Network: Часть 9. Анализ трафика

24.10.2026 10:06 edited 24.10.2026 10:52

От теории к наблюдаемой реальности

Современные информационные системы настолько зависимы от сети, что любая их проблема рано или поздно упирается в один и тот же вопрос: что на самом деле происходит в проводах и эфирах между компонентами. Инженер может в деталях знать, как устроены протоколы, как работает управление перегрузкой и чем рукопожатие отличается от завершения соединения, но это знание остается книжным до тех пор, пока он не откроет реальный дамп и не увидит в нем живое поведение системы — со всеми задержками, потерями, аномалиями и атаками. Анализ трафика — это процесс изучения сетевых пакетов и потоков с целью понять, что происходит в сети, где находится проблема и кто или что ее вызвало. Это не отдельный инструмент и не отдельная глава учебника, а практическое применение всех уровней сетевой модели — от канального до прикладного — к реальным данным.

Ключевой переход, который отличает практика от теоретика, формулируется просто: от утверждения «я знаю, как работает TCP» к умению «я вижу в дампе, почему TCP тормозит». Этот переход превращает анализ трафика из вспомогательной процедуры в самостоятельную инженерную дисциплину, которую сегодня применяют сетевые инженеры, специалисты по производительности (Performance Engineer), инженеры по надежности (SRE) и аналитики безопасности. У дисциплины есть собственный объект изучения (пакет, поток, поведение системы под нагрузкой), собственный инструментарий (анализаторы, зонды, телеметрия), собственные методики (корреляция событий, локализация по симптому, визуализация) и собственная история, уходящая корнями еще в эпоху радиосвязи.

Как и любая зрелая область, анализ трафика порождает ряд сквозных вопросов, которые определяют и повседневную практику, и развитие технологий:

1. **Диагностический вопрос.** Где проходит граница между «виновата сеть» и «виновато приложение»? Подавляющее большинство производственных споров о причинах сбоя сводится к тому, что разные команды видят разные части одной картины. Только сквозное наблюдение за трафиком — от клиента до сервера и обратно — позволяет локализовать проблему на конкретном уровне и в конкретном узле.
2. **Вопрос производительности.** Почему формально «здоровая» сеть, в которой нет потерь пакетов, может давать плохой пользовательский опыт? Ответы включают разбухание буферов (bufferbloat), хвостовые задержки, блокировку во главе очереди (head-of-line blocking) и стоимость шифрования — явления, которые не видны в средних цифрах и требуют анализа распределений и временных рядов.

3. **Вопрос шифрования.** Что остается анализируемым, когда большая часть трафика закрыта криптографией? Как выясняется, очень многое: метаданные, временные характеристики, размеры пакетов, параметры рукопожатий и отпечатки клиентов. Анализ трафика — это не чтение данных, а анализ поведения, и шифрование содержимого не отменяет его, а меняет его предмет.
4. **Вопрос безопасности.** Как распознать атаку в трафике — от классического SYN-флуда и сканирования портов до медленных прикладных атак, отраженных DDoS рекордной мощности и скрытых каналов утечки данных? Большинство атак оставляет протокольные следы задолго до того, как их замечают системы защиты, и умение читать эти следы остается базовым навыком аналитика.
5. **Методологический вопрос.** Как превратить чтение пакетов из искусства отдельных экспертов в воспроизводимый инженерный процесс? Ответ лежит в области корреляции: связывании пакетных дампов с потоковой телеметрией, метриками, распределенными трассировками и журналами приложений в единую систему наблюдаемости (observability).

Далее эта статья рассматривает анализ трафика последовательно: исторический контекст и объект анализа, инструментарий аналитика, поведение TCP в реальном трафике, включая управление перегрузкой и его современные алгоритмы, прикладной уровень — HTTP, DNS, TLS и новые транспорты, классические сетевые проблемы и их узнаваемые паттерны, метрики качества, связь анализа с безопасностью, методологию диагностики и, наконец, особенности анализа в современных архитектурах — программно-конфигурируемых сетях, оверлеях и облаках.

От радиоэфира к пакетам: краткая история анализа трафика

Трафик-анализ до появления пакетов

Сам термин «анализ трафика» (traffic analysis) пришел в сетевую инженерию из радиоразведки и значительно старше компьютерных сетей. Еще в Первую мировую войну воюющие стороны обнаружили, что даже не расшифровывая сообщения противника, можно извлечь ценнейшие сведения из самого факта их передачи: кто, кому, когда и сколько передает. Резкое оживление радиопередач в секторе фронта предвещало наступление, смена позывных и графиков связи выдавала перемещение частей. Во Второй мировой войне эта практика достигла зрелости: анализ объемов и маршрутов радиообмена дополнял работу дешифровщиков, а иногда и заменял ее, когда шифры не поддавались взлому. Фундаментальный принцип — «метаданные говорят не меньше, чем содержимое» — был сформулирован задолго до появления Интернета.

Гражданским аналогом военного трафик-анализа стали данные о телефонных соединениях (call detail records, CDR). Операторы связи с самого начала фиксировали, кто кому звонил, когда и как долго — для выставления счетов. Именно эти записи, а не прослушивание разговоров, исторически составляли основу коммуникационной разведки правоохранительных органов во многих странах: судебных запросов на получение

детализации соединений на порядки больше, чем на перехват содержимого разговоров, поскольку они дешевле, массовее и проще юридически. Преступный мир ответил собственным трафик-анализом в защиту: одноразовые телефоны, частая смена SIM-карт, намеренное «заглушение» безобидным трафиком. Эта гонка вокруг метаданных продолжается и в эпоху пакетных сетей — только вместо телефонных счетов объектом наблюдения стали потоки пакетов, журналы соединений и телеметрия.

Эра ARPANET и рождение утилит

С появлением пакетных сетей анализ трафика постепенно превращался из разведывательной практики в инженерную. Перелом произошел в 1980-х годах, когда исследователи начали строить утилиты, ставшие классикой. В декабре 1983 года Майк Муус, работавший в исследовательской лаборатории армии США (BRL), написал программу ping — зонд, отправляющий ICMP-сообщения echo и измеряющий время ответа. Название отсылало к эху гидролокатора, сама утилита стала первым массовым инструментом проверки «живости» узла и до сих пор остается отправной точкой любой диагностики.

В 1987–1988 годах Ван Джейкобсон из Lawrence Berkeley National Laboratory (LBNL) создал traceroute — утилиту, восстанавливающую маршрут прохождения пакетов за счет последовательного увеличения поля TTL и анализа ICMP-ответов «время превышено». По некоторым свидетельствам, идея метода обсуждалась ранее Стивом Дирингом, но рабочая реализация и ее распространение связаны именно с Джейкобсоном. Трассировка маршрута с измерением задержек на каждом переходе (hop) позволила впервые «увидеть» интернет как последовательность узлов, а не как абстрактное облако.

Там же, в LBNL, во второй половине 1980-х годов Джейкобсон вместе с Крейгом Лересом и Стивеном Макканном начал работу над tcpdump — утилитой захвата и декодирования пакетов. Ее ключевым новшеством стал механизм фильтрации в ядре: в 1993 году Макканн и Джейкобсон опубликовали работу о BSD Packet Filter (BPF), виртуальной машине, исполняющей фильтрующие программы непосредственно в ядре операционной системы. Это решило главную проблему раннего захвата — необходимость копировать в пользовательское пространство все пакеты подряд. BPF стал фундаментом библиотеки libpcap и, по сути, всей современной экосистемы снифферов: язык фильтров tcpdump до сих пор является стандартом де-факто для задания условий захвата.

Показательно, что анализ трафика и совершенствование протоколов развивались совместно. В середине 1980-х интернет столкнулся с явлением, получившим название «коллапс перегрузки» (congestion collapse): при росте нагрузки пропускная способность магистралей падала на порядки из-за лавинообразных повторных передач. Именно наблюдения за реальным трафиком привели Джейкобсона к созданию в 1988 году алгоритмов управления перегрузкой TCP — медленного старта (slow start) и избегания перегрузки (congestion avoidance), — которые буквально спасли интернет от паралича. С тех пор цикл «измерение — понимание — изменение протокола» повторялся не раз: анализ трафика был не наблюдателем, а двигателем эволюции сетей.

Графические анализаторы и системы обнаружения вторжений

В 1990-х годах анализ вышел за пределы консольных утилит. В середине десятилетия Верн Паксон (также из LBNL) разработал Bro — систему обнаружения вторжений нового типа, которая вместо сравнения байтов с сигнатурами строила событийную модель поведения сети: скриптовый язык позволял описывать политику безопасности на уровне «событий» (установление соединения, DNS-запрос, HTTP-сессия). В 2018–2019 годах проект был переименован в Zeek и остается одной из двух главных открытых платформ сетевого мониторинга безопасности. В 1998 году Марти Роэш выпустил Snort — легковесный сигнатурный анализатор трафика, который стал основой целого класса открытых систем обнаружения вторжений (NIDS), позднее проект был приобретен Sourcefire, а затем Cisco, но правила Snort по-прежнему используются повсеместно, включая многопоточный «наследник» Suricata, развиваемый сообществом OISF с 2009–2010 годов.

Параллельно развивались графические анализаторы. В конце 1997 года Джеральд Комбс, нуждавшийся в инструменте для поиска сетевых проблем и одновременно желавший глубже изучить сети, начал писать Ethereal, первый публичный релиз состоялся в июле 1998 года. Проект рос вместе с интернетом: добавлялись диссекторы протоколов, система фильтров отображения, статистики и графики. В мае 2006 года из-за проблем с торговой маркой проект был переименован в Wireshark — под этим именем он стал безусловным стандартом интерактивного анализа пакетов, поддерживающим сотни протоколов и работающим на всех основных платформах. Консольный собрат tshark позволил встроить тот же механизм декодирования в автоматизированные конвейеры обработки.

От дампов к телеметрии и программируемому анализу

Захват каждого пакета (full packet capture) дает максимальную детальность, но плохо масштабируется: дамп магистрального канала за сутки измеряется терабайтами. Поэтому с середины 1990-х годов развивался параллельный подход — агрегированная телеметрия. В 1996 году Cisco ввела NetFlow — экспорт сведений о потоках (кто с кем обменивался данными, в каком объеме) вместо самих пакетов, версия 9 (2003–2004 годы) сделала формат шаблонизируемым, а в 2013 году IETF стандартизировал его преемника IPFIX (RFC 7011). Альтернативой стал sFlow (RFC 3176, 2001), сочетающий статистическую выборку пакетов со счетчиками портов. Потоковая телеметрия пожертвовала детальностью ради способности хранить картину сети месяцами и искать в ней аномалии.

Следующий сдвиг принесла программируемость. В 2014 году Алексей Старовойтов радикально переработал механизм BPF, превратив его в eBPF — полноценную виртуальную машину в ядре Linux, позволяющую безопасно исполнять пользовательские программы в точках трассировки ядра. На основе eBPF выросли инструменты, которые извлекают события TCP (установка и время жизни соединений, повторные передачи, изменения окна) напрямую из ядра — без захвата пакетов вообще, с минимальными накладными расходами. Проекты вроде Cilium и Hubble перенесли этот подход в

контейнерные среды. Одновременно для задач высокоскоростного захвата появились механизмы обхода ядра (kernel bypass): DPDK, PF_RING, позднее AF_XDP — они позволяют обрабатывать потоки 10–100 Гбит/с, недоступные классическому стеку.

Итог этой эволюции — современная пирамида наблюдаемости: метрики и потоки дают постоянную картину и служат триггером, а полный захват пакетов применяется точечно, для сбора доказательств и детального разбора. Понимание того, какой слой пирамиды отвечает на какой вопрос, — часть методологии анализа, к которой мы вернемся в соответствующем разделе.

Объект анализа: пакет, поток и поведение

Инкапсуляция: чтение снаружи внутрь

Любой анализ начинается с понимания того, что именно попадает в дамп. Пакет в сети — это конструкция из вложенных заголовков:

```
Ethernet frame → IP packet → TCP/UDP segment → Данные приложения
```

Каждый уровень модели добавляет свой заголовок при отправке и снимает его при получении, аналитик читает их снаружи внутрь. Канальный кадр несет MAC-адреса соседей по сегменту и, возможно, метку VLAN, IP-заголовок — конечные адреса и параметры маршрутизации, транспортный сегмент — порты, флаги и номера, и только внутри находится полезная нагрузка прикладного протокола — HTTP-запрос, DNS-сообщение или зашифрованная запись TLS. В туннелированных средах уровней вложенности больше: пакет VXLAN или IPsec содержит внутри полноценный внутренний IP-пакет, и анализаторы умеют разбирать такие «матрешки», но аналитик должен всегда точно понимать, на каком уровне вложенности он находится.

Что видно на каждом уровне

Каждый слой заголовков отвечает на свои вопросы, и опытный аналитик извлекает из них максимум еще до того, как посмотрит на полезную нагрузку:

- **Канальный уровень (L2).** MAC-адреса источника и получателя, метки VLAN (802.1Q), служебные протоколы сегмента — ARP, NDP. Эти поля редко участвуют в диагностике интернет-трафика, но незаменимы внутри локальной сети: они показывают, кто с кем реально обменивается кадрами, и позволяют выявлять подмену адресов.
- **Сетевой уровень (L3).** IP-адреса источника и получателя, поле TTL, флаги и смещения фрагментации, поле DSCP. TTL уменьшается на единицу при прохождении каждого маршрутизатора, поэтому по его значению в полученном пакете можно грубо оценить расстояние до отправителя и операционную систему стека (разные реализации выбирают разные начальные значения — 64, 128, 255). Аномальный TTL в середине соединения — признак вмешательства промежуточного устройства: NAT, прокси, балансировщика или системы фильтрации.

- **Транспортный уровень (L4).** Порты, флаги TCP, порядковые номера (sequence number) и номера подтверждений (acknowledgment number), размер окна, параметры соединения — MSS, SACK, метки времени, масштабирование окна. Это самый информативный слой для диагностики производительности: почти все классические сетевые проблемы — потери, перегрузка, исчерпание буферов — проявляются именно здесь.
- **Прикладной уровень (L7).** Методы и коды ответов HTTP, запросы и ответы DNS, сообщения рукопожатия TLS, команды почтовых протоколов. Прикладной слой отвечает на вопрос «что делали», но его анализ возможен только там, где трафик не зашифрован или где у аналитика есть законные ключи.

Анатомия одного пакета: чтение по байтам

Полезно однажды разобрать пакет буквально по байтам — дальше структура воспроизводится по памяти. Кадр Ethernet II: 6 байт MAC-адреса назначения, 6 байт MAC-адреса источника, 2 байта типа полезной нагрузки (0x0800 — IPv4, 0x0806 — ARP, 0x86DD — IPv6), при наличии VLAN перед типом вставляется 4-байтовая метка 802.1Q. Заголовок IPv4 начинается с полубайтов версии и длины (4 и 5), далее байт DSCP/ECN, два байта полной длины, два байта идентификатора, флаги и смещение фрагмента (2 байта), байт TTL, байт протокола (6 — TCP, 17 — UDP, 1 — ICMP), контрольная сумма и по четыре байта адресов. Заголовок TCP: порты источника и получателя (по 2 байта), порядковый номер (4), номер подтверждения (4), смещение заголовка со служебными битами и флагами (2+2), окно (2), контрольная сумма (2), указатель срочности (2) и необязательные параметры — MSS, SACK, метки времени, масштаб окна. Умение мысленно наложить эту сетку на hex-дамп отличает инженера от пользователя анализатора и спасает в ситуациях, когда диссектор протокола отсутствует или ошибается.

Пакет против потока: TCP как поток байтов

Фундаментальная концептуальная граница проходит между мышлением «пакетами» и мышлением «потоками». TCP — это не последовательность пакетов, а поток байтов: приложение записывает в сокет байты, а протокол сам решает, как нарезать их на сегменты для передачи. Границы записей приложения не сохраняются, один сегмент может содержать части разных HTTP-запросов, а один запрос может быть размазан по десяткам сегментов. Анализаторы выполняют обратную операцию — сборку потока (reassembly), — и команды вроде «Follow TCP Stream» в Wireshark показывают именно байтовый поток таким, каким его видит приложение.

Практическое следствие: многие аномалии имеют смысл только на уровне потока. Переупорядочивание сегментов, повторные передачи, разрывы на границах окна получателя, паузы из-за алгоритмов пакетирования — все это невозможно корректно интерпретировать, рассматривая пакеты поодиночке. Инженер, который думает пакетами, видит «странные пакеты», инженер, который думает потоками, видит поведение транспортной системы.

Временная линия: событие важнее пакета

Высший уровень абстракции — рассмотрение трафика как временной линии событий. Аналитик смотрит не на отдельные пакеты, а на последовательность «время → событие → поведение». Простейший пример — разбор одной веб-транзакции:

```
0 ms     SYN
50 ms    SYN-ACK
100 ms   ACK
200 ms   HTTP-запрос
500 ms   HTTP-ответ
```

Из этих пяти строк извлекается больше, чем из сотни пакетов, рассмотренных вне контекста: время оборота (RTT) до сервера составляет около 50 мс — это характеристика сети, между получением запроса и началом ответа прошло около 300 мс — это характеристика сервера и приложения. Вывод «сеть в порядке, тормозит сервер» делается за секунды и не требует чтения полезной нагрузки.

Именно поэтому анализ трафика — это синтез всех уровней одновременно. Реальная транзакция выглядит как сквозная цепочка `DNS → TCP → TLS → HTTP → обработка на сервере`, и задача аналитика — найти, в каком звене цепочки теряется время или происходит сбой. Каждый уровень вносит свой вклад в общую задержку и может оказаться узким местом, мысленно «проигрывать» эту цепочку, глядя на дамп, — и есть то, что отличает системное понимание сети от простого перечисления фактов о протоколах.

Инструментарий аналитика

Инструментов анализа трафика сегодня десятки, но их функции складываются в несколько четких классов: интерактивные анализаторы пакетов, консольные средства захвата, утилиты состояния и активной диагностики, средства потоковой телеметрии и платформы программируемого анализа. Зрелость практика определяется не количеством освоенных программ, а пониманием того, какой класс инструмента отвечает на какой вопрос.

Wireshark и tshark: интерактивный стандарт

Wireshark остается главным инструментом интерактивного анализа. Он умеет захватывать трафик с интерфейсов в реальном времени, открывать сохраненные дампы (pcap, pcapng), декодировать сотни протоколов с полным разбором полей, применять фильтры отображения, выполнять сборку потоков и строить визуализации. Для повседневной диагностики особенно полезны несколько возможностей:

- **Expert Info** — встроенная система экспертных подсказок, которая ранжирует замеченные аномалии по уровням (Note, Warning, Error) и сразу указывает на повторные передачи, нулевые окна, ошибки контрольных сумм и прочие признаки проблем. С нее часто начинают разбор незнакомого дампа.

- **Фильтры отображения** — язык запросов к распакованному содержимому пакетов. Комбинации вроде `tcp.analysis.retransmission`, `tcp.analysis.zero_window`, `tcp.analysis.duplicate_ack`, `dns.time > 1`, `http.time > 0.5`, `tls.handshake.type == 1` позволяют за секунды отобрать интересующие события из сотен тысяч пакетов.
- **Follow Stream** — просмотр собранного потока TCP/UDP/HTTP так, как его видит приложение, быстрый способ понять семантику обмена, не листая сегменты.
- **Статистики и графики** — иерархия протоколов (Protocol Hierarchy), списки соединений и узлов (Conversations, Endpoints), графики интенсивности ввода-вывода (IO Graph) и специализированные графики TCP-потока: пропускная способность, время оборота, размер окна во времени. Визуальный паттерн проблемы часто узнается быстрее, чем находится перебором пакетов.
- **Дешифрование TLS по файлу ключей.** Если браузер, curl или приложение на Go настроены на запись секретов сессий в файл (переменная окружения `SSLKEYLOGFILE`), Wireshark использует его для полной расшифровки TLS-трафика. Это стандартный легитимный прием отладки собственных систем, превращающий «невидимый» HTTPS в читаемый HTTP поверх шифрованного канала.

Консольный tshark предоставляет тот же механизм декодирования в скриптуемом виде: извлечение полей (`-T fields -e ip.src -e tcp.len`), пакетная фильтрация, экспорт в CSV/JSON. В связке с утилитами mergecap (объединение дампов), editcap (нарезка и редактирование), capinfos (паспорт дампа) и tcpreplay (повторная инъекция трафика) он образует полноценный конвейер автоматизированной обработки, способный разобрать дамп в гигабайты там, где графический интерфейс безнадежно зависнет.

tcpdump и язык фильтров BPF

Там, где нет графической среды — на серверах, маршрутизаторах, в контейнерах, на встроенных устройствах, — работает tcpdump: консольный захват с фильтрацией в ядре через libpcap. Его язык фильтров (BPF-синтаксис) компактнее и работает на этапе захвата, а не отображения, поэтому именно им отсекается лишнее до записи на диск:

```
tcpdump -i eth0 port 80 -w capture.pcap          # весь HTTP-трафик в файл
tcpdump -i eth0 'tcp[tcpflags] & tcp-syn != 0'    # все SYN-пакеты (сканы, рукопожатия)
tcpdump -i eth0 host 10.0.0.5 and port 443 -C 100 -W 10   # кольцевой буфер: 10 файлов по 100 МБ
```

Кольцевой буфер (`-C`/`-W`) — стандартная практика продакшена: захват ведется непрерывно, старые данные перезаписываются, и при инциденте в распоряжении инженера оказываются «последние минуты перед катастрофой». Полезные приемы: ограничение длины захватываемого пакета (`-s`), когда важны только заголовки, захват на нескольких интерфейсах одновременно, обязательная запись в файл с последующим разбором в Wireshark вместо чтения вывода в консоли. Хороший тон — документировать условия захвата: время, интерфейс, примененные фильтры, — иначе дамп теряет доказательную ценность.

Утилита состояния и активной диагностики

Захват пакетов — пассивное наблюдение, ему сопутствуют активные и статусные инструменты:

- **ss** (современная замена netstat из пакета iproute2) — мгновенный снимок состояния сокетов: установленные соединения, состояния TCP, размеры очередей отправки и приема (Send-Q/Recv-Q), активные таймеры (повторная передача, keep-alive, zero-window probe). Непустой Send-Q при живом соединении — быстрый признак того, что данные «застряли»: либо сеть не принимает, либо партнер не подтверждает. Непустой Recv-Q — приложение не читает из сокета.
- **ping / traceroute / mtr** — активные зонды задержки и маршрута. mtr (1997 год) объединяет трассировку с многократными опросами каждого перехода, показывая распределение задержек и процент потерь по всему пути — часто этого достаточно, чтобы локализовать проблемный участок у провайдера.
- **iperf3** — измерение достижимой пропускной способности между двумя точками: TCP и UDP-режимы, параллельные потоки, двусторонние тесты, отчет о джиттере и потерях для UDP. Позволяет отделить «сеть не дает больше» от «приложение не может больше».
- **dig / nslookup** — ручные DNS-запросы к конкретным серверам с полным разбором ответа: коды возврата, флаги, TTL, время ответа. Незаменимы, когда нужно отделить проблему резолвинга от проблемы соединения.
- **curl с форматом вывода таймингов** (`-w`) — фактически микроанализатор одной HTTP-транзакции: раскладывает общее время на фазы DNS, TCP-рукопожатия, TLS-рукопожатия, время до первого байта (TTFB) и передачу тела. Один запуск curl часто заменяет минуту листания дампа.

Где захватывать: точки наблюдения

Выводы аналитика зависят от точки наблюдения, и выбор этой точки — первое методологическое решение. Захват на клиенте показывает трафик таким, каким его видит пользователь (включая все промежуточные задержки), захват на сервере — каким его видит приложение (без клиентской сети), захват в промежуточной точке (зеркало порта, TAP) — картину участка между ними. В идеальном случае при сложной проблеме снимают дампы одновременно на обоих концах: расхождение между ними и есть проблема.

Способы организации точки наблюдения различаются по надежности и цене:

- **Зеркалирование портов (SPAN)** на коммутаторах — самый доступный способ, но с оговорками: под нагрузкой коммутатор может терять зеркалируемые пакеты, направление иногда зеркалируется асимметрично, а сам факт потерь не всегда заметен в дампе.
- **Аппаратные ответвители (TAP)** — пассивные устройства, дающие полную копию трафика в обе стороны без влияния на канал, стандарт для ответственных измерений.

- **Сетевые пакетные брокеры** — агрегируют трафик с нескольких TAP/SPAN, фильтруют, дедуплицируют и распределяют копии по инструментам (IDS, анализаторы), применяются в крупных ЦОД.
- **Облачные зеркала** — у крупных провайдеров есть собственные механизмы: VPC Traffic Mirroring (AWS), Packet Mirroring (GCP), vTAP (Azure). В контейнерных средах захват ведут на хосте: интерфейсы veth-пар, подключение к сетевому пространству имен контейнера через nsenter.
- **Высокоскоростной захват** на каналах 10–100 Гбит/с требует обхода ядра (DPDK, PF_RING, AF_XDP) и специализированных систем хранения, обычный tcpdump на таких скоростях теряет пакеты, и аналитик обязан проверять статистику потерь самого захвата, чтобы не принять артефакт инструмента за свойство сети.

Потоковая телеметрия: NetFlow, IPFIX, sFlow

Полный захват отвечает на вопрос «что именно произошло», но не отвечает на вопрос «что происходило в сети последние три месяца» — объем не позволяет хранить дампы так долго. Эту нишу закрывает потоковая телеметрия: маршрутизаторы и хосты экспортируют агрегированные записи о потоках — пятикорневой набор (IP-адреса и порты источника и получателя, протокол) плюс объемы байтов и пакетов, метки времени, поля маршрутизации.

NetFlow (Cisco, с 1996 года, шаблонизируемая версия 9 — с 2003–2004 годов) и его открытый преемник IPFIX (RFC 7011, 2013) стали стандартом фактического сбора, sFlow (RFC 3176) дополняет картину статистической выборкой пакетов и счетчиками интерфейсов. На потоковых данных решаются задачи, недоступные pcap в масштабе сети: поиск аномалий потребления полосы, выявление сканирований и горизонтального перемещения атакующего, учет трафика по подразделениям, ретроспективный ответ «кто говорил с этим узлом до инцидента». Практическая иерархия такова: телеметрия потоков работает постоянно и служит триггером, а полный захват включается точечно — для сбора доказательств.

Программируемый анализ: от Scapy до eBPF

Когда готовых инструментов не хватает, в дело вступает программируемый анализ. Scapy (Филипп Бьонди, с 2003 года) позволяет конструировать и отправлять произвольные пакеты, собирать собственные зонды и тестировать поведение систем на «неправильных» входных данных — фактически это лаборатория протоколов на Python. Zeek превращает трафик в структурированные журналы событий (connection.log, dns.log, http.log, ssl.log), поверх которых пишутся скрипты обнаружения на собственном языке — подход, который лег в основу современных платформ NDR (Network Detection and Response).

Отдельная революция — eBPF. Программы, исполняемые в ядре Linux, позволяют регистрировать события транспортного стека (создание и закрытие соединений, повторные передачи, изменения окна, задержки в очередях) и агрегировать их без захвата пакетов, с пренебрежимо малыми накладными расходами. Инструменты

семейства bcc/bpftrace (tcplife, tcpretrans, tcptop) дают мгновенную статистику TCP-поведения работающих процессов, Cilium Hubble строит карты потоков в кластерах Kubernetes. Для продакшена, где непрерывный pcap-захват невозможен, eBPF стал основным способом «видеть транспортный уровень всегда».

TCP — сердце анализа

На TCP приходится львиная доля диагностических задач: это и основной транспорт веба, и протокол с самой сложной динамикой поведения. Ключ к пониманию — смена оптики: TCP следует рассматривать не как обмен пакетами, а как замкнутую систему автоматического управления, которая непрерывно измеряет сеть (через подтверждения и потери) и подстраивает под нее скорость передачи. Все «странное поведение» TCP — паузы, всплески, обвалы скорости — суть работа этой системы управления, и аналитик, знающий ее логику, читает дамп как протокол ее решений.

Жизненный цикл соединения

Установка соединения — трехэтапное рукопожатие:

```
SYN → SYN-ACK → ACK
```

Уже на этом этапе дамп дает однозначные диагнозы. SYN без какого-либо ответа — пакеты отбрасываются по пути (файрвол с политикой drop, «черная дыра» маршрутизации, недоступный хост), молчание отличается от отказа именно отсутствием пакетов. Немедленный RST в ответ на SYN — порт закрыт, или промежуточное устройство активно отклоняет соединение (политика reject). SYN-ACK, дошедший до сервера, но не подтвержденный, — проблема на обратном пути или на клиенте. Задержка между SYN и SYN-ACK — это «чистое» время оборота до сервера, первое измерение RTT в каждом соединении.

Флаги TCP — азбука анализа:

- **SYN** — запрос на установку соединения, начало отсчета порядковых номеров.
- **ACK** — подтверждение полученных байтов, присутствует почти в каждом пакете установленного соединения.
- **FIN** — штатное закрытие направления передачи, полный разрыв соединения — это обмен двумя парами FIN/ACK.
- **RST** — аварийный сброс: немедленное уничтожение соединения без штатных процедур. Кто отправил RST (сервер при закрытом сокете, файрвол, балансировщик по таймауту, система фильтрации) — ключевой вопрос, подсказками служат TTL и IP-идентификатор, отличающиеся от остального трафика соединения.
- **PSH** — запрос немедленной передачи данных приложению, **URG** — срочные данные. Встречаются реже, но аномальные комбинации флагов (только FIN, SYN+FIN, FIN+PSH+URG, так называемый «Xmas»-пакет, пакет вообще без флагов — NULL)

практически не встречаются в легитимном трафике и являются признаком сканирования.

Штатное закрытие порождает состояние TIME_WAIT на той стороне, которая первой отправила FIN: сокет удерживается удвоенное время жизни сегмента (2×MSL, типично 60–120 секунд), чтобы запоздалые дубликаты старых пакетов не попали в новое соединение с теми же номерами портов. Множество TIME_WAIT-состояний на балансировщике или прокси — не патология, а следствие его работы, патологией является исчерпание диапазона локальных портов при таком сценарии, которое лечится настройкой повторного использования сокетов и keep-alive.

Отдельного упоминания заслуживают SYN-cookies — техника защиты от SYN-флуда, предложенная Дэном Бернштейном в 1996 году: сервер не выделяет ресурсы на полуоткрытое соединение, а кодирует всю необходимую информацию в порядковом номере SYN-ACK, восстанавливая ее из ACK клиента. Если SYN-флуд все же исчерпал очередь полуоткрытых соединений, в дампе видны повторные передачи SYN-ACK и новые SYN без ответа, а на сервере — переполнение SYN-очереди в статистике ядра.

Номера, подтверждения и окна

Порядковый номер (sequence number) в TCP — это не номер пакета, а номер первого байта полезной нагрузки сегмента в общем потоке, номер подтверждения (acknowledgment number) — номер байта, который получатель ожидает следующим. Кумулятивная природа подтверждений означает: ACK с номером N подтверждает все байты до N−1 включительно. Умение читать эти два поля автоматически — базовый навык, без которого невозможно понять ни повторные передачи, ни упорядочивание, ни паузы.

Окно приема (receive window) сообщает, сколько байтов отправитель вправе передать до получения подтверждения, — это управление потоком (flow control), защита получателя от переполнения. Поскольку базовое 16-битное поле окна ограничивало объем 64 КБ, для быстрых каналов с большой задержкой еще в 1992 году ввели масштабирование окна (RFC 1323, ныне RFC 7323): множитель согласуется при рукопожатии, и окно достигает гигабайтов. Объем данных «в полете» (in-flight) ограничен минимумом из окна приема и окна перегрузки (cwnd) — и диагностика всегда начинается с вопроса, какое из двух ограничений сдерживает передачу в данный момент: если отправитель уперся в окно приема, проблема у получателя, если в окно перегрузки — в сети или в алгоритме ее оценки.

RTT: мера сети и ее ловушки

Время оборота (round-trip time) — центральная метрика транспортного уровня. В дампе оно измеряется несколькими способами: интервал SYN → SYN-ACK (чистый сетевой RTT без участия приложения), интервалы «данные → подтверждение» по меткам времени TCP (опция timestamps), встроенные оценки Wireshark. TCP использует RTT для вычисления

таймера повторной передачи (RTO) — и здесь кроется ловушка: RTO никогда не опускается ниже сотен миллисекунд, поэтому потеря одного пакета, обнаруженная только по таймеру, обходится соединению в 200+ мс простоя даже в сети с RTT 5 мс.

Принципиальное различие, которое аналитик обязан держать в голове: стабильный RTT характеризует сеть, а пауза между запросом и ответом характеризует сервер. Если в транзакции подтверждение запроса пришло через 50 мс, а ответ начал передаваться через 300 мс после запроса, то сеть обеспечила 50 мс, а 250 мс «думало» приложение — и никакие сетевые оптимизации эту задержку не устранят.

Потери и повторные передачи

Повторные передачи — главный сигнал неблагополучия канала. Wireshark помечает их явными флагами, и существуют разные механизмы, которые стоит различать:

- **Повторная передача по таймеру (RTO)** — подтверждение не пришло вовсе, самый дорогой сценарий: соединение замирает на сотни миллисекунд, окно перегрузки сбрасывается до минимума.
- **Быстрая повторная передача (fast retransmit)** — получатель шлет дублирующиеся подтверждения (duplicate ACK): каждый следующий сегмент после потерянного провоцирует повторный ACK с тем же номером ожидания. По правилу трех дублей (RFC 5681) отправитель повторяет сегмент, не дожидаясь таймера, и снижает скорость мягче. Последовательность одинаковых ACK в дампе — прямой сигнал «здесь потерян сегмент».
- **SACK** — избирательные подтверждения (RFC 2018, 1996): получатель сообщает, какие блоки байтов уже получены, позволяя повторить только действительно утраченные сегменты. Блоки SACK в заголовках — точная карта потерь.

Критически важно отличать истинные потери от мнимых (spurious retransmission): при сильном переупорядочивании или резком росте задержки отправитель может повторить сегмент, который на самом деле не потерян. Помогает анализ TTL, IP ID и меток времени: у повторно переданного по таймеру сегмента и у его «оригинала» задержавшаяся копия приходит позже повторной, и подтверждение накрывает обе. Ошибка в обе стороны дорого стоит: приняв переупорядочивание за потери, инженер будет искать несуществующую проблему канала, приняв потери за переупорядочивание — пропустит реальную перегрузку.

Цена потерь для производительности нелинейна. В соединении с включенным масштабированием окна единичная потеря в фазе медленного старта отбрасывает скорость передачи к началу, регулярные потери даже доли процента на высокоскоростных каналах способны урезать эффективную пропускную способность в разы. Отсюда практическое правило: потери и повторные передачи — первое, что проверяется при жалобах «сеть медленная», и только после их исключения имеет смысл искать причину глубже.

Zero Window: когда виновато приложение

Нулевое окно (window size = 0) — сообщение получателя «мой буфер полон, больше не присылай ничего». Это точка, где транспортный анализ переходит в анализ приложений: сеть доставила данные, но принимающее приложение не успевает их прочитать из сокета. За нулевым окном следуют зонды (zero window probe) — отправитель периодически шлет однобайтовые сегменты, проверяя, не открылось ли окно, дамп, заполненный парами «зонд — нулевой ACK», однозначно указывает на деградацию приемной стороны: перегруженный сервер базы данных, зависший обработчик, нехватка потоков чтения.

Смежный случай — Window Full: отправитель исчерпал разрешенный объем данных в полете и ждет подтверждений. Если при этом окно приема велико, значит ограничение создало окно перегрузки — узкое место в сети, если окно приема мало или нулевое — виноват получатель. Эта пара диагнозов (Zero Window vs Window Full) локализует проблему между приложением и сетью быстрее любого другого приема.

Исторически сюда же относится синдром глупого окна (silly window syndrome): получатель, читающий данные мелкими порциями, рекламирует крошечные окна, провоцируя передачу неэффективных мелких сегментов. Стандартные решения — алгоритм Дейва Кларка на стороне получателя и алгоритм Нейгла на стороне отправителя — знакомы каждому, кто сталкивался с «мелкопакетным» трафиком, их взаимодействие породило отдельный класс трудноуловимых задержек, о котором пойдет речь ниже.

Управление перегрузкой в живом трафике

Алгоритмы, созданные Джейкобсоном в конце 1980-х (Tahoe, затем Reno), заложили схему, живущую до сих пор:

- **Медленный старт (slow start)** — окно перегрузки удваивается каждый RTT, начиная с нескольких сегментов, в дампе это выглядит как все более плотные пачки пакетов и резко растущая пропускная способность, пока соединение не упрется в ограничение или не произойдет потеря.
- **Избегание перегрузки (congestion avoidance)** — после достижения порога (ssthresh) окно растет линейно, трафик становится равномерным.
- **Реакция на потерю** — быстрая повторная передача уменьшает окно вдвое (мультипликативное снижение), потеря по таймеру сбрасывает его почти до нуля, далее цикл роста повторяется. График скорости одного соединения — характерная «пила»: рост, обвал, рост.

Современные реализации развили эту базу. **CUBIC** (2008 год, алгоритм по умолчанию в Linux начиная с ядра 2.6.19) описывает рост окна кубической функцией времени: после потери окно быстро возвращается к прежнему уровню, а затем растет медленно в его окрестности и агрессивно — за его пределами. Это делает CUBIC эффективным на длинных каналах с большой задержкой, где линейный рост Reno добирался бы до полной скорости минуты. **BBR** (Google, 2016 год, публикация Кардуэлла, Ченга, Ганна, Егане и Джейкобсона в ACM Queue) принципиально меняет философию: вместо реакции на потери

алгоритм строит явную модель канала — непрерывно измеряет пропускную способность узкого места (bottleneck bandwidth) и минимальный RTT (round-trip propagation time) и поддерживает объем данных в полете, равный их произведению. Потери перестают считаться сигналом перегрузки: на каналах с фоновыми потерями (спутник, мобильные сети, трансконтинентальные магистрали) BBR показывает существенный выигрыш — по данным Google, его внедрение дало порядка 4% среднего прироста пропускной способности для сервисов компании а на отдельных дальних маршрутах прирост доходил, по опубликованным данным, до двух-двадцати пяти раз. Расплатой стали споры о честности: BBR-потоки могут занимать больше полосы в конкуренции с потеря-ориентированными алгоритмами, и вторая версия (BBRv2, с 2019 года) добавила учет сигналов перегрузки — ECN и потерь — чтобы снизить агрессивность.

Отдельный механизм — явное уведомление о перегрузке (ECN, RFC 3168, 2001): маршрутизатор вместо отбрасывания пакета помечает бит в IP-заголовке, а получатель возвращает отметку отправителю в подтверждении. В дампе ECN-метки (CE) видны в поле DSCP и в флагах TCP — это прямое свидетельство того, что очередь на пути начала заполняться, полученное без потерь. ECN десятилетиями оставался нишевым, но сегодня его включают крупные облачные провайдеры внутри своих ЦОД, и в их трафике анализ отметок становится частью рутины.

Практический вывод для аналитика: прежде чем объявлять «сеть виноватой», следует понять, какой алгоритм управления перегрузкой работает (на Linux — sysctl-параметр `tcp_congestion_control`, в дампе — характерные паттерны роста окна) и соответствует ли наблюдаемое поведение его логике. То, что выглядит как хаотичные обвалы скорости, часто оказывается штатной работой Reno в сети с единичными потерями, то, что выглядит как «сеть не дает разогнаться», — особенностью медленного старта на коротких транзакциях, которым просто не хватает RTT на разгон.

Тонкие взаимодействия: Nagle и delayed ACK

Два независимых оптимизирующих механизма, созданных с разницей в пять лет, при совместной работе дают один из самых частых трудноуловимых дефектов производительности. Алгоритм Нейгла (RFC 896, 1984) предписывает отправителю накапливать мелкие сегменты: пока предыдущие данные не подтверждены, новые мелкие порции буферизуются и уходят одним пакетом — мера против засорения сетей «телетайпным» трафиком из однобайтовых пакетов. Отложенные подтверждения (delayed ACK, RFC 1122, 1989) предписывают получателю не спешить с ACK в надежде отправить его вместе с ответными данными либо объединить два подтверждения в одно — типичная задержка составляет 40 мс в Linux и до 200–500 мс в других стеках.

Патология возникает в диалоге «запрос — ответ» на мелких пакетах: клиент, подчиняясь Нейглу, отправил небольшой запрос и ждет подтверждения, прежде чем отправить остаток, сервер, подчиняясь отложенным ACK, ждет данные приложения, прежде чем подтвердить. Обе стороны ждут друг друга, пока не сработает таймер, — и каждая транзакция получает регулярную паузу в десятки-сотни миллисекунд. Симптом в дампе

узнаваем: мелкие запросы, за ними странные равномерные паузы (~40 или ~200 мс), затем продолжение. Лечится отключением Нейгла на отправителе (`TCP_NODELAY` — стандарт для интерактивных протоколов вроде SSH и для большинства RPC) либо быстрыми подтверждениями на получателе. Класс задержки «ровно 40 мс на каждой операции» — почти всегда этот механизм, и его распознавание экономит дни поисков «проблем в сети».

Переупорядочивание и асимметричные маршруты

Пакеты одного потока могут приходить не по порядку (1 → 3 → 2 → 4): многопутевая маршрутизация, агрегация каналов (LAG), балансировка ECMP на разных переходах — все это легальные причины переупорядочивания. TCP реагирует на него так же, как на потерю, — дублирующимися подтверждениями и возможной ложной повторной передачей. В Wireshark события помечаются как Out-of-Order, отличить их от настоящих повторных передач помогают TTL и IP ID: у переупорядоченного пакета IP ID продолжает монотонный ряд, у повторной передачи — повторяет исходный. Умеренное переупорядочивание — повод не для паники, а для проверки: не слишком ли агрессивно настроены пороги (reordering), и не «наказывает» ли алгоритм перегрузки сеть за безобидную многопутность.

Более коварна асимметричная маршрутизация, при которой прямой и обратный трафик соединения идут разными путями. Для захвата это означает, что точка наблюдения на одном маршрутизаторе видит лишь половину диалога: подтверждения «пропадают», RTT выглядит аномальным, повторные передачи кажутся массовыми. Признак асимметрии — односторонний дамп при заведомо живом соединении. Вывод методологический: прежде чем интерпретировать статистику потерь и задержек, нужно убедиться, что наблюдение охватывает оба направления.

Произведение «полоса × задержка»

Максимальный объем данных, который соединение может держать «в полете», ограничен произведением пропускной способности на RTT (bandwidth-delay product, BDP). Для канала 1 Гбит/с с задержкой 100 мс (типичная трансконтинентальная трасса) BDP составляет 12,5 МБ — столько байтов должно непрерывно находиться в пути, чтобы канал был загружен. Одиночное TCP-соединение достигает этого объема только при соответствующих окнах и исправном медленном старте, приложения, открывающие соединение на каждую операцию, физически не могут быстро передать большой файл через океан — и это не «проблема сети», а следствие законов распространения сигнала.

Отсюда три практических следствия. Первое: медленная передача на дальних трассах часто лечится не «добавлением полосы», а настройкой протокола — масштабированием окон, сессиями с возобновлением, параллельными потоками, алгоритмами вроде BBR. Второе: измерение скорости одним коротким тестом бессмысленно — соединение не успевает выйти из медленного старта, измерять нужно на объемах, многократно превышающих BDP. Третье: оценка «сколько RTT нужно транзакции» — первичный

инструмент проектирования: сокращение числа оборотов (TLS 1.3 вместо 1.2, QUIC с 0-RTT, переиспользование соединений) на дальних каналах дает больший выигрыш, чем ускорение сервера на десятки миллисекунд.

Прикладной уровень: HTTP, DNS, TLS

Транспортный уровень объясняет, как данные доходят, прикладной объясняет, зачем они вообще отправлялись. Веб-трафик дает аналитику богатейший набор наблюдаемых событий: здесь каждая пользовательская операция распадается на цепочку протокольных фаз, каждую из которых можно измерить отдельно.

HTTP: анатомия транзакции

Базовая единица анализа — транзакция «запрос — ответ»: метод и URL в запросе, код состояния и тело в ответе. Коды ответов — первый ориентир: 4xx указывают на проблему на стороне клиента или логики приложения (неверный путь, отсутствующая авторизация), 5xx — на сбой серверной стороны (исключение в коде, недоступность базы данных, таймаут балансировщика). Время до первого байта ответа (TTFB, time to first byte) — ключевая метрика: оно включает обработку запроса приложением и ожидание зависимостей (база, кэш, внешние API). Большой TTFB при быстром TCP-рукопожатии — почти однозначный вердикт «тормозит backend».

Полезно раскладывать полную задержку загрузки ресурса на фазы: разрешение DNS, TCP-рукопожатие, TLS-рукопожатие, TTFB, передача тела. Именно так показывает транзакцию waterfall-диаграмма в инструментах разработчика браузера и формат HAR, тот же расклад извлекается из pcap и из таймингов curl. Сопоставление фаз мгновенно локализует проблему: долгая фаза DNS — к резолверу, долгое TLS — к криптографии и RTT, долгая передача тела — к полосе или серверу отдачи. Отдельного внимания заслуживают цепочки перенаправлений: каждое 301/302 добавляет полный цикл «запрос — ответ» (а иногда и новое соединение), и каскад из трех-четырех редиректов незаметно съедает секунды — в дампе он виден как серия коротких транзакций до начала реальной работы.

Переиспользование соединений (keep-alive) принципиально меняет картину: в HTTP/1.1 браузер поддерживает до шести параллельных TCP-соединений к одному хосту и переиспользует их для десятков запросов. Если в дампе видно новое рукопожатие на каждый запрос — конфигурация или промежуточное устройство ломают keep-alive, и каждая операция платит полную цену установки соединения. Конвейерная передача запросов (pipelining) в HTTP/1.1, наоборот, почти не применялась на практике — из-за блокировки во главе очереди: ответ на первый запрос задерживал все последующие в том же соединении.

HTTP/2: мультиплексирование и его цена

HTTP/2 вырос из экспериментального протокола SPDY (Google, 2009–2012 годы), был стандартизирован как RFC 7540 в 2015 году и уточнен в RFC 9113 (2022). Его центральная идея — мультиплексирование: множество независимых потоков (streams) в рамках одного

TCP-соединения, с приоритезацией и собственным управлением потоком на уровне кадра (WINDOW_UPDATE), со сжатием заголовков HPACK. Одно соединение вместо шести устраняет конкуренцию рукопожатий и позволяет серверу видеть всю картину загрузки страницы.

Но у мультиплексирования есть цена, и она проявляется именно в анализе трафика. Все потоки HTTP/2 делят один TCP-канал, а TCP обеспечивает строгий порядок доставки байтов: потеря одного сегмента останавливает выдачу данных приложению по всем потокам, даже тем, чьи данные уже прибыли, — это транспортная блокировка во главе очереди (head-of-line blocking). В дампе картина характерна: единственное соединение, десятки перемежающихся HTTP-запросов, и внезапное общее замирание, совпадающее с повторной передачей. Один потерянный пакет стоимостью в RTO замораживает всю страницу. Парадокс HTTP/2: протокол создавался, чтобы снизить влияние задержек, а в каналах с потерями он иногда работает хуже HTTP/1.1 с его независимыми соединениями — те хотя бы деградируют частично.

HTTP/3 и QUIC: транспорт поверх UDP

Ответом на транспортную блокировку стал QUIC — протокол, начатый Google в 2012 году и стандартизированный IETF в мае 2021 года (RFC 9000 — транспорт, RFC 9001 — интеграция TLS, RFC 9002 — обнаружение потерь и управление перегрузкой, версия 2 — RFC 9369, 2023). Прикладной профиль поверх него — HTTP/3 (RFC 9114, 2022). QUIC реализует надежную потоковую доставку непосредственно поверх UDP:

- **Независимые потоки без блокировки во главе очереди** — потеря в одном потоке не задерживает остальные: упорядочивание выполняется на уровне потока, а не соединения.
- **Встроенное шифрование** — TLS 1.3 является частью протокола, а не слоем над ним: рукопожатие QUIC совмещено с транспортным, отсюда старт за один оборот, а при возобновлении сессии — за ноль (0-RTT).
- **Миграция соединений** — соединение идентифицируется не четверкой «адреса-порты», а идентификатором (Connection ID), поэтому смена сети (Wi-Fi → сотовая связь) не рвет сессию.
- **Улучшенная оценка RTT и номеров пакетов** — защищенная от подделки монотонная нумерация устраняет двусмысленность повторных передач, мучившую TCP.

К середине 2020-х годов HTTP/3 поддерживают все основные браузеры и крупнейшие CDN, по измерениям Cloudflare, на его долю приходится уже порядка четверти-трети HTTP-запросов в их сети, и доля продолжает расти.

Для анализа трафика QUIC — одновременно подарок и вызов. Подарок: исчезли артефакты TCP-уровня, а метрики RTT и потерь доступны точнее. Вызов: шифруется практически все — не только полезная нагрузка, но и большая часть заголовков (защита заголовков, header protection), а ключи по умолчанию никто не отдает. Wireshark декодирует QUIC и HTTP/3 только при наличии файла ключей (тот же `SSLKEYLOGFILE`) или отладочных сборок, в

производственном трафике аналитик работает с тем, что осталось видимым: UDP-пакеты на порту 443, размеры и тайминги пакетов, идентификаторы соединений, первоначальный незашифрованный заголовок версии, паттерны подтверждения. Фактически анализ QUIC — это анализ поведения по форме трафика, и эта тенденция будет усиливаться. Есть и инфраструктурная проблема: промежуточные устройства, оптимизированные под TCP, нередко режут или занижают приоритет UDP, из-за чего клиенты откатываются на TCP — в дампе такой откат виден как попытка QUIC, затем новое TCP-рукопожатие к тому же хосту, и это сигнал проверить UDP-политики на пути.

DNS: невидимый виновник задержек

Огромная доля жалоб «сервер тормозит» при разборе оказывается проблемой DNS: разрешение имени предшествует каждому новому соединению, и его стоимость целиком добавляется к задержке первой транзакции. Что искать в дампе:

- **Время ответа** (`dns.time` в Wireshark): здоровый резолвер отвечает за единицы миллисекунд, десятки и сотни — признак деградации.
- **Повторы одного и того же запроса**: клиент переспрашивает, потому что ответ потерян или резолвер молчит, каскад повторов с растущими интервалами — классическая картина недоступности DNS.
- **Коды возврата**: NXDOMAIN (имя не существует) там, где оно должно существовать, — признак устаревшего кэша, перехвата DNS провайдером или ошибки конфигурации, SERVFAIL — сбой резолвинга выше по цепочке.
- **Флаг усечения (TC)**: ответ не влез в UDP-дейтаграмму, клиент повторяет запрос уже по TCP — дополнительная задержка, часто связанная с крупными DNSSEC-подписями.
- **Цепочки CNAME**: каждое каноническое имя может потребовать нового запроса, домен, обвешанный пятью CNAME до CDN и сервисов аналитики, резолвится в пять последовательных обращений.
- **Слишком малые TTL**: настройки, заставляющие клиента резолвить имя на каждое соединение, слишком большие — проблема при смене адресов.

DNS — еще и зона, где «шторм» трафика служит симптомом чужой проблемы: массовые ретраи тысяч клиентов при отказе резолвера выглядят как DDoS на DNS-сервер. И, как и везде, тренд на шифрование дошел до DNS: DNS-over-TLS (RFC 7858, порт 853) и DNS-over-HTTPS (RFC 8484, поверх стандартного 443) закрывают запросы от наблюдения в сети — со всеми вытекающими последствиями и для приватности пользователей, и для видимости трафика операторами, и для систем фильтрации.

TLS: стоимость рукопожатия и остатки видимости

TLS определяет «цену входа» каждого защищенного соединения. В TLS 1.2 (и ранее) рукопожатие требовало двух оборотов сверх TCP-рукопожатия, TLS 1.3 (RFC 8446, 2018) сократил его до одного оборота (1-RTT), а при возобновлении сессии — до нуля (0-RTT, ценой риска повторного использования перехваченных ранних данных). Суммарная арифметика первого обращения к новому HTTPS-сайту на дальней трассе: DNS (1+ запрос)

+ TCP (1 RTT) + TLS (1 RTT) + запрос-ответ (1 RTT) — и при RTT 150 мс только протокольная часть съедает полсекунды до первого байта. Вывод «HTTPS всегда дороже HTTP» верен именно для первого соединения, переиспользование сессий и билетов (session resumption) снижает разницу почти до нуля, а TLS 1.3 с 0-RTT может сделать защищенное соединение быстрее незащищенного аналога прежних версий.

Что аналитик видит в TLS-трафике без ключей? Значительно больше, чем кажется:

- **Client Hello**: поддерживаемые версии, наборы шифров (cipher suites), расширения, эллиптические кривые — и, главное, SNI (Server Name Indication), то есть доменное имя запрашиваемого сайта в открытом виде. SNI — основной источник осведомленности о посещаемых ресурсах для всех, кто наблюдает канал, именно вокруг его закрытия ведутся современные баталии приватности (см. ниже).
- **Сертификатная цепочка** (в TLS 1.2 — открыто, в 1.3 сообщение Certificate уже зашифровано, но сертификат сервера все равно передается при рукопожатии и может быть получен наблюдателем отдельным запросом): кто выдал сертификат, на какие домены, срок действия.
- **ALPN**:согласованный прикладной протокол (h2, http/1.1) — видно, каким протоколом пойдет обмен.
- **Хронометраж и объемы**: длительность рукопожатия, размеры записей, ритм обмена — материал для поведенческого анализа.

Отпечатки клиентов: JA3 и JA4

Наблюдение за параметрами Client Hello породило целое направление — TLS-отпечатки (fingerprinting). Метод JA3, предложенный инженерами Salesforce (Джон Олтхаус и коллеги) в 2017 году, вычисляет хеш от упорядоченного набора полей Client Hello — версии, списка шифров, расширений, эллиптических кривых и форматов точек. Разные TLS-стеки формируют разные наборы, поэтому хеш устойчиво отличает Chrome от Firefox, curl от Python-библиотеки requests, а массовый ботнет — от обоих. Серверный вариант JA3S отпечатывает ответ сервера. В 2023 году появилось семейство JA4+ (FoxIO): модульные отпечатки для TLS (JA4), HTTP-заголовков (JA4H), сертификатов (JA4X), TCP-параметров (JA4T) и других протоколов — с человечески читаемой структурой вместо «слепого» хеша.

Применения двояки. Защитники блокируют нежелательных клиентов (скраперов, ботов) по отпечатку, даже если те маскируют User-Agent, и выявляют вредоносное ПО: многие C2-фреймворки и скриптовые клиенты имеют характерные, известные отпечатки. Наблюдатели (в том числе цензурные системы) используют отпечатки для идентификации инструментов обхода блокировок. Атакующая сторона отвечает рандомизацией параметров рукопожатия. Так шифрование содержимого породило новый слой анализа — по форме, а не по содержанию, — и этот слой будет только расти.

Классические сетевые проблемы и их паттерны

Ценность опыта в анализе трафика состоит в узнавании паттернов: большинство проблем проявляется стандартными сигнатурами, которые опытный инженер опознает в первые секунды просмотра. Ниже — канонический набор таких сигнатур с механизмами их возникновения.

Потери пакетов

Признаки: повторные передачи, дублирующиеся ACK, пробелы в блоках SACK, экспоненциальные паузы RTO. Причины делятся на два больших класса, и их важно различать. Перегрузочные потери (tail drop в переполненной очереди) коррелируют с пиками нагрузки: они приходят всплесками в моменты, когда интенсивность трафика максимальна, и сопровождаются реакцией алгоритмов перегрузки. Случайные потери (ошибки на физической среде — деградированный оптический линк, зашумленный радиоканал Wi-Fi, дефектный кабель) распределены равномерно во времени и не связаны с объемом трафика, их выдают растущие счетчики CRC-ошибок на интерфейсах. Различение важно, потому что лечение противоположно: перегрузку лечат управлением очередями и полосой, физику — ремонтом линка, алгоритмы же перегрузки трактуют любую потерю как перегрузку и на зашумленной радиолинии бессмысленно душат скорость — случай, где выигрывают модель-ориентированные алгоритмы вроде BBR.

Задержка и джиттер

Высокий базовый RTT объясняется геометрией (длинные трассы, спутниковые каналы, обходные маршруты через точки обмена) и очередями. Джиттер — вариация задержки между пакетами одного потока — не менее важен: интерактивные приложения (голос, видео, удаленный доступ, игры) страдают от неравномерности сильнее, чем от абсолютной величины. В дампе джиттер виден как «плавающие» интервалы между подтверждениями при стабильном среднем, измеряется он и активно — UDP-режимом iperf3, mtr с длительным наблюдением. Причинами чаще всего выступают общие очереди с разнородным трафиком (см. bufferbloat), балансировка по неравнозначным путям и беспроводная среда с повторными передачами на уровне L2, невидимыми для IP-наблюдателя.

Bufferbloat: «здоровая» сеть, которая тормозит

Одна из самых недооцененных проблем последнего полутора десятилетий. Термин bufferbloat («раздувание буферов») ввел Джим Геттис в 2009–2011 годах, описав системный дефект: производители сетевого оборудования закладывали чрезмерно большие очереди, руководствуясь наивной логикой «больше буфер — меньше потерь». Следствие оказалось разрушительным: при насыщении канала пакеты не теряются, а накапливаются в очереди на сотни миллисекунд и даже секунды, RTT растет пропорционально загрузке, TCP не видит потерь и продолжает накачивать очередь, а интерактивный трафик задыхается. Симптоматика узнаваема: базовый ping в покое —

единицы миллисекунд, под нагрузкой (загрузка файла, видеострим) — сотни, видеозвонки рассыпаются именно тогда, когда кто-то в сети начал скачивание, ретрансляций при этом почти нет.

Лечится проблема активным управлением очередями (AQM) — дисциплинами, которые намеренно создают ранний сигнал перегрузки вместо накопления: RED (Random Early Detection, Салли Флойд и Ван Джейкобсон, 1993) отбрасывает пакеты с растущей вероятностью до заполнения очереди, CoDel (Николс и Джейкобсон, 2012) отслеживает не длину очереди, а время пребывания пакета в ней и давит ее, когда задержка превышает целевую (~5 мс), fq_codel сочетает CoDel со справедливой очередью по потокам и с 2012 года входит в ядро Linux, став стандартом для домашних маршрутизаторов (OpenWrt/SQM), разработана и более «вежливая» к TCP альтернатива PIE, и комплексная дисциплина CAKE. Для аналитика главный урок bufferbloat методологический: «потерь нет» не означает «сеть в порядке» — отсутствие потерь при растущей задержке само по себе является диагнозом.

Фрагментация и Path MTU

Каналы по пути между хостами имеют разные максимальные размеры кадра (MTU): классический Ethernet — 1500 байт, PPPoE — 1492, туннели IPsec/GRE/VXLAN съедают еще 20–50+ байт. Механизм обнаружения MTU по пути (Path MTU Discovery, RFC 1191) полагается на то, что маршрутизатор, не способный передать пакет целиком, вернет ICMP-сообщение «Fragmentation Needed». Если файрволы по пути блокируют ICMP целиком (частая практика «на всякий случай»), обнаружение ломается, и возникает «черная дыра PMTU»: TCP-соединение устанавливается (рукопожатие — мелкие пакеты), но передача данных зависает на первом же крупном сегменте — он молча отбрасывается, ICMP-уведомление не возвращается, отправитель бесконечно повторяет один и тот же большой сегмент. Сигнатура в дампе неповторима: соединение живо, ACK на рукопожатие есть, а один сегмент полного размера ретранслируется раз за разом без единого подтверждения.

Обходные пути: MSS clamping (RFC 2923) — промежуточное устройство уменьшает MSS в SYN-пакетах, чтобы стороны сразу выбрали безопасный размер сегмента, фрагментация на уровне IP (в IPv4) с ее собственными бедами — потеря одного фрагмента убивает всю дейтаграмму, а фрагментированный трафик любим атакующими для обхода фильтров, в IPv6 транзитная фрагментация запрещена вовсе, и работает только Packetization Layer PMTUD (RFC 4821), осуществляющий обнаружение зондами — что делает блокировку ICMPv6 еще более фатальной. Для аналитика правило простое: «работает мелкое, не работает крупное, соединение при этом установлено» — проверяй MTU.

Блокировки, сбросы и инъекции

Два способа, которыми соединение убивают намеренно, выглядят в дампе по-разному и означают разное. Отбрасывание (drop): SYN уходит и не возвращается ничего — молчание, так ведут себя файрволы по умолчанию и так же выглядят маршрутизационные «черные дыры». Отклонение (reject): на SYN мгновенно приходит RST или ICMP-сообщение «порт

недоступен» — так ведут себя хостовые файрволы с политикой reject и закрытые сервисы. Молчание диагностически тяжелее: по нему невозможно отличить фильтрацию от недоступности без трассировки.

Особый класс — инъекция пакетов промежуточными системами. Промежуточное устройство (система фильтрации контента, DPI-шлюз) может само сгенерировать TCP RST от имени сервера, чтобы разорвать нежелательное соединение. Первый задокументированный массовый пример такого рода описан в 2006 году исследователями Клейтоном, Мердоком и Уотсоном при изучении «Великого китайского файрвола»: система фильтрации отправляла поддельные RST обеим сторонам соединения при обнаружении запрещенных ключевых слов. В 2007–2008 годах аналогичную технику — поддельные RST для разрыва BitTorrent-сессий — применял американский провайдер Comcast, что привело к разбирательству в Федеральной комиссии по связи США и стало одним из краеугольных камней дебатов о сетевом нейтралитете. Сигнатура инъекции в дампе: два RST на одно событие — с разными TTL, разными IP ID, иногда с разными порядковыми номерами, либо RST, пришедший быстрее, чем физически возможен ответ реального сервера. Умение видеть эту разницу превращает аналитика из наблюдателя «странных обрывов» в свидетеля целенаправленного вмешательства.

CDN, балансировщики и NAT: иллюзия прямого соединения

Современный клиент почти никогда не разговаривает с «настоящим» сервером. За доменным именем стоит anycast-адрес сети доставки контента (CDN), за которым — ближайший к клиенту edge-узел, за которым — цепочка балансировщиков и только потом origin-сервер. Для анализа это означает несколько обязательных вопросов. С кем реально установлено соединение — IP в дампе принадлежит CDN, и его RTT характеризует путь до кэша, а не до источника. Почему один и тот же домен резолвится в разные адреса и почему ответы приходят с аномально малым TTL (edge-узел рядом) или, наоборот, подозрительно большим. Совпадает ли сертификат в рукопожатии TLS с ожидаемым доменом и не подменяет ли корпоративный прокси цепочку своим удостоверяющим центром. Где находится точка наблюдения относительно балансировщика — одна и та же «медленная транзакция» может быть быстрой на origin и медленной на edge или наоборот.

NAT добавляет собственный слой иллюзий: множество клиентов за одним адресом, таблицы трансляции с таймаутами (типично минуты для простаивающих TCP-соединений и десятки секунд для UDP), «помощники» ALG, переписывающие содержимое протоколов. Длительно простаивающее соединение (SSH-сессия, keep-alive пула базы данных) может быть молча выброшено из таблицы NAT, и следующий пакет породит RST «из ниоткуда» — в дампе это выглядит как внезапный сброс живого соединения без каких-либо предшествующих ошибок. Признаки NAT в трафике: расхождение между адресами в заголовках и в прикладных данных (например, в SIP или FTP), «прыгающие» TTL от одного источника, совпадение времени жизни множества соединений с типовыми таймаутами трансляторов.

Широковещательный и многоадресный трафик: невидимая нагрузка

Не весь трафик сети — одноадресный (unicast), и служебная его часть регулярно сбивает анализ с толку. Широковещательные рассылки (broadcast) адресованы всем узлам сегмента: ARP-запросы «кто владеет этим IP», обнаружение служб (mDNS, SSDP, NetBIOS), DHCP-поиск сервера. В нормальном сегменте это фоновый шум в единицы процентов полосы, который аналитик отфильтровывает первым же фильтром отображения, но при петле в коммутационной топологии или заражении сети широковещательный шторм превращает сегмент в неработоспособный — в дампе это выглядит как лавина одинаковых кадров, забивающая канал, а лечится она протоколом spanning tree (STP) и сегментацией. Многоадресные рассылки (multicast) — видеопотоки, биржевые ленты, протокольные группы маршрутизации — управляются IGMP и требуют от инфраструктуры групповой доставки, в неподготовленной сети multicast вырождается во флуд во все порты VLAN. Практические следствия для анализа: служебный трафик нужно уметь быстро отделять от полезного (иначе статистика Conversations вводит в заблуждение), а жалобы «сеть легла» при живых серверах иногда оказываются широковещательным штормом, который виден только на канальном уровне.

Сводка паттернов

Сжатая шпаргалка, связывающая наблюдаемое с причиной:

- **SYN → тишина.** Отбрасывание файрволом, недоступность хоста, «черная дыра» маршрута.
- **SYN → RST.** Порт закрыт, сервис не слушает, активное отклонение политикой.
- **Долгий DNS, повторы запросов.** Деградация резолвера, «сервер» может быть ни при чем.
- **TCP быстр, HTTP медленный (большой TTFB).** Проблема backend: приложение, база, блокировки.
- **RTT растет под нагрузкой, потерь нет.** Bufferbloat: очередь, а не канал.
- **Соединение есть, крупные сегменты ретранслируются без ACK.** Черная дыра Path MTU.
- **Регулярные паузы 40–200 мс на мелких обменах.** Взаимодействие Nagle и delayed ACK.
- **Одно соединение HTTP/2, все замирает одновременно.** Потеря сегмента, транспортная блокировка во главе очереди.
- **Два RST с разными TTL/IP ID.** Инъекция промежуточной системой.
- **Zero window, зонды.** Принимающее приложение не читает сокет.
- **Пила скорости с обвалами.** Штатная реакция алгоритма перегрузки на потери, ищи источник потерь.
- **Много мелких соединений с почти нулевой отдачей.** Возможна «медленная» атака (Slowloris) или сломанный клиент.

Метрики качества: от QoS к QoE

Анализ трафика бессмысленен без измеримых критериев: «быстро» и «медленно» — не инженерные категории. Базовый набор метрик невелик, но каждая из них имеет тонкости, определяющие, будет ли диагноз верным.

Базовые метрики и культура измерения

- **RTT / задержка (latency).** Время оборота пакета до целевой точки. Измеряется пассивно (по рукопожатиям и подтверждениям в дампе) и активно (ping, mtr, зонды TCP-соединений). Требует указания распределения, а не только среднего.
- **Пропускная способность (throughput).** Объем данных, доставленный за единицу времени. Отличается от goodput — полезной нагрузки за вычетом заголовков, повторных передач и служебного трафика: на зашумленном канале разница достигает десятков процентов.
- **Потери (packet loss).** Доля пакетов, не дошедших до получателя. В пассивном анализе выводится из пробелов номеров и ретрансляций, активно измеряется UDP-зондами (iperf3) и длительным mtr.
- **Джиттер (jitter).** Вариация задержки, критичен для реального времени. Формально — среднее отклонение интервалов доставки, на практике важнее хвосты распределения.

Культура измерения не менее важна, чем сами метрики. Единичное измерение — не данные, данные — это распределение во времени и по потокам. Измерение в спокойный период не характеризует пик. И всякое активное измерение (ping, iperf) трактует сеть по-своему: зонды проходят теми очередями и политиками, что и обычный трафик, — или не проходят, когда ICMP фильтруется, что систематически искажает картину в пользу «идеальной» задержки.

Хвостовые задержки: почему среднее врет

Классическая работа Джеффа Дина и Луиса Андре Баррозу «The Tail at Scale» (Communications of the ACM, 2013) сформулировала проблему, давно знакомую сетевикам: в распределенной системе запрос затрагивает десятки компонентов, и его итоговая задержка определяется максимумом из их задержек. Если каждый компонент имеет 1% запросов с аномальной задержкой, то запрос, затрагивающий сто компонентов, почти наверняка попадет в хвост хотя бы одного из них. Среднее время ответа при этом выглядит прекрасно.

Применительно к трафику это означает: анализировать нужно перцентили — p50, p95, p99, p99.9. «Средний RTT 20 мс» совместим с «p99 = 1,5 с», и именно хвосты создают пользовательский опыт «тяжелых» страниц. Современная практика производительности (SRE-подход, мониторинг SLI/SLO) строится на перцентилях, и пакетный анализ следует той же дисциплине: графики распределения RTT по соединениям, доля транзакций с TTFB выше порога, хвосты времени DNS. Хвостовые задержки — это там, где живут bufferbloat,

ретрансляции по таймеру, конкуренция потоков и деградация отдельных серверов пула: единичные причины, невидимые в среднем, но определяющие худший пользовательский опыт.

QoS: как сеть управляет трафиком

Quality of Service — механизм, которым сеть гарантирует (или пытается гарантировать) параметры доставки для разных классов трафика. Исторически сложились два подхода. IntServ (RFC 1633, 1994) с протоколом RSVP предполагал резервирование ресурсов под каждый поток — элегантно, но не масштабируемо на интернет: маршрутизаторы ядра не могут хранить состояние миллионов потоков. Победил DiffServ (RFC 2474, 1998): пакет маркируется кодом DSCP в IP-заголовке на границе сети, а внутри маршрутизаторы обслуживают очереди в соответствии с классом — без состояния отдельных потоков. Голос получает приоритет, фоновые загрузки — остаточную полосу.

Внутри узла QoS реализуется дисциплинами очередей и планировщиками: приоритетные очереди (PQ), взвешенная справедливая очередность (WFQ и ее потомки SFQ/FQ), шейпинг (сглаживание всплесков до заданного профиля,как правило, через буферизацию) и полисинг (жесткое отбрасывание сверх профиля). Для аналитика QoS — объяснительная рамка: «голос чистый, а загрузка медленная» — штатное поведение приоритизации, «весь трафик медленный, DSCP обнулен на середине пути» — типичная болезнь стыков между доменами управления, где провайдер не доверяет маркировке клиента и перетирает ее. Понимание того, какие политики shaping/policering применены на пути, часто важнее измерения самого канала.

QoE: пользовательская правда

Quality of Experience — метрика иного рода: не что обеспечивает сеть, а что ощущает пользователь. Ключевой парадокс, который обязан знать каждый аналитик: сеть может быть «идеальной» по всем показателям QoS, а пользователь — недоволен. Классический пример: RTT в норме, потерь нет, полоса достаточна — но видео лагает, потому что проблема лежит в логике адаптивного стриминга, буферизации плеера, выборе CDN-узла или деградации конкретного origin-сервера, невидимой из сети доступа.

Инструментарий QoE включает: для видео — время старта, долю времени перебуферизации (rebuffering ratio), качество выбранного профиля в адаптивных системах (HLS от Apple, 2009, MPEG-DASH, стандарт ISO с 2012 года), где плеер постоянно переключает битрейт по фактической пропускной способности, для голоса — оценку MOS (ITU-T P.800), требования к односторонней задержке (рекомендация G.114: до 150 мс приемлемо, свыше 400 мс разговор распадается) и размер jitter-буфера, для веба — метрики пользовательского опыта вроде Core Web Vitals (время отрисовки крупнейшего элемента, отзывчивость, стабильность верстки), которые поисковые системы превратили в фактический стандарт качества сайтов.

Методологический вывод: анализ трафика следует начинать с воспроизведения пользовательского сценария и фиксации QoE-симптома («видео стартует 8 секунд»), и только затем спускаться к пакетам за причиной. Обратный порядок — смотреть дамп в поисках «странностей» без привязки к жалобе — рождает бесконечные списки найденных аномалий, не имеющих отношения к делу.

Анализ трафика и безопасность

Тот же навык чтения протокольного поведения, который лечит тормозящие сайты, составляет основу сетевого обнаружения атак. Большинство вторжений оставляет следы на транспортном и прикладном уровне задолго до того, как их зафиксируют конечные точки, и для целого класса угроз трафик — единственный источник сигналов.

Атаки, видимые в дампе

- **SYN-флуд.** Лавина SYN-пакетов, часто с поддельными адресами, без завершающих ACK: очередь полуоткрытых соединений переполняется, легитимные клиенты не дожидаются ответа. Сигнатура: плотный поток SYN на один порт, ретрансляции SYN-ACK со стороны сервера, отсутствие рукопожатий до конца, на сервере — переполнение SYN-очереди в счетчиках ядра. Стандартная защита — SYN-cookies (см. выше), включающаяся при переполнении очереди, ее следы — «странные» начальные номера SYN-ACK.
- **Сканирование портов.** Серия SYN к последовательным или характерным портам одного хоста либо к одному порту множества хостов, ответы — RST (порт закрыт) или SYN-ACK (открыт). Продвинутые варианты используют аномальные комбинации флагов: FIN-скан, NULL-скан (без флагов), Xmas-скан (FIN+PSH+URG) — они эксплуатируют требования RFC к реакции на некорректные сегменты и хуже детектируются простыми правилами. Медленное сканирование (пакет в минуту, распределенное по подсетям) обходит пороговые сигнатуры и выявляется только агрегацией в потоковой телеметрии.
- **Slowloris и «медленные» атаки.** Атака, опубликованная Робертом Хансеном в 2009 году: множество соединений, в которых заголовок HTTP-запроса отправляется по байту с паузами, не давая соединению ни завершиться, ни закрыться по таймауту. Пул соединений веб-сервера исчерпывается без единого «плохого» пакета. Сигнатура в трафике: десятки долгих соединений с околонулевой скоростью и регулярными мелкими дозаписями. Родственный класс — RUDY (медленная отправка тела POST-запроса) и slow-read-атаки (медленное чтение ответа, удерживающее серверный поток открытым).
- **Прикладной DDoS.** Лавина корректных HTTP-запросов — к поиску, к логину, к дорогому API. Каждый пакет легитимен, отличить атаку от флешмоба пользователей можно только поведенческой агрегацией: распределение User-Agent и TLS-отпечатков, концентрация по источникам и географии, однотипность параметров запросов, неестественная равномерность ритма. Это зона совместной работы трафик-анализа и веб-аналитики.

Отражение и усиление: рекордные DDoS

Отдельное семейство атак использует серверы с поддельным адресом источника (UDP позволяет это без рукопожатия) и протоколы, у которых ответ кратно больше запроса. Усилители: DNS (фактор ~28–54 при рекурсивных запросах ANY), NTP с командой monlist (~556x), Memcached (~51 000x — текстовый протокол без аутентификации, отвечающий мегабайтами на байты запроса). Жертва получает лавину с адресов ничего не подозревающих открытых серверов.

Две вехи, показавшие масштаб проблемы. В октябре 2016 года атака на DNS-провайдера Dyn, выполненная ботнетом Mirai из зараженных IoT-устройств (камеры, видеорегистраторы с заводскими паролями), достигала по различным оценкам 1,2 Тбит/с и на часы вывела из строя значительную часть популярных сайтов Восточного побережья США. В феврале–марте 2018 года серия атак с усилением через Memcached обрушила на GitHub 1,35 Тбит/с — на тот момент крупнейшую публично зафиксированную DDoS-атаку, атака была отражена за минуты, но продемонстрировала, насколько дешево стало покупать «рекордную» мощность: достаточно нескольких тысяч незащищенных memcached-серверов, выставленных в интернет. В дампе на стороне жертвы эти атаки выглядят как вал UDP (порт 53, 123 или 11211 — источник) от разнородных источников с фрагментами огромных ответов, защита — фильтрация по типу трафика и ограничение у владельцев усилителей, а не у жертвы.

Утечка данных и командные каналы

Похищенные данные и управление ботнетами тоже должны ехать по сети — и трафик-анализ ищет не содержимое (оно зашифровано), а ритм и форму. Биконинг (beaconing): вредоносный агент периодически «отмечается» в командном центре — соединения через фиксированные интервалы, иногда с небольшим случайным дрожанием (jitter). Ровная периодичность — сильнейший сигнал: живые пользователи и легитимные приложения так не делают, поиск регулярных интервалов во временных рядах соединений стал стандартной техникой NDR-платформ.

Туннелирование: DNS-туннели (данные кодируются в длинных метках доменных имен — инструменты класса iodine), ICMP-туннели (полезная нагрузка внутри echo-запросов), HTTPS-туннели (C2 неотличим от браузера по одному лишь факту шифрования — поэтому важны репутация назначения, SNI, TLS-отпечаток клиента: JA3 у скриптовых библиотек и C2-фреймворков характерен и хорошо каталогизирован). Объемные признаки: аномальный исходящий upload (у пользовательских сетей исходящий трафик исторически мал), ночная активность рабочих станций, обращения к редко используемым внешним адресам, «толстые» DNS-запросы.

Сниффинг, подмена и «человек посередине»

Внутри локальных сетей классика жанра — ARP-спуфинг: атакующий рассылает поддельные ARP-ответы, привязывая IP шлюза к своему MAC, и перехватывает трафик сегмента. В дампе жертвы это видно как два разных MAC-адреса для одного IP, всплески

gratuitous ARP, в статике — расхождение ARP-таблиц у соседей. Родственные техники: подставной DHCP-сервер (в предложениях — «свой» шлюз и DNS), подмена DNS-ответов (ответ приходит быстрее, чем от легитимного сервера, с другим TTL).

MITM против TLS возможен лишь тогда, когда атакующий контролирует доверенный удостоверяющий центр клиента. Легальный случай такого контроля — корпоративные прокси с инспекцией TLS: в доверенные хранилища сотрудников установлен корневой сертификат компании, прокси распаковывает HTTPS, инспектирует и упаковывает заново. В трафике это проявляется как разные цепочки сертификатов и разные JA3S для одного и того же сервиса в зависимости от точки наблюдения. Практика повсеместна и одновременно рискованна: точка инспекции становится концентратором чувствительных данных и целью атак, а ошибки ее реализации (например, неизбирательная пересылка) ломают шифрование на практике. Для аналитика важно уметь отличать такую инспекцию от настоящей атаки — по издателю сертификата и поведению инфраструктуры.

Что остается от приватности в зашифрованном трафике

Шифрование закрыло содержимое, но не закрыло поведение. Набор наблюдаемых метаданных — кто, с кем, когда, как долго, в каком объеме, в каком ритме — по-прежнему обширен: SNI выдает посещаемый домен (в незашифрованном виде до развертывания ECH), DNS-запросы — полный список обращений, объемы и тайминги — характер активности. Научные результаты показывают, насколько это серьезно: задача website fingerprinting — определение посещенного сайта по последовательности размеров и интервалов пакетов зашифрованного соединения — решается методами машинного обучения с точностью до 99% в закрытых экспериментах (Deep Fingerprinting, Sirinam и соавторы, 2018), в открытых условиях точность ниже, но все еще достаточна для целевого наблюдения. Классификаторы приложений (AppScanner и последующие работы) определяют тип ПО по форме трафика без расшифровки. Это не теоретическая угроза: именно на метаданных — записях о соединениях, потоковой телеметрии — строится основная масса коммуникационной разведки правоохранительных органов, поскольку такие данные дешевле и юридически проще в получении, чем перехват содержимого.

Ответные меры развиваются по всем фронтам: стандартизируется Encrypted Client Hello (ECH) — преемник ESNI, прячущий SNI внутри зашифрованного рукопожатия (поддержка уже разворачивается крупными CDN и браузерами, спецификация на стадии финализации в IETF), DoH/DoT закрывают DNS, защитные схемы (padding, трафик-форминг, смешивающие сети типа Tor) усложняют fingerprinting — хотя и исследования показывают, и гонка продолжается. Инженерный вывод для аналитика двоякий: с одной стороны, анализ зашифрованного трафика смещается к метаданным и машинному обучению, и это нужно уметь, с другой — сбор трафик-данных сам по себе стал объектом правового регулирования. Директива ЕС об обязательном хранении трафик-данных (2006/24/EC) была признана недействительной Судом ЕС в 2014 году (решение Digital Rights Ireland) как несоразмерное вмешательство в частную жизнь, споры о допустимых сроках и объемах

хранения метаданных продолжаются в национальных судах по сей день. Телеметрия — это не только диагностика, но и приватность, и минимизация собираемых данных должна быть частью архитектуры наблюдения.

Методология: анализ как детективное расследование

Навык чтения пакетов сам по себе не делает инженера аналитиком: аналитиком его делает метод. Трафик-расследование строится вокруг трех вопросов, на которые обязан ответить каждый разбор.

Три вопроса расследования

1. **Где проблема?** — локализация: уровень модели (L2/L3/L4/L7), узел пути (клиент, сеть доступа, магистраль, ЦОД, сервер), фаза транзакции (DNS, рукопожатие, передача).
2. **Почему?** — механизм: потери, очередь, исчерпание окна, блокировка приложения, ошибка конфигурации, вмешательство промежуточного устройства, атака.
3. **Кто отвечает?** — зона ответственности: сеть, сервер, клиент, внешняя сторона (провайдер, CDN, регулятор) или безопасность. Без третьего ответа разбор не имеет практического завершения — он не превращается в действие.

Локализация по симптому

Первичная сортировка выполняется еще до дампа — по характеру жалобы:

Симптом	Вероятный уровень	Первые проверки
Нет никакого ответа	Сеть / IP / файрвол	ping, traceroute, SYN в дампе
Соединение не устанавливается	TCP / L4	RST или молчание, состояние порта
Устанавливается, но медленно	TCP / приложение	RTT, окно, TTFB
Ошибки 4xx/5xx	Приложение	коды ответов, журналы сервера
Работает с перебоями	Стыки и состояния	NAT-таймауты, переупорядочивание, bufferbloat
Медленно «у всех, всегда»	Архитектура / емкость	базовые метрики на масштабе
Медленно «иногда у некоторых»	Хвосты и деградация узла	перцентили, per-server статистика

Эта таблица — не догма, а порядок перебора гипотез: она экономит часы, предотвращая классическую ошибку — начинать с листания пакетов, когда проблема локализуется одной командой mtr или ss.

Корреляция событий: сквозная цепочка

Высший пилотаж — связывание уровней в единую причинно-следственную цепочку: сеть → TCP → TLS → HTTP → логика приложения. Каждое звено получает свою оценку по дампу, и вердикт выносится по расхождению оценок. Канонический пример: RTT в норме, TCP-рукопожатие быстрое, TLS-рукопожатие быстрое, а первый байт HTTP-ответа приходит через полсекунды — сеть оправдана, виновата обработка запроса, следующий шаг — журналы приложения и базы данных, где обычно обнаруживается медленный запрос, блокировка или сборщик мусора. Мышление сдвигается с «пакеты идут странно» на «сервер отвечает медленно из-за блокировки базы данных» — и это уже диагноз, а не наблюдение.

Современная практика усиливает корреляцию сквозными идентификаторами. Системы распределенной трассировки (OpenTelemetry и совместимые) распространяют контекст запроса через HTTP-заголовки (traceparent), и эти заголовки видны в pcap: пакетный дамп и трассировка приложения связываются буквально одним полем. Метка времени инцидента из мониторинга сужает гигабайты захвата до минут, идентификатор соединения из журналов прокси — до одного потока. Коррелированный анализ — пакет, поток, метрика, трассировка, журнал — и есть та система наблюдаемости, к которой движется отрасль.

Практические кейсы

Кейс 1: сайт не открывается. Порядок действий становится вторым дыханием: ping — есть ли IP-связность вообще, traceroute — где обрывается маршрут, dig — резолвится ли имя и в тот ли адрес, дамп — что происходит на уровне TCP. Три исхода: SYN без ответа (drop по пути — файрвол, «черная дыра», недоступность), SYN → RST (порт закрыт или активное отклонение), рукопожатие есть, а обмена нет (проблема выше — TLS-сбой, приложение). В каждом случае ответ на вопрос «кто виноват» дается за минуты.

Кейс 2: сайт медленный. Декомпозиция по фазам: dns.time долгий — проблема резолвинга, сервер может быть ни при чем, TCP-рукопожатие долгое — география или очереди канала, TLS долгое — версия протокола (1.2 вместо 1.3), отсутствие переиспользования сессий, TTFB большой — backend, тело передается медленно при нормальном TTFB — полоса, окно, потери. Пять разных диагнозов за одной жалобой «тормозит» — и дамп различает их однозначно.

Кейс 3: соединение рвется. RST — всегда чей-то: сервер закрыл сокет (краш, рестарт, исчерпание ресурсов), файрвол отклонил по политике или таймауту состояния, NAT выбросил запись, балансировщик превысил лимит idle, либо RST инъектирован системой фильтрации. Различение — по TTL и IP ID относительно остального трафика соединения, по времени появления относительно физического RTT (инъекция «опережает» сервер), по порядковым номерам (инъектированный RST часто имеет некорректный seq и игнорируется стеком, что тоже видно — по повторной передаче после «сброшенного» соединения).

Кейс 4: видео тормозит, «сеть в порядке». Потерь нет, RTT в покое отличный — но под нагрузкой RTT растет в разы (bufferbloat домашней очереди), либо плеер переключает адаптивный профиль вниз из-за конкурентного трафика, либо деградировал конкретный edge-узел CDN, невидимый из общей статистики. Диагноз требует смотреть не на средние по каналу, а на конкретный поток в конкретное время — и на метрики QoE, а не QoS.

Визуализация: глаз быстрее перебора

Опытные аналитики смотрят графики раньше пакетов. Wireshark строит IO Graph (интенсивность трафика во времени с фильтрацией по любому выражению), TCP Stream Graphs (throughput, RTT, размер окна по времени — наследие утилиты tcptrace) и Flow Graph (диаграмма последовательности сообщения между сторонами). Паттерны узнаются мгновенно: «пила» — работа алгоритма перегрузки, ступеньки с плато на нуле — zero window, резкие вертикальные всплески одинаковых пакетов — сканирование или флуд, постепенный наклон линии RTT вверх под нагрузкой — рост очередей. То, что в списке пакетов потребовало бы тысяч строк прокрутки, на графике представляет собой одну характерную форму. Визуализация — не украшение анализа, а его первичный инструмент.

Ментальная модель эксперта

Разницу уровней квалификации проще всего показать парами формулировок. Новичок: «пакеты идут странно». Эксперт: «TCP снижает окно перегрузки из-за потерь на третьем переходе — вот дублирующиеся ACK и SACK-пробелы, вот счетчики ошибок на интерфейсе провайдера». Новичок: «сайт тормозит». Эксперт: «после TLS-рукопожатия за 40 мс backend отдает первый байт через 500 мс — сеть в порядке, смотрите базу». Новичок: «какие-то лишние пакеты». Эксперт: «RST с TTL, отличным от остального потока, пришел раньше физического ответа сервера — это инъекция промежуточной системы».

Финальный сдвиг мышления, который дает этот этап подготовки инженера: анализ трафика — это не про пакеты. Это про поведение системы под нагрузкой, реконструируемое по ее внешним проявлениям. Дамп — лог системы, который она пишет сама, не зная об этом, задача аналитика — прочитать его грамотнее, чем система успела его запутать. Показатель зрелости: способность открыть незнакомый pcap без графической оболочки (tshark или даже hex-просмотр), увидеть проблему в первые тридцать секунд и четко разделить зоны ответственности — сеть, сервер, клиент, безопасность.

Современные архитектуры: SDN, оверлеи, облака

Инструменты и паттерны, описанные выше, предполагают классическую сеть с относительно статичными маршрутами и явными точками захвата. Современные инфраструктуры размывают эти предположения, и анализ трафика должен это учитывать.

SDN: когда поведение сети программно определяемо

Программно-конфигурируемые сети (SDN) отделяют плоскость управления от плоскости пересылки (forwarding): решения о пересылке принимает централизованный контроллер (или набор контроллеров), а коммутаторы исполняют устанавливаемые им правила, интерфейс OpenFlow был предложен группой Ника Маккиоуна с соавторами в статье 2008 года, ставшей манифестом направления. Для аналитика это означает: путь пакета может меняться динамически, в обход классических протоколов маршрутизации, traceroute может не отражать реальную пересылку, «странное» поведение (петли, асимметрия, внезапные смены пути) бывает следствием логики контроллера, а не сбоя оборудования. Точкой наблюдения добавляется сам контроллер: его журналы установки потоков и статистика правил — источник данных, которого не существовало в классической сети.

Оверлеи и туннели: «матрешки» пакетов

Виртуализация перенесла сети внутрь хостов: VXLAN (RFC 7348, 2014) и Geneve инкапсулируют кадры виртуальных машин в UDP для переноса через физическую сеть (fabric), GRE и IPsec соединяют ЦОД, WireGuard и подобные решают задачи удаленного доступа, в мобильных сетях трафик смартфона путешествует внутри туннелей GTP-U от базовой станции до ядра. Следствия для анализа: внутренний трафик невидим на физических интерфейсах без учета инкапсуляции, захват нужно осознанно выбирать — до инкапсуляции (виден внутренний пакет, не виден путь) или после, offload-механизмы хоста (TSO/GSO сегментируют пакеты на отправке, GRO/LRO склеивают на приеме) искажают картину — в дампе на гипервизоре видны «суперпакеты» в десятки килобайт, которых физически в сети не было, и интерпретировать их размеры и MSS-динамику нужно с осторожностью.

Облака и контейнеры: исчезнувшие точки захвата

В облаке нет ни TAP, ни физического доступа к коммутатору, наблюдение организуется средствами провайдера (VPC Traffic Mirroring в AWS, Packet Mirroring в GCP, vTAP в Azure) либо агентами на хостах. В Kubernetes путь запроса проходит через сетевые пространства имен, veth-пары, CNI-плагин, цепочки iptables или eBPF-программы и часто через sidecar-прокси сервис-меша (Envoy в Istio и подобных). Захват «просто на интерфейсе Pod» может не показать картину целиком — правила балансировки и NAT применяются между точками захвата. Одновременно растет роль eBPF-наблюдаемости: Cilium Hubble и аналогичные инструменты строят карты потоков и L7-событий (HTTP-коды, gRPC-методы, DNS) прямо из ядра, без захвата пакетов, покрывая кластер постоянно — там, где непрерывный pcap физически невозможен по объему.

Наблюдаемость как система: место pcap в общей картине

Финальный элемент методологии — место анализа трафика в общей системе наблюдаемости. Столпы классической триады (метрики, журналы, трассировки) дополняются сетевым слоем: потоковой телеметрией (NetFlow/IPFIX/sFlow), счетчиками оборудования и точечным полным захватом. Рабочий цикл выглядит как каскад: метрики

дают сигнал («p99 вырос втрое»), потоки локализуют его («деградация на соединениях с конкретным сегментом ASN»), захват пакетов объясняет механизм («ретрансляции после изменения маршрута, RTT вырос на 80 мс»), трассировки приложения связывают с бизнес-логикой («медленные ответы конкретного сервиса за этим соединением»). Каждый слой дешевле предыдущего в хранении и дороже в детальности, искусство — запрашивать у каждого слоя ровно то, на что он способен отвечать.

Итоги

Анализ трафика — это точка сборки всего знания о сетях. В нем сходятся устройство протоколов от канального уровня до прикладного, теория управления перегрузкой, архитектура интернета с ее CDN, NAT и оверлеями, метрики производительности и практика безопасности. У него нет собственной «главы» в классических курсах — потому что оно и есть практическое применение всех остальных глав к реальным данным.

Главный переход, который совершает специалист, — от чтения пакетов к реконструкции поведения системы под нагрузкой. Пакет перестает быть объектом анализа и становится свидетельским показанием: важны не байты сами по себе, а временная линия, причинно-следственные связи и зоны ответственности, которые из них восстанавливаются. Три вопроса расследования — где, почему, кто отвечает — превращают дамп из «набора странностей» в инженерный документ.

Шифрование изменило предмет, но не отменило дисциплину. Когда содержимое закрыто, анализ смещается к метаданным и форме трафика: тайминги, объемы, ритмы, параметры рукопожатий, TLS-отпечатки. Это одновременно и расширение возможностей (поведенческий анализ, обнаружение скрытых каналов, fingerprinting), и новый вызов приватности — метаданные, как показывает и история телефонных сетей, и современная наука о website fingerprinting, способны рассказать о человеке не меньше, чем содержимое его сообщений. Инженеру приходится держать в голове обе стороны: его инструменты наблюдения работают и на диагностику, и на слежение, и архитектура собираемой телеметрии — это еще и архитектурное решение о приватности.

Практический чек-лист того, чем должен владеть современный специалист по анализу трафика:

- извлекать из дампа базовые метрики — RTT, пропускную способность, потери, джиттер — и работать с распределениями и перцентилями, а не средними,
- различать вклад сети, сервера и клиента в задержку транзакции и уверенно локализовать bottleneck,
- понимать динамику TCP — медленный старт, реакцию на потери, работу окна, отличия CUBIC и BBR — и узнавать их сигнатуры в трафике,
- анализировать прикладной уровень — фазы HTTP-транзакции, DNS, стоимость TLS-рукопожатия, особенности HTTP/2 и QUIC,
- знать классические паттерны проблем: потери, bufferbloat, черные дыры PMTU, Nagle с delayed ACK, переупорядочивание, NAT-таймауты, инъекции RST,

- распознавать атаки в трафике — флуды, сканирование, медленные атаки, отраженные DDoS, биконинг и туннели утечки,
- владеть иерархией наблюдаемости: метрики, потоки (NetFlow/IPFIX/sFlow), точечный полный захват, eBPF-события,
- учитывать особенности современных архитектур: оверлеи и туннели, облачные зеркала, контейнерные сети, сервис-меши, SDN.

Инструментарий продолжит меняться: eBPF вытесняет классический захват в повседневном мониторинге, машинное обучение — сигнатурные методы в обнаружении атак, ECH и QUIC — сокращают поверхность наблюдения, а новые транспорты добавляют слои инкапсуляции. Но фундамент останется прежним: понимание того, как протоколы ведут себя под нагрузкой, и методическая дисциплина измерений. Сложность — главный противник надежности систем, и по мере того как системы усложняются, цена умения быстро и доказательно ответить на вопрос «что на самом деле происходит в сети» только растет. Инженер, способный прочитать дамп как лог системы — увидеть в нем пилу алгоритма перегрузки, блокировку базы данных, раздувшийся буфер домашнего маршрутизатора или начало целевой атаки, — останется востребованным до тех пор, пока существуют сами сети. А они, судя по всему, существовать не перестанут.