Basis_Habr
4 апр в 17:52

Что будет, если не использовать TCP или UDP?

Средний 16 мин 49K

Блог компании Базис, Сетевые технологии*, Стандарты связи*, Читальный зал

Кейс Перевод

Автор оригинала: Hawzen

Коммутаторы, маршрутизаторы, брандмауэры — все это устройства, на которых держится интернет. Они перекидывают, фильтруют, дублируют и вырезают трафик такими способами, о которых большинство даже не догадывается. Без них вы бы не смогли прочитать этот текст.

Но сеть — это всего лишь один из уровней. Операционная система тоже играет по своим правилам: классификация, очереди, правила фаервола, NAT — все это влияет на то, что проходит, а что отбрасывается без следа. Каждый слой работает по-своему, и вместе они формируют ответ на вопрос: «А этот пакет вообще можно пропустить?»

Однажды мне стало интересно: а что будет, если отправить пакет с несуществующим транспортным протоколом? Не TCP, не UDP, не ICMP — вообще что-то выдуманное. Пропустит ли его ОС? Дойдет ли он хотя бы до сетевого интерфейса? Не зарежет ли его какой-нибудь промежуточный маршрутизатор? А вдруг он еще и быстрее обычного дойдет, потому что никто не знает, что с ним делать?

Ответа у меня не было. Так что я решил проверить.

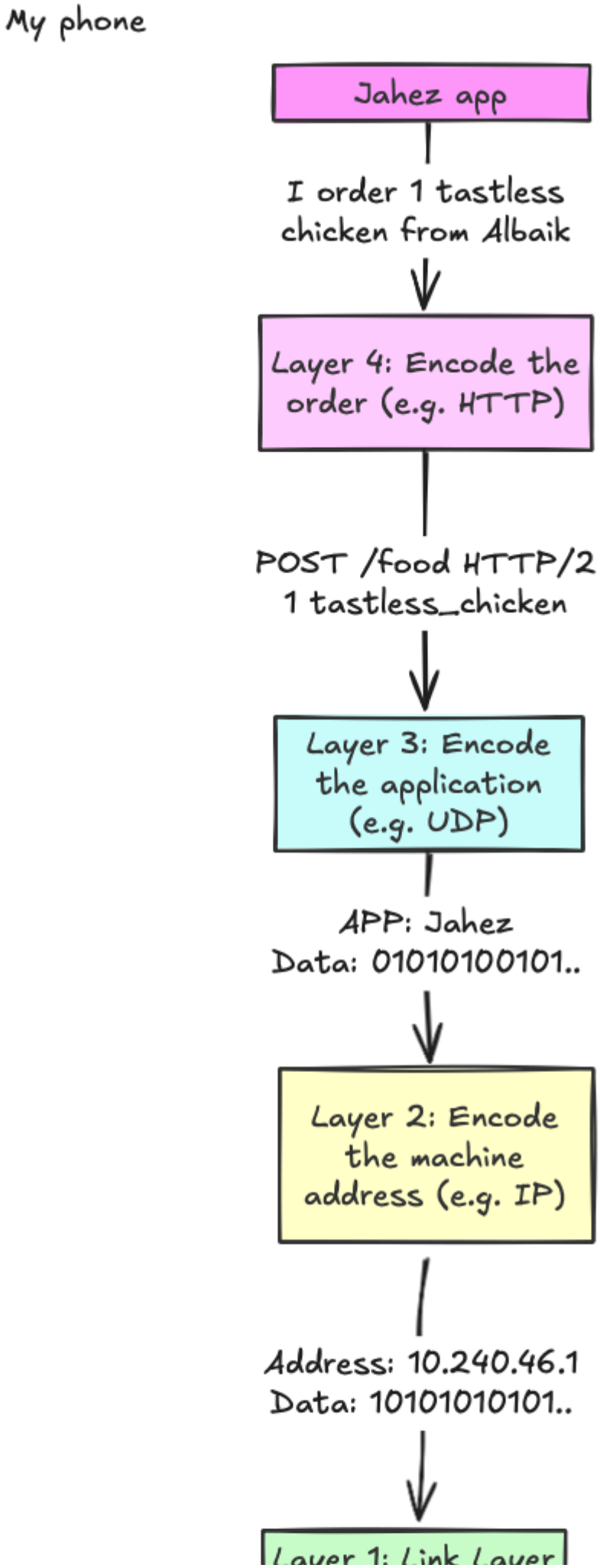
Сначала — самый простой эксперимент: отправить такой пакет самому себе. Посмотреть, как мой собственный компьютер справится с этим ядом, который я приготовил. Потом — попробовать переслать его через океан на удаленную Linux-машину, чтобы проверить, смогут ли они действительно добраться туда.

Немного предыстории

Эту часть можно пропустить, если вы знаете, как устроен интернет. Остальным — добро пожаловать.

Что вообще такое «транспортный протокол»? Почему все говорят про TCP и UDP, но никто не говорит о протоколе номер 42?

Интернет — не магия, хоть иногда и выглядит как волшебство. На самом деле это аккуратная стопка протоколов. Каждый — со своей задачей, своей областью ответственности. Они передают данные друг другу, слой за слоем, пока байты не окажутся там, где надо.


My phone
Jahez app
I order 1 tastless chicken from Albaik
Layer 4: Encode the order (e.g. HTTP)
POST /food HTTP/2
1 tastless_chicken
Layer 3: Encode the application (e.g. UDP)
APP: Jahez
Data: 01010100101..
Layer 2: Encode the machine address (e.g. IP)
Address: 10.240.46.1
Data: 10101010101..

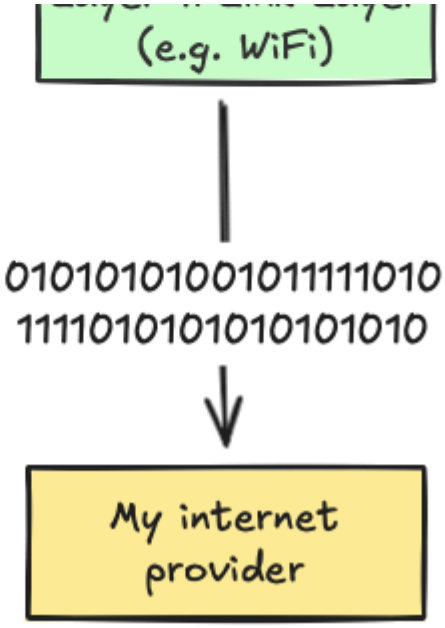


Диаграмма на 100% корректна и должна быть включена во все учебники по сетям

Сверху — приложения. Браузеры, мессенджеры, игры. Они говорят: «дай страницу», «отправь сообщение», «подключи меня к серверу». Все это превращается в запрос. И этот запрос начинает обрастать метаданными: адресами, портами, заголовками. Слой за слоем. Пока не останется просто поток битов, уходящий в никуда.

IP говорит: «Этот пакет — туда». Он отвечает за доставку. Канальный уровень — это физическая передача: Wi-Fi, Ethernet, оптика. Тут мы останавливаться не будем, потому что интересное начинается дальше.

И вот он, транспортный уровень. Первый по-настоящему сложный уровень протокола. Тут уже не просто «отправь куда-то». Тут — «передай надежно», «раздели между приложениями», «проверь, все ли дошло». Здесь живут TCP и UDP.

В заголовке каждого IP-пакета есть поле Protocol. Если там стоит 6 — это TCP. Если 17 — это UDP. А еще есть десятки других номеров. Некоторые — задокументированы. Некоторые — зарезервированы на будущее. А некоторые — просто ничьи.

Что будет, если взять и отправить пакет с одним из этих «ничейных» номеров?

Эксперимент №1: Отправка трафика... самому себе!

В этом эксперименте слишком много переменных: моя ОС, мой роутер, гипотетическая ОС получателя и целая пачка промежуточных звеньев в интернете. Пытаться разгрести такую кашу — не самая благодарная задача. Поэтому я решил начать с самого простого: отправить пакеты самому себе.

Такой подход исключает все лишнее. Результаты будут зависеть только от поведения моей операционной системы и сетевого стека.

Я написал свой транспортный протокол. Назвал его HDP. Что он делает — неважно. Главное — он не похож ни на один из известных. Это что-то совсем чуждое, ни одна ОС такого не ждет.

Потом я написал сервер — он же слушатель. Он просто ждет пакеты с определенным номером протокола. А еще — клиент, который эти пакеты отправляет. Все просто.

План действий:

- Запустить HDP-сервер

  → Он попросит ОС перенаправлять все IP-пакеты с номером протокола 255 на свой сокет.

- Запустить HDP-клиент

  → Он отправит пакет на 127.0.0.1 — локальный адрес, он же loopback.

  → ОС направит пакет на интерфейс обратной петли.

  → Интерфейс решит: «Ага, это для нас» — и вернет пакет обратно в систему.

- ОС доставит этот пакет на серверный сокет — без изменений… по крайней мере, я на это надеюсь.

Давайте попробуем. Открываю два терминала. В первом — сервер:

```
$ sudo cargo run --bin server
```

Во втором — клиент:

```
$ fortune | cowsay | sudo cargo run --bin client 127.0.0.1
```

3... 2... 1... Сервер получил сообщение!

```
$ sudo cargo run --bin server
~~~ IP Header ~~~
Version: 4
IHL: 5
DSCP: 0
ECN: 0
Total Length: 58625
Identification: 36455
Flags: 0
Fragment Offset: 0
TTL: 64
Protocol: 255
Header Checksum: 0
Source IP: [127, 0, 0, 1]
Destination IP: [127, 0, 0, 1]

~~~ HDP Header & Data ~~~
Source Port: 420
Destination Port: 420
Timestamp: 1739640243546134000
Data: ________________________________________
/ Marriage is not merely sharing the \
| fettucine, but sharing the burden of    |
| finding the fettucine restaurant in the |
| first place.                            |
|                                         |
\ -- Calvin Trillin /
 ----------------------------------------
        \   ^__^
         \  (oo)\_______
            (__)\       )\/\
                ||----w |
                ||     ||
```

Победа! Пакет с номером **255** не только не был отброшен, но и вернулся обратно. ОС спокойно его приняла и передала серверу. Я ожидал подвоха. А его не было. Но эксперимент на этом не закончился. Мне стало интересно: а если отправить пакет не с 255, а с чем-то более... необычным?

Например:

- **6** — это **TCP**;
- **2** — это **ICMP** (все, что связано с ping);

- **256** — вообще за пределами допустимого диапазона.

Что сделает ОС? Примет? Отбросит? Или просто зависнет? Давайте узнаем:

```
fortune | cowsay | sudo cargo run --bin client 127.0.0.1 # На этот раз перебираем номер
```

Результаты

▸ Огромная таблица

Что за сбои?

Большинство номеров протоколов сработали нормально — ОС принимала пакеты, возвращала их на loopback-интерфейс, и сервер их успешно получал. Но не все номера оказались такими покладистыми. Некоторые пакеты терялись в пути, причем по разным причинам:

- **Протоколы 1, 2 и 6** (например, ICMP и TCP) не доходили до сервера. Клиент их отправлял, но ОС на серверной стороне перехватывала их до того, как они попадали в сокет.
- **Протоколы 50 и 51** даже не уходили с клиента — ОС отказывалась их отправлять.
- **Протокол 256** вообще не проходил вызов socket() — значение вне диапазона, невалидно.

Почему так? Что именно заставляет ОС вести себя по-разному?

Системные вызовы: что на самом деле важно

Одна из самых полезных техник отладки, которую я освоил в ходе моего эксперимента — при работе с низкоуровневым кодом отслеживать системные вызовы, которые выполняет процесс.

Системный вызов для непосвященных — это просто функция, которая позволяет приложениям запрашивать привилегированные ресурсы у ОС, будь то открытие файла, выделение памяти или, в нашем случае, отправка пакета по сети.

В моем коде на Rust я использовал библиотеку socket2, которая оборачивает системные вызовы в удобный интерфейс. Вот пример кода, создающего сокет:

```
int sockfd = socket(
    AF_INET,    // Домен: Интернет-протоколы ARPA. Это сообщает ОС, что нас интересуют
    SOCK_RAW,   // Тип: Сырой сокет. Обычно ОС обрабатывает транспортный уровень, но эт
    255         // Протокол: Мы перебирали это поле.
);
```

Этот вызов говорит ОС: «Я хочу прямой доступ к IP-пакетам, вот протокол, с которым собираюсь работать». А дальше уже от ОС зависит — пустит она нас или нет.

Возвращаясь к неудачам

1, 2 и 6: Сервер их не видит

Эти пакеты клиент успешно отправлял, но сервер их не видел вовсе. Значит, ОС что-то сделала с ними по пути — возможно, перехватила, отбросила или обработала по-другому.

Я предполагал, что мой сервер сможет принимать вообще все IP-пакеты. Сокет я создавал вот так:

```
int sockfd = socket(
    AF_INET,    // Интернет-домен
    SOCK_RAW,   // Сырой сокет: должен дать нам полный контроль
    0           // Пусть ОС решает, какой протокол использовать
);
```

Я думал, что 0 — это универсальный вариант. Мол, *«передай все, что можешь»*. Но оказалось, это не так.

Для контекста: эксперименты я запускал на Mac, который работает на Darwin. Darwin похож на BSD, но с тонной макияжа. Поэтому он унаследовал не только системные вызовы, но и все причуды BSD-сокетов.

Немного покопавшись в документации, я не нашел ничего полезного про protocol = 0. Но потом наткнулся в BSD-документации на раздражающе расплывчатую фразу:

> Значение 0 для protocol позволит системе выбрать подходящий протокол для запрошенного типа сокета.

То есть, вместо того чтобы честно передавать все подряд, система молча и без объяснений фильтрует пакеты. Например, ICMP(1), IGMP(2) или TCP(6) просто не доходят до моего сокета. Видимо, Darwin решил, что он лучше знает, что именно я должен получать.

Вот и ответ: не все сокеты одинаково полезны. А с Darwin — еще и не всегда предсказуемы.

50 и 51: Клиент даже не может их отправить

На этом этапе стало понятно: есть номера протоколов, к которым ОС относится как к особо охраняемым. Протоколы **50** и **51** — это не просто какие-то случайные значения. Это IPSec: ESP и AH, применяемые для шифрования VPN-трафика.

Darwin категорически отказался их отправлять. Почему? Точной причины нет. Но я предполагаю, что это встроенная мера безопасности: мол, если ты не VPN, не лезь.

256: Вызов socket() немедленно завершается неудачей

Этот случай прост:

- поле protocol в IP-заголовке — 8-битное;
- максимальное значение — 255;
- 256 просто не помещается.

ОС моментально отвергла этот аргумент, даже не попытавшись его обработать.

Честно говоря, ничего удивительного. Но вот что действительно удивило — это поведение Linux.

После всех этих несостыковок я решил: а давай-ка посмотрим, что скажет Linux. Поднял виртуалку, повторил все шаги — и сразу увидел другое поведение.

Linux, в отличие от Darwin, **не позволяет** привязать сырой сокет к протоколу 0. Но при этом он **разрешил** использовать некоторые нестандартные значения, включая те, которые на macOS даже не создавали сокет. В том числе — 256.

Результаты я сохранил в results_no_server_linux_client_loopback. И остался доволен тем, что хотя бы часть моих ожиданий оправдалась.

Извлеченные уроки

Написать свой транспортный протокол — технически возможно. Но ОС от этого будет не в восторге. Сетевой стек забит предположениями, которые не всегда очевидны, а «сырой» сокет оказывается не таким уж и сырым.

Все это, как мне кажется, и объясняет, почему подавляющее большинство новых протоколов живут на уровне приложений. Вместо того чтобы бодаться с ОС и файрволлами, инженеры просто строят поверх чего-то, что уже работает. Например, QUIC — это фактически новый транспортный протокол, но он катается на UDP и избегает всей этой возни.

Если вдруг решите поиграть с сырыми сокетами — умоляю, проверяйте код на разных ОС. То, что позволяет сделать Darwin, может вызывать ступор у Linux. А то, что позволяет сделать Linux, в Windows работать вообще не будет. Поведение не стандартизовано, даже если все клянутся в POSIX-соответствии.

Следующий шаг: что происходит за пределами loopback?

До сих пор эти пакеты никогда не покидали мой компьютер. Теперь я хочу отправить HDP через публичный интернет:

- Будут ли маршрутизаторы пересылать его или отбрасывать?
- Пропустят ли его брандмауэры или отметят как атаку?
- Будет ли у него другая задержка по сравнению с TCP?
- Не уроню ли я случайно DigitalOcean?

Пора выяснить.

Эксперимент №2:

Изначально казалось, что этот эксперимент будет простым (спойлер: НЕТ). Я хотел взять самый дешевый VPS на DigitalOcean, запустить там сервер и начать швыряться в него чем попало: TCP, UDP, мой кастомный HDP и все остальное. Считать потери, смотреть задержки, делать выводы. В теории — просто.

На практике… все пошло не так. Не потому, что что-то не настроилось. А потому что результаты были странные. Они не вписывались в мои ожидания, и я не был морально готов их распутывать.

Настройка сервера

Я арендовал самый дешевый VPS на Digital Ocean, который смог найти, а затем настроил свой сервер и все необходимые инструменты. Отлично! Осталось понять, где он физически находится.

```
root@debian-s-1vcpu-512mb-10gb-fra1-01:~# curl myip.wtf
161.35.222.56
root@debian-s-1vcpu-512mb-10gb-fra1-01:~# curl ipinfo.io/161.35.222.56
{
  "ip": "161.35.222.56",
  "city": "Frankfurt am Main",
  "region": "Hesse",
  "country": "DE",
  "loc": "50.1155,8.6842",
  "org": "AS14061 DigitalOcean, LLC",
  "postal": "60306",
  "timezone": "Europe/Berlin",
  "readme": "https://ipinfo.io/missingauth"
}
```

Франкфурт. Отлично. А клиент у меня работает из Саудовской Аравии. Эксперимент — межконтинентальный. Прежде чем кидаться пакетами, я решил проверить, как пингуется сервер:

```
❯ ping 161.35.222.56
PING 161.35.222.56 (161.35.222.56): 56 data bytes
64 bytes from 161.35.222.56: icmp_seq=0 ttl=47 time=125.364 ms
64 bytes from 161.35.222.56: icmp_seq=1 ttl=47 time=128.061 ms
64 bytes from 161.35.222.56: icmp_seq=2 ttl=47 time=177.931 ms
64 bytes from 161.35.222.56: icmp_seq=3 ttl=47 time=225.798 ms
64 bytes from 161.35.222.56: icmp_seq=4 ttl=47 time=130.101 ms
64 bytes from 161.35.222.56: icmp_seq=5 ttl=47 time=194.563 ms
64 bytes from 161.35.222.56: icmp_seq=6 ttl=47 time=159.518 ms
64 bytes from 161.35.222.56: icmp_seq=7 ttl=47 time=134.343 ms
64 bytes from 161.35.222.56: icmp_seq=8 ttl=47 time=501.139 ms
64 bytes from 161.35.222.56: icmp_seq=9 ttl=47 time=153.672 ms
64 bytes from 161.35.222.56: icmp_seq=10 ttl=47 time=137.927 ms
64 bytes from 161.35.222.56: icmp_seq=11 ttl=47 time=355.672 ms
64 bytes from 161.35.222.56: icmp_seq=12 ttl=47 time=138.777 ms
64 bytes from 161.35.222.56: icmp_seq=13 ttl=47 time=166.116 ms
64 bytes from 161.35.222.56: icmp_seq=14 ttl=47 time=288.758 ms
64 bytes from 161.35.222.56: icmp_seq=15 ttl=47 time=151.458 ms
64 bytes from 161.35.222.56: icmp_seq=16 ttl=47 time=164.025 ms
```

```
64 bytes from 161.35.222.56: icmp_seq=17 ttl=47 time=170.132 ms
64 bytes from 161.35.222.56: icmp_seq=18 ttl=47 time=279.034 ms
^C
--- 161.35.222.56 ping statistics ---
19 packets transmitted, 19 packets received, 0.0% packet loss
```

Похоже, он довольно далеко, но мне кажется все в порядке. Давайте отправим несколько пакетов, используя наш новый протокол!

Сначала запускаю сервер на машине DigitalOcean:

```
root@debian-s-1vcpu-512mb-10gb-fra1-01:~/hdp/hdp# sudo cargo run --bin server
Listening on protocol 255
```

И со своего Mac отправляю пакет:

```
❯ fortune | cowsay | sudo cargo run --bin client 161.35.222.56
| Protocol Number | Succeeded (Client) | Time (µs) (Client) | Byte sum (Client) | Failu
| 255 | 🐮 | timestamp | 563 | - |
```

Пакет отправлен. Давайте снова проверим сервер:

```
root@debian-s-1vcpu-512mb-10gb-fra1-01:~/hdp/hdp# sudo cargo run --bin server
Listening on protocol 255
| Protocol Number | Time (µs) (Server) | Source IP (Server) | Byte Sum (Server) |
| --- | --- | --- |
| 255 | timestamp | my_ip | 563 |
```

Отлично. Похоже, все прошло хорошо, по крайней мере, так я думал. На самом деле, с этого момента все пошло под откос. Я сделал короткий перерыв, а затем вернулся и попробовал отправить пакет снова:

```
| Protocol Number | Time (µs) (Server) | Source IP (Server) | Byte Sum (Server) |
| --- | --- | --- |
```

```
| 255 | timestamp | my_ip | 563 |
```

Зависло? Я не вижу второй пакет. Пусто. Второй пакет не приходит. Жму Ctrl+C, пробую еще. Ноль. Лезу в tcpdump:

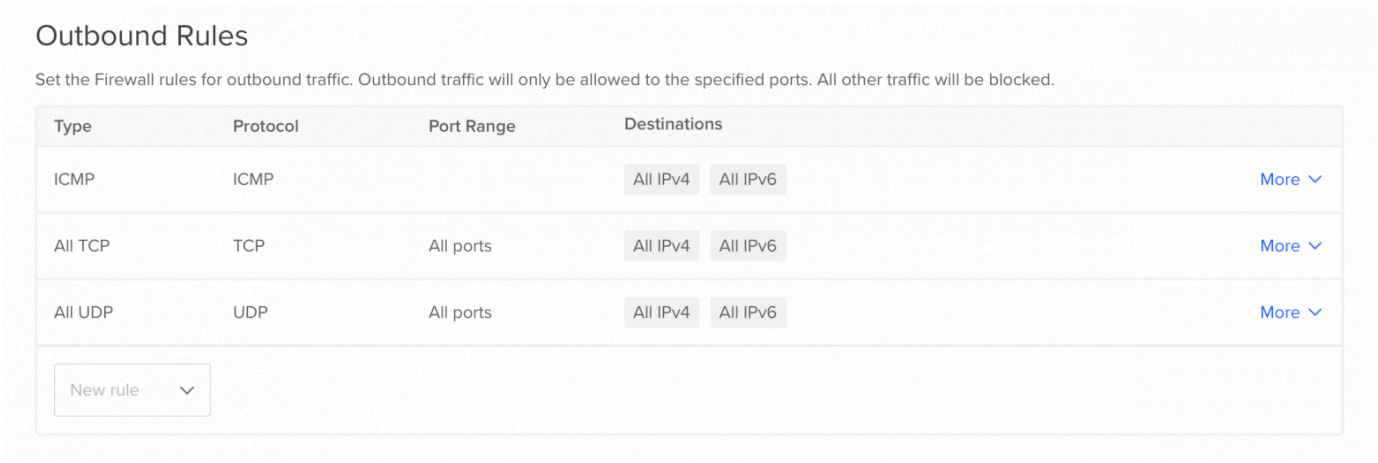
```
❯ sudo tcpdump -i any 'ip[9] == 255'
tcpdump: data link type PKTAP
tcpdump: verbose output suppressed, use -v[v]... for full protocol decode
listening on any, link-type PKTAP (Apple DLT_PKTAP), snapshot length 524288 bytes
IP mac > 161.35.222.56:  reserved 427
IP mac > 161.35.222.56:  reserved 427
IP mac > 161.35.222.56:  reserved 427
IP mac > 161.35.222.56:  reserved 427
IP mac > 161.35.222.56:  reserved 427
IP mac > 161.35.222.56:  reserved 427
IP mac > 161.35.222.56:  reserved 427
IP mac > 161.35.222.56:  reserved 427
```

Хорошо. Значит, мой клиент их точно отправляет. Что там на сервере?

```
root@debian-s-1vcpu-512mb-10gb-fra1-01:~/hdp# tcpdump -i any 'ip[9] > 17'
tcpdump: data link type LINUX_SLL2
tcpdump: verbose output suppressed, use -v[v]... for full protocol decode
listening on any, link-type LINUX_SLL2 (Linux cooked v2), snapshot length 262144 bytes
```

Тут у меня начали сдавать нервы. Я пересмотрел старые логи — нет, я не сошел с ума. Пакет действительно доходил. Метка времени, сумма байт — все совпадает. Но остальные — исчезают в пустоте. Может, Торвальдс лично меня газлайтит?

Стоп. Как NAT-устройство моего интернет-провайдера переслало пакет? NAT полагается на порты — но мой протокол для них просто черная магия. Так что вполне возможно, он просто не знает, что с этим делать — и блокирует. После небольшого ресерча я нашел подтверждение: DigitalOcean не поддерживает нестандартные IP-протоколы.



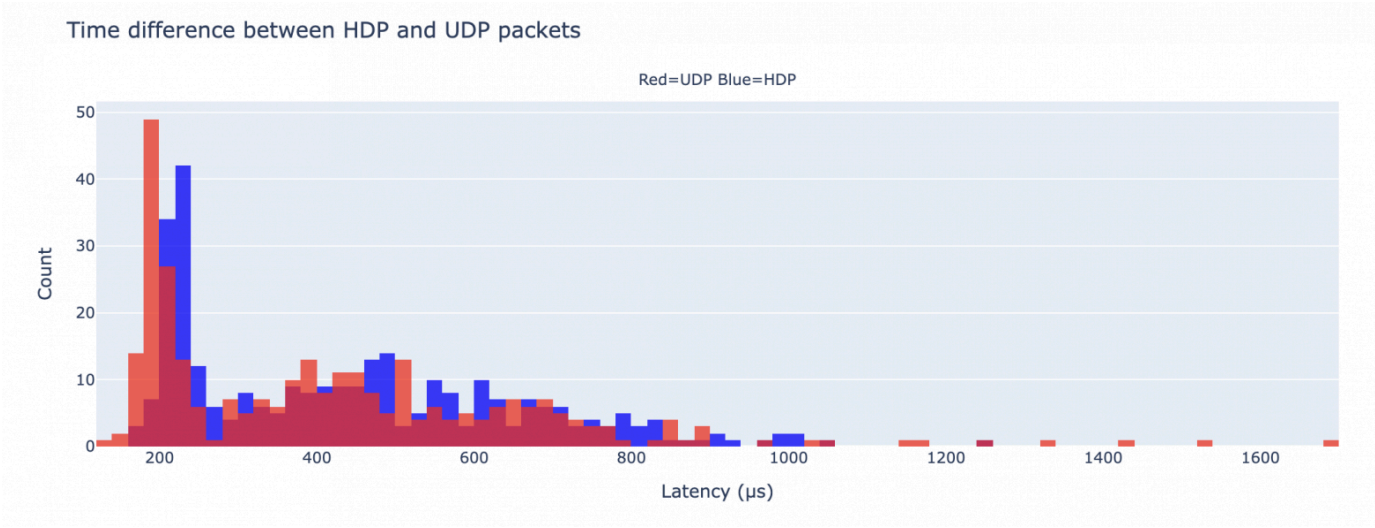
Что, впрочем, только еще больше запутало ситуацию. Если они не поддерживают, то как первый пакет прошел? Ответа нет. Все, что у меня осталось — это лог, доказывающий, что он действительно дошел.

Самая. Последняя. Попытка.

Если где-то и существует облачный провайдер, поддерживающий нестандартные IP-протоколы, то это AWS. Я поднял две виртуалки, настроил сервер и клиент. Оно работает!

```
admin@ip-172-31-13-218:~/hdp$ sudo cargo run --bin server 255
Server is listening on SockAddr { ss_family: 2, len: 16 }, protocol: 255
| Protocol Number | Time (µs) (Server) | Source IP (Server) | Byte Sum (Server) |
| --- | --- | --- |
| 255 | timestamp | 54.153.13.186 | 33 |
| 255 | timestamp | 54.153.13.186 | 34 |
| 255 | timestamp | 54.153.13.186 | 35 |
| 255 | timestamp | 54.153.13.186 | 36 |
```

Правда, сервер находился всего в двух хопах от клиента, так что страшных приключений через весь интернет не было.



Задержка в микросекундах, оба сервера находятся в одном дата-центре

Я замерил разницу между HDP и UDP. Разница есть — но настолько крошечная, что ей можно пренебречь: примерно 20 мкс в среднем. Никакой реальной выгоды.

А как насчет интернета?

Я попробовал отправлять пакеты со своего Mac на сервер в AWS — и получил тот же эффект: первый пакет доходит, и все. Оставил результаты в файле tcpdump_tokyo_server_mac_client.md. Послал по одному пакету для каждого протокола. Работают только TCP, UDP и ICMP. Все остальное — тишина после первого запроса.

Как и ожидалось, между машиной на DigitalOcean и AWS ничего не передается. Гарантированно ничего сказать нельзя, но выводы напрашиваются.

Что я понял

Теоретически — да, можно использовать свой IP-протокол. Но если вы не мазохист — **не надо**.

- Код будет привязан к ОС. Поддерживать это на кросс-платформе — отдельный ад.
- NAT, фаерволы, маршрутизаторы — все будет стараться убить ваш пакет. И в большинстве случаев — убьет.
- Локально, может, и заработает. В интернете — забудьте.
- И главное: **никакого выигрыша по задержке** за счет использования нестандартного протокола я не увидел.

TL;DR: Используйте UDP или TCP.

А если через IPv6?

Несколько читателей предложили попробовать мой протокол по IPv6 — там нет NAT'а, как в IPv4. Любопытно? Еще бы.

Я добавил поддержку IPv6, подключился к тому же серверу на AWS:

```
admin@ip-172-31-2-72:~/hdp$ RUST_BACKTRACE=full sudo cargo run --bin server 6 255 # The
| Protocol Number | Time (µs) (Server) | Source IP (Server) | Byte Sum (Server) |
| --- | --- | --- |
```

Теперь запускаю клиента с Mac, через океаны и материки. И — вуаля:

```
❯ fortune | cowsay | sudo cargo run --bin client 200 '2600:1f1c:1cf:b1ce:f653:afc7:4650
Running `target/debug/client 2000 '2600:1f1c:1cf:b1ce:f653:afc7:4650:8aa0' 17 udp`
| Protocol Number | Succeeded (Client) | Time (µs) (Client) | Byte sum (Client) | Failu
| 255 | 🧑 | 1740779795209398 | 49 | - |
| 255 | 🧑 | 1740779795413344 | 50 | - |
| 255 | 🧑 | 1740779795620781 | 51 | - |
```

Он появляется на сервере | 255 | 1740779715088323 | my_ip | 49 |

Сработало! Это были настоящее американские горки, и поездка удалась.

Полезные ссылки

- Спецификация UDP-протокола — настолько минимальна, что это даже смешно.
- Список IP-протоколов, назначенных для тестов.
- Протоколы, официально поддерживаемые IP.
- Статья про отличия raw sockets в Linux и FreeBSD.
- Интересный ответ на тему: как реализовать NAT для чего-то кроме TCP/UDP.

Теги: эксперимент, передача данных, udp, tcp, протоколы

Хабы: Блог компании Базис, Сетевые технологии, Стандарты связи, Читальный зал

+174 295 118

Базис

Современные средства виртуализации

Сайт Telegram ВКонтакте

27 Карма 26 Рейтинг

@Basis_Habr

Пользователь

Подписаться

Комментарии 118

Публикации

ЛУЧШИЕ ЗА СУТКИ ПОХОЖИЕ

Maximov_psy
12 часов назад

Что я узнал, проконсультировав 100 айтишников

12 мин 8.5K

+45 49 53

ru_vds
14 часов назад

Как защищают фильмы и доставляют их в кинотеатры

Средний 12 мин 3.2K

Обзор Перевод

+36 30 6

MrSotnik
17 часов назад

Новый язык от 1С: Зачем? Кому? Стоит ли лезть?

5 мин 22K

+36 34 76

RationalAnswer
22 часа назад

Тотальный блэкаут на юге Европы, а также чудеса нейро-лизоблюдия от ChatGPT

11 мин 13K

Дайджест

+34 17 70

alizar
18 часов назад

Ян Лекун, создатель LeNet, формата DjVu и адвокат опенсорса

Средний 7 мин 1.8K

Обзор

+28 13 6

El_Gato_Grande
18 часов назад

Как ИИ-контент проклял интернет и почему это закономерно

8 мин 4.4K

+25 10 9

BaDInMe
18 часов назад

ML-обработка видео в web-браузере для видеоконференций SaluteJazz

Средний 14 мин 238

Кейс

+21 5 0