «выбором пользователя» — конфликт интересов крупнейшего браузера, принадлежащего крупнейшему продавцу рекламы, в чистом виде. Apple в апреле 2021-го (iOS 14.5) потребовала явного разрешения на трекинг в мобильных приложениях — удар, стоивший, по оценкам, до 10 миллиардов долларов выручки одному только Facebook за год. Отслеживание мутировало: вместо cookies — фингерпринтинг (шрифты, canvas, характеристики устройства), который нельзя выключить, не сломав веб, ответы — partitioned cookies (CHIPS), Topics API, Storage Access API. Приватность стала свойством протоколов, а не только настроек.

Приватность как свойство протокола

HTTPS убрал из цепочки наблюдателя-посредника, но не наблюдателя-получателя: сервер по-прежнему знает о пользователе все. Метаданные тоже текут: SNI в TLS-рукопожатии называет домен даже в зашифрованном соединении (лечение — Encrypted Client Hello), DNS-запросы раскрывают историю (лечение — DoH/DoT), IP-адрес связывает сеансы. Прикладной уровень стал фронтом приватности потому, что именно здесь байты приобретают смысл — и именно здесь есть что защищать.

Безопасность прикладного уровня

HTTPS: от привилегии к умолчанию

Первые два десятилетия веб был открытым текстом: SSL родился в Netscape (1994–1995), TLS 1.0 стандартизован в 1999-м (RFC 2246), TLS 1.2 — в 2008-м (RFC 5246), TLS 1.3 — в августе 2018-го (RFC 8446, после четырех лет работы и 28 черновиков: рукопожатие в один RTT, выброшенные небезопасные шифры). Но до середины 2010-х HTTPS оставался витриной для банков и платежных страниц: сертификаты стоили денег, выпуск — ручного труда. Перелом сделали два события: разоблачения Эдварда Сноудена (2013) превратили открытый веб в политическую проблему, а Let's Encrypt (проект ISRG, первый сертификат — 3 декабря 2015-го) — в решенную инженерную: бесплатные сертификаты, автоматизация по протоколу ACME, срок жизни 90 дней вместо года. К концу 2025-го Let's Encrypt выдавал до 10 миллионов сертификатов в день, защищая сотни миллионов сайтов, доля HTTPS-загрузок в Chrome достигла 95–99% еще к 2020 году, браузер с 2016-го помечает открытый HTTP как «Не защищено», а с октября 2026-го (Chrome 154) включает режим «Всегда использовать безопасное соединение» по умолчанию. Сегодня HTTPS — норма, а исключение требует объяснений.

Атаки на приложения и на сами протоколы

Классика прикладных атак вошла в списки OWASP: инъекции (SQL — данные как код), XSS (внедренный скрипт крадет cookies — отсюда HttpOnly), CSRF (злоумышленник заставляет браузер отправить запрос с автоматически приложенными cookies — отсюда SameSite и токены). Отдельный пласт — атаки на сам HTTP: request smuggling (рассинхронизация трактовки Content-Length и Transfer-Encoding между цепочкой прокси — описана в 2005-м, вернулась волной после исследований Джеймса Кеттла 2019 года), отравление веб-кэшей

(неучтенный заголовок попадает в кэш CDN и обслуживает жертвам чужой контент), атаки на заголовок Host. DDoS седьмого уровня бьет не в канал, а в логику: флуд дорогими эндпоинтами и Slowloris (2009) — удержание тысяч соединений неполными запросами. Отдельная эпоха — атаки на цепочку поставок: тайпосквоттинг пакетов npm и PyPI, dependency confusion и венец жанра — SolarWinds (2020), где целью стал не сервер, а механизм обновления ПО. Общий вывод сорока лет: прикладной уровень расширял поверхность атаки быстрее, чем средства защиты.

Доверие к имени: шов между DNS и TLS

Доверие в вебе собирается из двух половинок: DNS говорит, где находится имя, а сертификат удостоверяющего центра (CA) подтверждает, что отвечающий этим именем владеет. Шов между ними — излюбленное место атак. Подделка DNS упирается в TLS: атакующий не предъявит валидный сертификат на чужой домен — но лишь при условии, что сертификаты не выдают кому попало. Когда выяснилось, что Symantec массово выпускал их без должной проверки, браузеры в 2017–2018 годах лишили доверия всю компанию — прецедент казни удостоверяющего центра. Механизмы починки доверия: Certificate Transparency (RFC 6962) — публичные логи всех выпущенных сертификатов, CAA-записи в DNS — владелец домена указывает, каким CA разрешено выдавать ему сертификаты, DANE/TLSA — привязка сертификата к домену через DNSSEC, вовсе без CA. Ни один механизм не стал повсеместным — безопасность имен остается лоскутным одеялом.

Путь одного заказа: собираем все вместе

Проследим сквозной сценарий — покупку в интернет-магазине пользователем, который находится за океаном от исходного сервера, — и увидим все механизмы уровня в одной связке.

В адресной строке появляется shop.example.com/checkout. Браузер проверяет кэш DNS: запись жива, TTL не истек — разрешение имени не стоило ни одного запроса. Холодный старт прошел бы длиннее: рекурсивный резолвер обошел бы anycast-корни, зону .com и авторитетные серверы магазина, где запись CNAME увела бы поиск в DNS CDN-провайдера, а тот вернул адрес ближайшего края. DNS-запись типа HTTPS подсказала еще до соединения: край говорит по HTTP/3 — браузер открыл QUIC-соединение со встроенным рукопожатием TLS 1.3, а поскольку визит повторный, данные ушли в первом же пакете (0-RTT).

Запрос POST /checkout несет сессионные cookies (Secure, HttpOnly, SameSite=Lax), условные заголовки If-None-Match и Accept-Encoding: br. Персонализированный ответ кэшировать нельзя — Cache-Control: private, no-store, — и край проксирует запрос через промежуточный щит к origin: обратный прокси nginx балансирует его на бэкенд, тот вызывает внутренние сервисы по gRPC (protobuf поверх HTTP/2), а публичный контракт

описан REST-ресурсами. Пока бэкенд генерировал страницу, статус 103 Early Hints уже отправил браузеру подсказку начать загрузку стилей — водопад ресурсов стартовал до основного ответа.

Приходит 200 OK с HTML. Браузер строит DOM и CSSOM: критический CSS встроен, основной JavaScript помечен defer, preload scanner заранее разослал запросы к статике. Десятки файлов — шрифты, скрипты, изображения в AVIF — имеют хеши в именах и заголовки max-age=31536000, immutable: их отдают кэши ближайшего края, почти не тревожа origin. Ленивая загрузка откладывает все, что ниже первого экрана, рендеринг укладывается в нормативы Core Web Vitals даже на среднем мобильном. Сторонний домен аналитики получает beacon — и ставит partitioned cookie, уже не способный связать пользователя между сайтами напрямую.

Подтверждение заказа уходит письмом. Бэкенд отдает его своему серверу отправки по SMTP на порт 587 с аутентификацией, тот подписывает сообщение DKIM-подписью домена и соединяется с почтовым провайдером получателя — адрес найден по MX-записи, транспорт поднят через STARTTLS согласно политике MTA-STS. Принимающий сервер проверяет SPF, DKIM и DMARC — письмо ложится в ящик, а мобильная платформа шлет push-уведомление. Будь получатель недоступен, письмо спокойно пролежало бы в очереди сутки: store-and-forward никуда не спешит.

Одна покупка задействовала именование и кэширование, мультиплексирование и адаптацию, три модели взаимодействия, четыре механизма доверия — и ни один из протоколов не знал о существовании остальных. В этом и состоит устройство прикладного уровня: десятки независимых соглашений складываются в одно пользовательское действие.

Итоги

Прикладной уровень — место, где Интернет становится собой для людей: нижние уровни пересылают байты, и лишь здесь байты превращаются в страницы, письма, видео и вызовы функций. Все, что мы разобрали, укладывается в несколько сквозных линий.

Первая линия — смысл поверх доставки. Уровень решает четыре задачи: формат сообщений, логика взаимодействия, именование и семантика. Протокол здесь — сетевой API, контракт между программами, а не между машинами, песочные часы Интернета расширяются вверх именно потому, что для добавления нового приложения не нужно ничьего разрешения — достаточно соглашения двух сторон. Отсюда и рекордная скорость эволюции слоя, и его главная беда: вместе со свободой творить пришла свобода творить небезопасно.

Вторая линия — именование как фундамент. DNS вырос из кризиса одного файла HOSTS.TXT в распределенную иерархическую базу на 401,6 миллиона доменов, работающую на кэшировании, делегировании и TTL. Из переводчика имен он превратился в план управления Интернетом — через него балансируют нагрузку, геолоцируют пользователей, включают failover и направляют трафик в CDN. Он же остался

незащищенным по рождению: спуфинг и отравление кэша лечились рандомизацией и DNSSEC, приватность — DoT/DoH, а роль критической инфраструктуры подтверждалась атаками на корни (2007), Spamhaus (2013) и Dyn (2016). Без DNS Интернет технически жив — но для людей и программ нерешаем.

Третья линия — кэш как способ масштабирования. Иерархия «браузер → прокси → CDN → origin» с ее Cache-Control, ETag и 304 превратила планетарные расстояния в локальные ответы: медианная страница весит мегабайты, но большая часть их отдается из двух шагов от пользователя. CDN выросли из математической задачи MIT в инфраструктуру, несущую большинство потребительского трафика и исполняющую код на краю (edge). Расплата — организационная централизация: одна конфигурация Fastly (2021) или Akamai (2021) роняла заметную часть Веба, и мульти-CDN стал частью инженерной гигиены. Инвалидация кэша осталась сложной задачей — но версионирование URL и дисциплина TTL делают ее решаемой.

Четвертая линия — эволюция через совместимость. HTTP/0.9 знал один метод, HTTP/3 говорит на QUIC поверх UDP — а семантика (методы, заголовки, коды) не изменилась за тридцать пять лет, и страницы девяностых открываются сегодняшними браузерами. SMTP 1982 года до сих пор возит мировую почту, обрастая надстройками: MIME научил его вложениям, STARTTLS — шифрованию, SPF/DKIM/DMARC — проверке отправителя. FTP пережил собственный стек: родился до TCP и умер в браузерах лишь в 2021-м. Там, где совместимость окостенела, новый протокол маскируется под старый — QUIC прячется в UDP на порту 443, WebSocket начинает жизнь как HTTP-запрос. Эволюция прикладного уровня — работа с наследием, а не против него.

Пятая линия — три модели взаимодействия. «Запрос — ответ» (HTTP, REST, GraphQL), «поток» (WebSocket, gRPC-стриминги, WebRTC) и «сохранение с пересылкой» (почта, очереди) исчерпывают архитектурное пространство сетевых приложений, реальные системы комбинируют их, и выбор модели — первое архитектурное решение любого сервиса. Модель диктует все: инфраструктуру, обработку отказов, способ масштабирования.

Шестая линия — доверие и приватность достраиваются задним числом. Протоколы эпохи всеобщего доверия пришлось экстренно латать криптографией: TLS сделал HTTPS нормой (95–99% загрузок в Chrome, сотни миллионов сертификатов Let's Encrypt), Certificate Transparency и CAA чинят систему удостоверяющих центров, шифрованный DNS закрывает последнюю массовую дыру приватности. Одновременно прикладной уровень стал фронтом борьбы за приватность: сторонние cookies, трекеры на трех четвертях страниц, фингерпринтинг — и ответные SameSite, блокировки браузеров, GDPR. Война не окончена, и идет она заголовками и атрибутами — тем же оружием, каким строился сам уровень.

Для практика выводы приземленные. Во-первых, диагностика уровня проста и доступна: dig и nslookup показывают работу имен, curl -v — весь разговор HTTP, инструменты разработчика — водопад и кэши, большинство «загадочных» проблем — это TTL, кэш или cookies, понятые неверно. Во-вторых, архитектура приложения — это выбор модели

взаимодействия и бюджета задержки: физика RTT неумолима, и каждый лишний запрос, каждая глубина графа ресурсов, каждый синхронный вызов в цепочке стоят пользователю миллисекунд и секунд. В-третьих, кэш и состояние — части корректности, а не оптимизации: Cache-Control, Vary, SameSite и политика инвалидации должны проектироваться вместе с бизнес-логикой. В-четвертых, надежность — в разнообразии: несколько DNS-провайдеров, мульти-CDN, очереди вместо синхронных вызовов там, где допустима асинхронность.

Стек пройден до верха. От «LO» 29 октября 1969 года — двух букв оборвавшегося логина — до 390 миллиардов писем в сутки, сотен миллионов сайтов и видео на две трети мирового трафика: прикладной уровень превратил инженерию доставки в человеческую инфраструктуру. Первый веб-сайт info.cern.ch до сих пор отвечает по своему родному адресу — живой памятник главному свойству уровня: совместимости, которая оказалась важнее совершенства. Выше протоколов остаются только смыслы — и вся история прикладного уровня есть история о том, как байты учились их выражать.