

Отчет по лабораторной работе 3

Генералов Даниил, НПИбд-01-21, 1032202280

Содержание

1 Цель работы	5
2 Задание	6
3 Выполнение лабораторной работы	7
4 Выводы	28

Список иллюстраций

3.1 ip addr	7
3.2 ip link+route+neighbor	8
3.3 MSI	10
3.4 Rivet	11
3.5 wireshark	12
3.6 icmp echo	13
3.7 icmp echo reply	13
3.8 arp	14
3.9 rudn.ru	15
3.10 dns query	16
3.11 dns response	16
3.12 ping	17
3.13 http	18
3.14 tls client hello	19
3.15 tls server hello	20
3.16 quic	21
3.17 quic	22
3.18 tcp	23
3.19 tcp syn	23
3.20 tcp syn ack	24
3.21 tcp ack	25
3.22 tcp psh ack client	26
3.23 tcp psh ack server	26
3.24 tcp fin ack	27

Список таблиц

1 Цель работы

В рамках этой лабораторной работы требуется использовать инструмент Wireshark для анализа трафика.

2 Задание

3.3.1.1.1. Изучение возможностей команды ipconfig для ОС типа Windows (ifconfig для систем типа Linux). 3.3.1.1.2. Определение MAC-адреса устройства и его типа. 3.3.2.1.1. Установить на домашнем устройстве Wireshark. 3.3.2.1.2. С помощью Wireshark захватить и проанализировать пакеты ARP и ICMP в части кадров канального уровня. 3.3.3.1. С помощью Wireshark захватить и проанализировать пакеты HTTP, DNS в части заголовков и информации протоколов TCP, UDP, QUIC. 3.3.4.1. С помощью Wireshark проанализировать handshake протокола TCP.

3 Выполнение лабораторной работы

Сначала я использовал утилиту ip (бывшая ifconfig) для того, чтобы получить информацию о сетевых соединениях своего компьютера. Большинство полезной информации об этом можно получить из выдачи команды ip address.

```
[danya@archlinux ~]$ ip -c address
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
        valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host noprefixroute
        valid_lft forever preferred_lft forever
2: enp3s0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq state UP group default qlen 1000
    link/ether 4c:cc:6a:e2:4a:f6 brd ff:ff:ff:ff:ff:ff
    inet 10.0.0.128/24 brd 10.0.0.255 scope global dynamic noprefixroute enp3s0
        valid_lft 24531sec preferred_lft 24531sec
    inet6 fe80::4317:cbd3:c507:238d/64 scope link noprefixroute
        valid_lft forever preferred_lft forever
3: wlp2s0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group default qlen 1000
    link/ether 9c:b6:d0:67:46:01 brd ff:ff:ff:ff:ff:ff
    inet 10.0.192.19/24 brd 10.0.192.255 scope global dynamic noprefixroute wlp2s0
        valid_lft 24540sec preferred_lft 24540sec
    inet6 fe80::5aed:c2bc:55d:f664/64 scope link noprefixroute
        valid_lft forever preferred_lft forever
4: outline-tun0: <NO-CARRIER,POINTOPOINT,MULTICAST,NOARP,UP> mtu 1500 qdisc fq_codel state DOWN group default qlen 500
    link/none
    inet 10.0.85.1/32 scope global outline-tun0
        valid_lft forever preferred_lft forever
[danya@archlinux ~]$
```

Рис. 3.1: ip addr

Из этого видно, что у меня есть 4 сетевых интерфейса: - loopback-интерфейс `lo`, который присутствует на каждом компьютере и перенаправляет пакеты обратно на этот компьютер; - Ethernet-адаптер `enp3s0`, который подключен с IP-адресом `10.0.0.128` и работает; - Wi-Fi-адаптер `wlp2s0`, который подключен с IP-адресом `10.0.192.19` и работает; - TUN-адаптер `outline-tun0`, который управляемся приложением Outline и сейчас не работает, но если бы работал, то

мой компьютер имел бы адрес 10.0.85.1.

Из других подкоманд команды ip можно узнать другую интересную информацию. Например:

- соединения enp3s0 и wlp2s0 имеют link/ether, что значит, что по этим интерфейсам проходят совместимые Ethernet-пакеты, и приведен их MAC-адрес.
- по проводному соединению enp3s0 можно связаться только с роутером 10.0.0.1, потому что провод подключен непосредственно туда, однако по беспроводному соединению wlp2s0 можно связаться не только с роутером 10.0.192.1, но и с моей IP-камерой 10.0.192.3; именно это соединение предоставляет изображение с камеры для видео выполнения этой лабораторной работы;
- мой компьютер будет отправлять пакеты для подсети 10.0.0.0/24 через устройство enp3s0 и подписывать их своим IP-адресом 10.0.0.128, для подсети 10.0.102.0/24 через устройство wlp2s0 и подписывать их своим IP-адресом 10.0.192.19, а для остальных сначала будет пробовать отправить их через 10.0.0.1, а затем через 10.0.192.1.

```
[danya@archlinux ~]$ ip -c link
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN mode DEFAULT group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
2: enp3s0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq state UP mode DEFAULT group default qlen 1000
    link/ether 4c:cc:6a:e2:4a:f6 brd ff:ff:ff:ff:ff:ff
3: wlp2s0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP mode DORMANT group default qlen 1000
    link/ether 9c:b6:d0:67:46:01 brd ff:ff:ff:ff:ff:ff
4: outline-tun0: <NO-CARRIER,POINTOPOINT,MULTICAST,NOARP,UP> mtu 1500 qdisc fq_codel state DOWN mode DEFAULT group default
    qlen 500
    link/none
[danya@archlinux ~]$ ip -c neighbor
10.0.0.1 dev enp3s0 lladdr 50:ff:20:8c:34:53 REACHABLE
10.0.192.3 dev wlp2s0 lladdr 60:a4:b7:ea:21:e0 REACHABLE
10.0.192.1 dev wlp2s0 lladdr 52:ff:20:8c:34:51 STALE
fe80::52ff:20ff:fe8c:3453 dev enp3s0 lladdr 50:ff:20:8c:34:53 router STALE
[danya@archlinux ~]$ ip -c route
default via 10.0.0.1 dev enp3s0 proto dhcp src 10.0.0.128 metric 100
default via 10.0.192.1 dev wlp2s0 proto dhcp src 10.0.192.19 metric 600
10.0.0.0/24 dev enp3s0 proto kernel scope link src 10.0.0.128 metric 100
10.0.85.2 dev outline-tun0 proto kernel scope link src 10.0.85.1 linkdown
10.0.192.0/24 dev wlp2s0 proto kernel scope link src 10.0.192.19 metric 600
[danya@archlinux ~]$
```

Рис. 3.2: ip link+route+neighbor

Также можно проанализировать MAC-адреса двух физических устройств: `enp3s0=4c:cc:6a:e2:4a:f6` и `wlp2s0=9c:b6:d0:67:46:01`. Из последних двух битов первого байта можно определить тип этих адресов – поскольку оба этих бита равны нулю, можно понять, что это уникастовые (индивидуальные) и глобально-настроенные MAC-адреса (потому что я их не менял на своем устройстве).

Первые три байта, за исключением этих двух битов, обозначают производителя сетевого устройства. Если поискать в интернете первые три байта, то можно определить, что `4c:cc:6a` принадлежит `Micro-Star INTL CO., LTD.`, а `9c:b6:d0` – `Rivet Networks`.

Micro-Star INTL CO., LTD.

Vendor

Details

 OUI: 4C:CC:6A

 Vendor name: [Micro-Star INTL CO., LTD.](#) 

 Address:

No.69
Lide St.

New Taipei City Taiwan 235
TW.

 Assignment Type MA-L

Mac Address Block Large (previously named OUI). Number of address 2^{24} (~16 Million)

 Initial registration: 03 December 2015

Рис. 3.3: MSI

Rivet Networks

Vendor

Details

💻 OUI: 9C:B6:D0 ⚡

🏭 Vendor name: [Rivet Networks](#) ⚡

📘 Address:

11940 Jollyville Rd
Austin tx 78759
US.

📅 Assignment Type MA-L

Mac Address Block Large (previously named OUI). Number of address 2^{24} (~16 Million)

📅 Initial registration: 17 November 2015

Рис. 3.4: Rivet

Первое название мне известно, потому что это – редко-используемое полное название компании MSI, от которой мой ноутбук. Второе название мне незнакомо, но на первой странице поиска можно найти новости о том, что эта компания была куплена Intel и раньше была производителем Wi-Fi модулей под брендом Killer, что соответствует надписи на рекламной этикетке на ноутбуке.

После этого я начал использование Wireshark. Я открыл программу и выбрал захват на интерфейсе `enp3s0`, потому что другой интерфейс в данный момент слишком занят трафиком с IP-камеры. Запустив в отдельном окне `ping` на адрес своего роутера, я увидел ICMP-пакеты от моего компьютера к роутеру, а также сразу после этого пакеты от роутера к компьютеру.

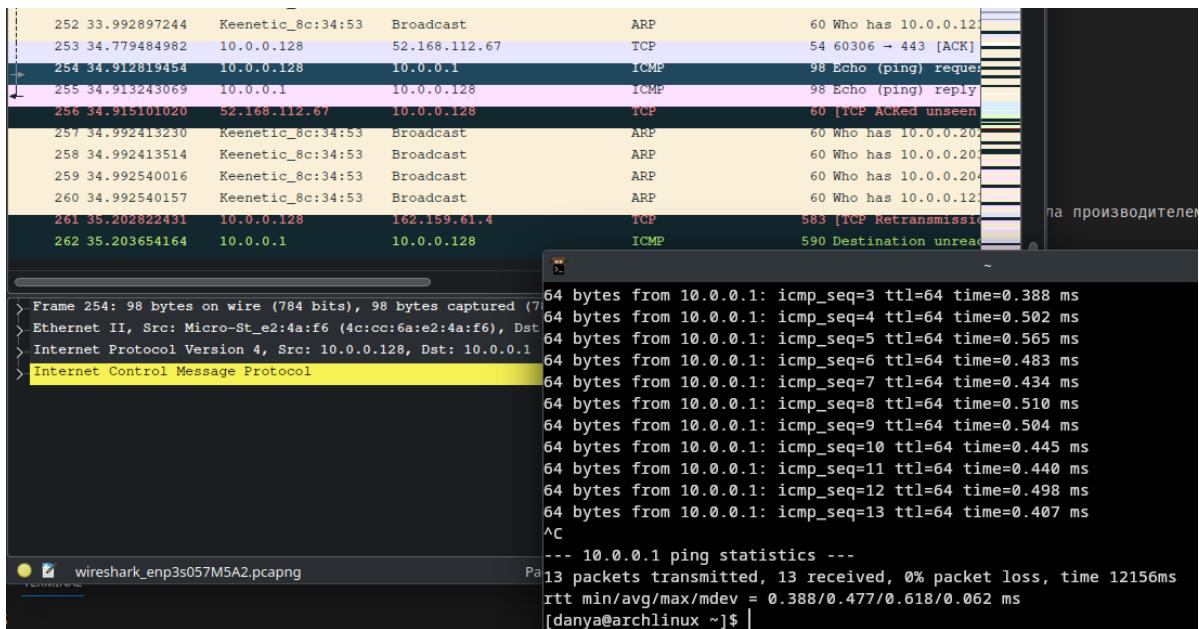


Рис. 3.5: wireshark

Мы рассмотрим пакеты 254 и 255 в записи. Пакет 254 – это ICMP Echo request, который находился в IP-пакете, который – внутри Ethernet-кадра. Ethernet-кадр отправлялся от моего MAC-адреса, 4c:cc:6a:e2:4a:f6, и направлялся к MAC-адресу моего роутера, 50:ff:20:8c:34:53. Кадр имеет тип 0x0800, что значит, что он содержит IPv4-пакет. Весь кадр занимал 98 байтов, все из которых видны в записи.

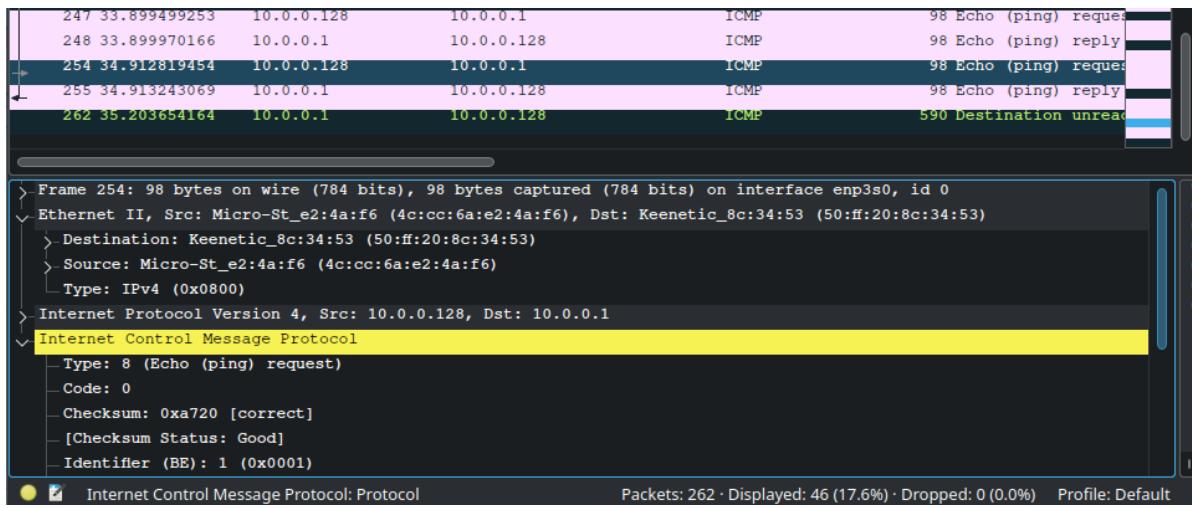


Рис. 3.6: icmp echo

Пакет 255 – ICMP Echo reply, который в итоге находился в Ethernet-кадре, который также имеет тип 0x0800 и содержит IPv4-пакет. Этот кадр исходит от роутера, 50:ff:20:8c:34:53, и направляется к моему ноутбуку, 4c:cc:6a:e2:4a:f6. Он также имеет длину в 98 байтов, потому что он отличается только в порядке двух адресов, а также в значениях нескольких байтов.

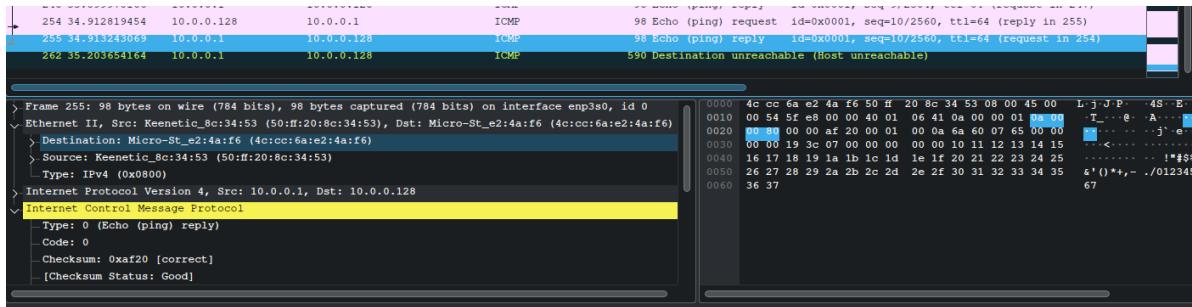


Рис. 3.7: icmp echo reply

Также в этой записи присутствуют много ARP-пакетов от моего роутера. Для того, чтобы работал веб-интерфейс, показывающий список устройств, роутер должен знать про все устройства, которые подключены к его сети; поэтому он систематически отправляет ARP-запросы по каждому из IP-адресов, которые относятся к нему. Каждый из этих запросов – это широковещательный

Ethernet-пакет (направленный на `ff:ff:ff:ff:ff:ff`), который имеет Ethertype `0x0806`, то есть ARP. Внутри этого кадра, роутер говорит, что речь идет про Ethernet и IPv4-адреса, которые имеют размеры 6 и 4 соответственно, и он делает запрос, в котором говорит, что он имеет IP-адрес `10.0.0.1` и MAC-адрес `50:ff:20:8c:34:53`, и он хочет узнать про IP-адрес `10.0.0.198`, MAC-адрес для которого ему пока что неизвестен. Никто не отвечает, потому что по этому физическому порту к роутеру подключен только мой компьютер, про который роутер уже знает, и никаких других IP-адресов у него нет.

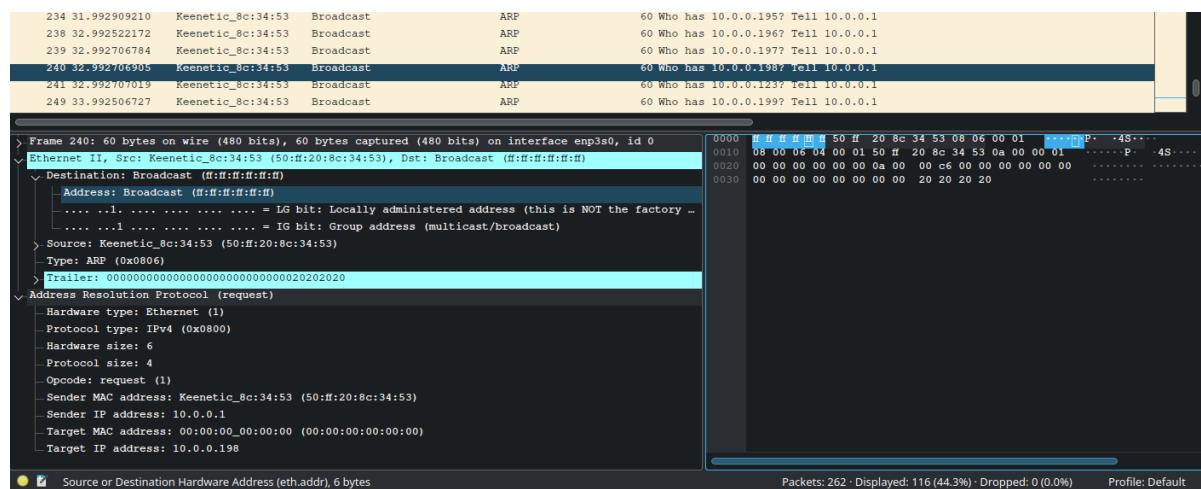


Рис. 3.8: arp

После этого я перезапускаю запись, и в другом окне запускаю `ping ruds.ru`. В окне Wireshark мы видим, как происходил этот весьма односторонний разговор.

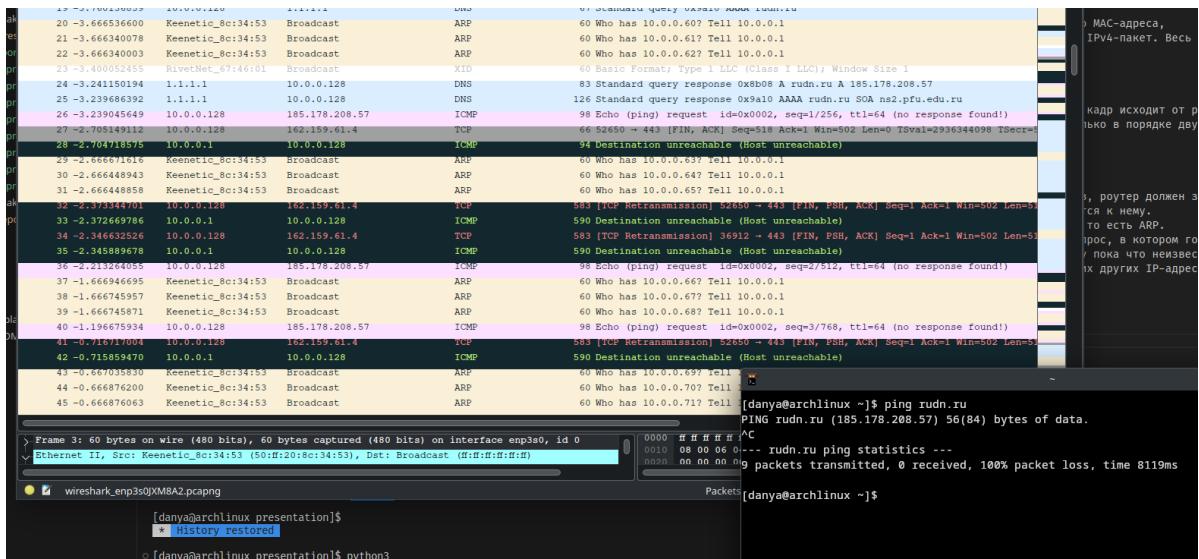


Рис. 3.9:rudn.ru

Сначала мой компьютер должен узнать, какой IP-адрес у домена `rudn.ru`. Для этого он задает DNS-запрос своему настроенному DNS-серверу, который в моем случае – `1.1.1.1`. В запросе он спрашивает про IN A-запись, связанную с `rudn.ru`. Это все происходит внутри UDP-пакета, который внутри IP-пакета который находится внутри Ethernet-кадра с `EtherType=0x0800` и длиной в 67 байтов. (В лабораторной работе написано проанализировать ARP-пакеты, но они не имеют отношения к этому разговору, и мне кажется, что на самом деле подразумевались запросы DNS.)

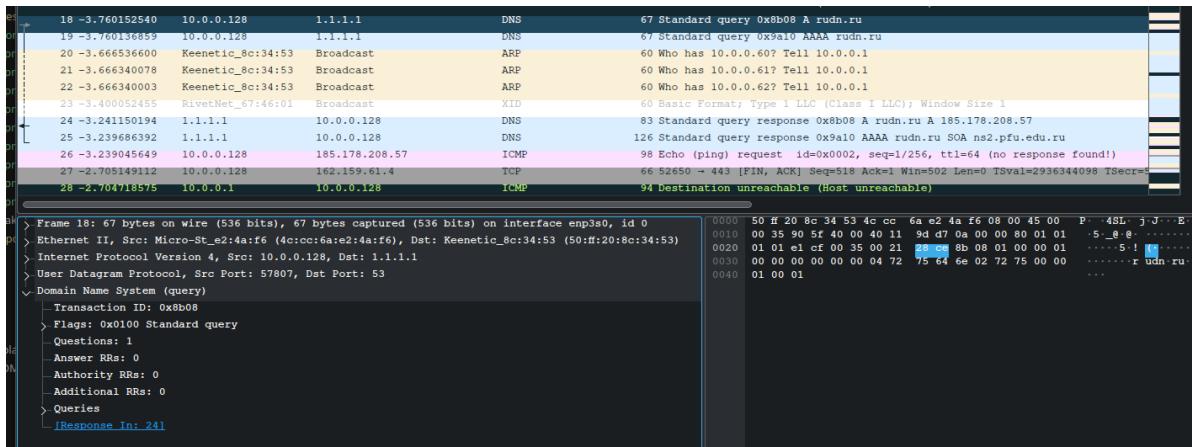


Рис. 3.10: dns query

Вскоре после этого мне приходит ответ – IN A для `rudn.ru` равен `185.178.208.57`. Этую информацию можно запомнить на 3 часа, или 10800 секунд, в течении которых она должна быть актуальной. Этот ответ также находится в UDP-пакете, в IP-пакете от `1.1.1.1`, который в Ethernet-пакете от моего роутера.

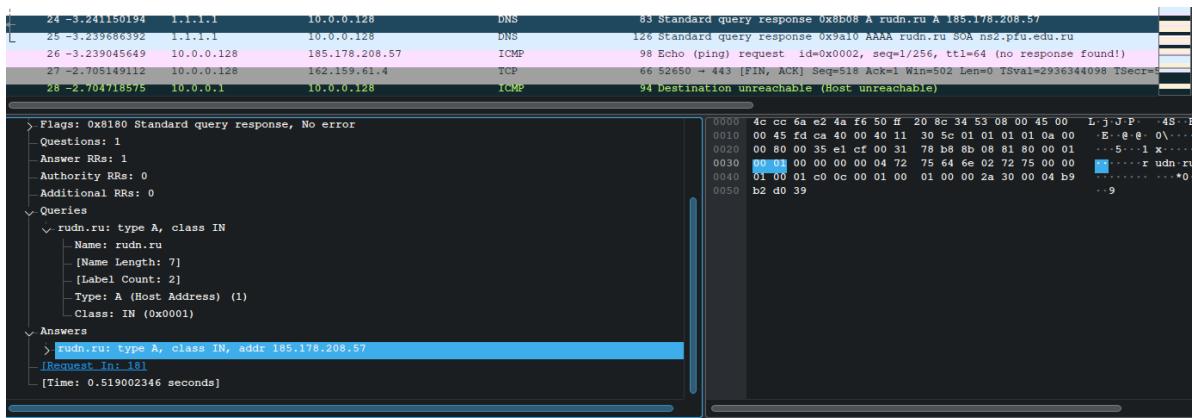


Рис. 3.11: dns response

Теперь, когда мы знаем IP-адрес нашего назначения, `rudn.ru`, мы можем начать отправлять ICMP Echo туда. Они уходят от нас, внутри IP-пакетов, а затем внутри Ethernet-кадров с `EtherType=0x0800` и адресованных роутеру на его индивидуальный, глобально-настроенный MAC-адрес. Но после этого мы не получаем ответов. Это может значить, что на пути между мной и `185.178.208.57`

есть проблема с сетью, которая мешает моим пакетам добраться до него, или его пакетам добраться до меня. Однако в этом случае я знаю, что сам сайт rudn.ru работает и доступен с моего компьютера, поэтому вторая причина – это администратор 185.178.208.57 настроил его так, чтобы он не отвечал на ICMP Echo. Это обычно делают ради возможного улучшения в безопасности, потому что становится слегка сложнее определить, что по этому IP-адресу есть работающий компьютер, но это также делает отладку сетевых соединений гораздо более сложной.

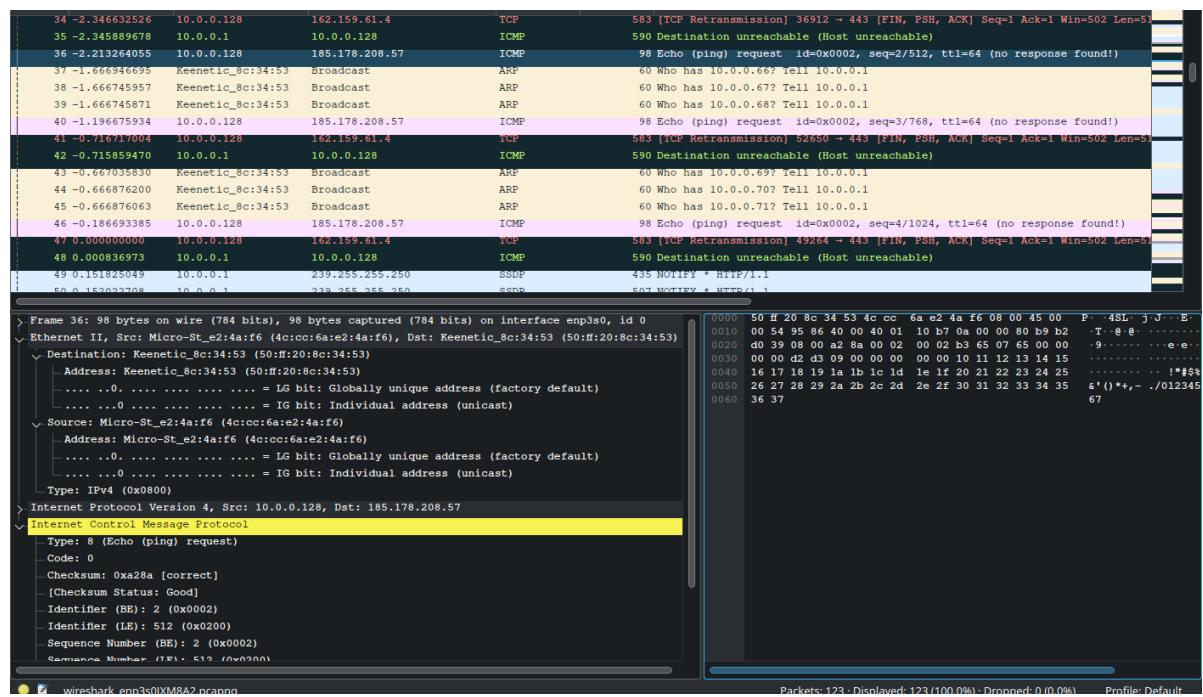


Рис. 3.12: ping

После этого я начал анализ протоколов прикладного уровня, на примере HTTP. Я запустил новую запись, а затем в браузере зашел на страницу <http://neverssl.com>, затем на <https://example.com>, а затем на <https://cloudflare-quic.com>. В первом случае я получил пакеты простого HTTP, во-втором – пакеты HTTPS, а именно HTTP внутри TLS, а в третьем – пакеты QUIC.

Сначала мы смотрим на http-сообщения. HTTP – это текстовый протокол, поэтому незашифрованные сообщения можно прочитать глазами. Сначала

браузер посыпает запрос, в данном случае GET / HTTP/1.1 на домен neverssl.com. После этого сервер отвечает 200 OK, и возвращает HTML-страницу, которая сжата gzip. Из-за этого сжатия, когда браузеру потребовалось целых 410 байтов, чтобы попросить у сервера показать страницу, серверу же хватило лишь 2339, чтобы согласиться, отправить мне метаданные, а также весь текст и разметку страницы. После этого там есть несколько других запросов-ответов, которые в целом выглядят так же: я спрашиваю у сервера вернуть мне ресурс, и указываю в каком виде я хочу его видеть, а сервер отправляет мне в ответ содержимое этого ресурса. Все это, конечно же, происходит внутри TCP-пакетов, которые внутри IP-пакетов, которые внутри Ethernet-кадров, которыми обменивается мой компьютер и роутер.

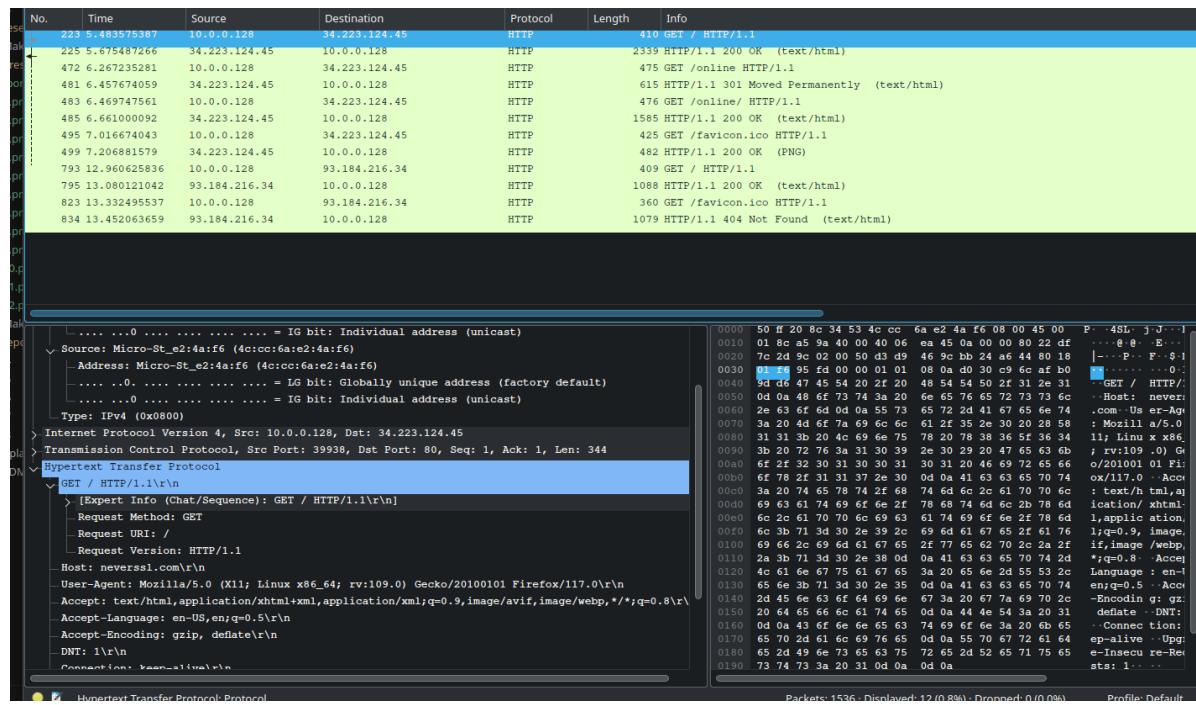


Рис. 3.13: http

Затем рассмотрим tls-сообщения, которыми мы обменивались с example.com, когда использовали HTTPS. TLS-соединения используются чаще всего, и мой браузер устанавливает многие из них в фоновом режиме, поэтому я также

фильтрую по IP-адресу.

Сначала мой компьютер делает `Client Hello`, в котором он говорит серверу, с кем именно мы хотим говорить (в данном случае, с `example.com`), какие протоколы мы хотим использовать внутри TLS-соединения (`h2` и `http/1.1`), и какие криптографические примитивы мы готовы использовать.

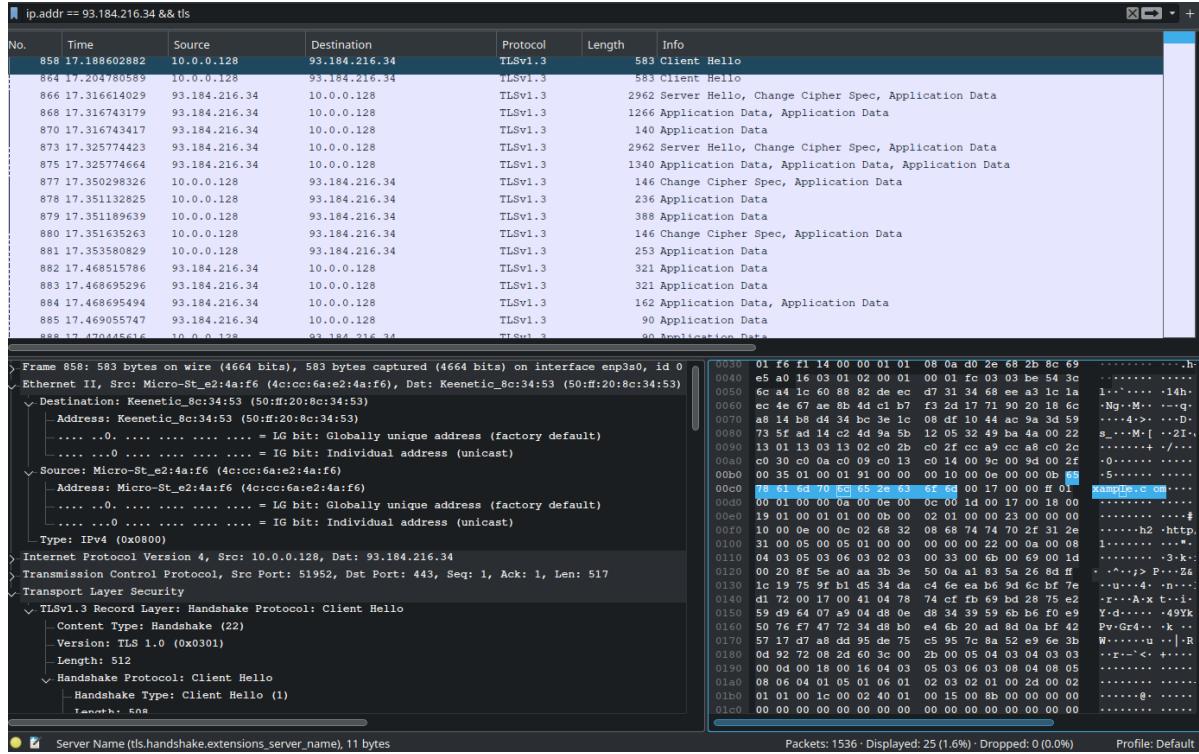


Рис. 3.14: tls client hello

Затем сервер отвечает своим `Server Hello`, где он говорит нам, что он поддерживает `TLSv1.3`, затем указывает, какие шифры он будет использовать, выполняет свою половину обмена ключами (через эллиптическую кривую `secp256r1`), а затем посыпает нам несколько зашифрованных данных. Браузер в этот момент совершил свою половину обмена ключами и знает, как расшифровать эти данные, и уже внутри них они ведут разговор о более высокоровневых деталях, вроде сертификатов сервера и клиента, а также содержимым веб-страницы.

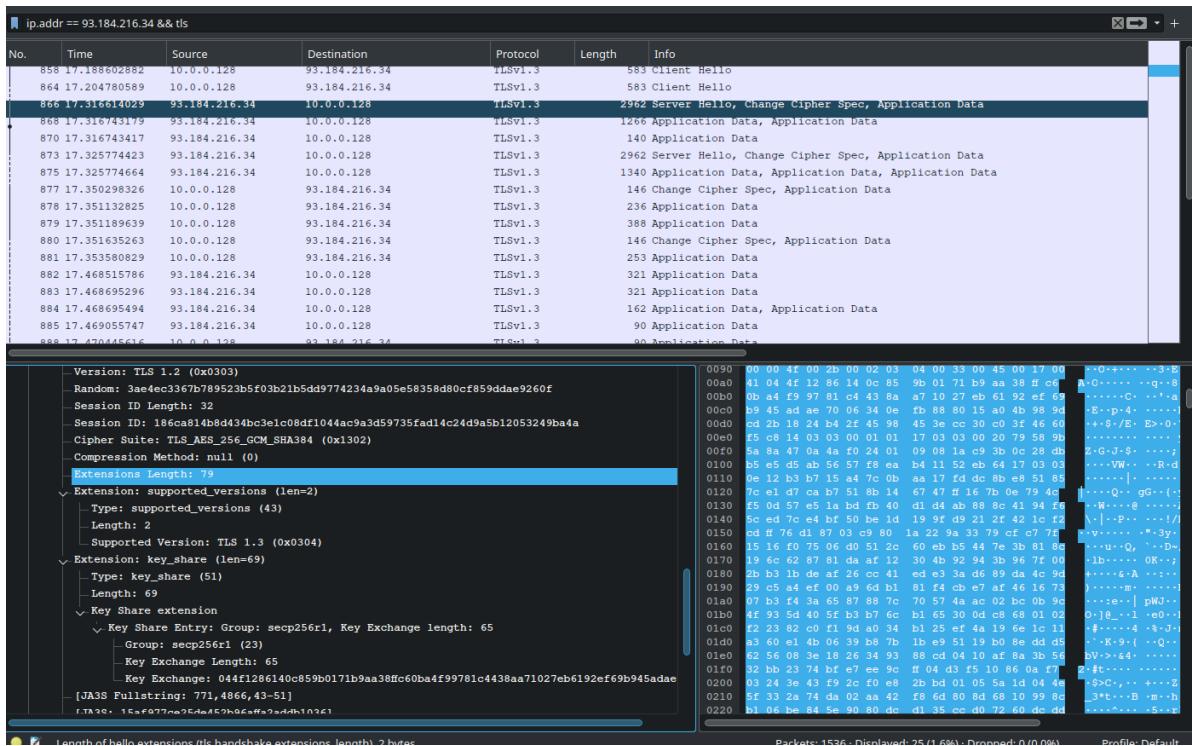


Рис. 3.15: tls server hello

Наконец, мы смотрим на quic-сообщения. Здесь уже сложнее понять, что происходит, в том числе потому что тут идут разговоры с сайтами на которые я не заходил (вроде `cloudflare.com`), что возможно является предварительной загрузкой от браузера.

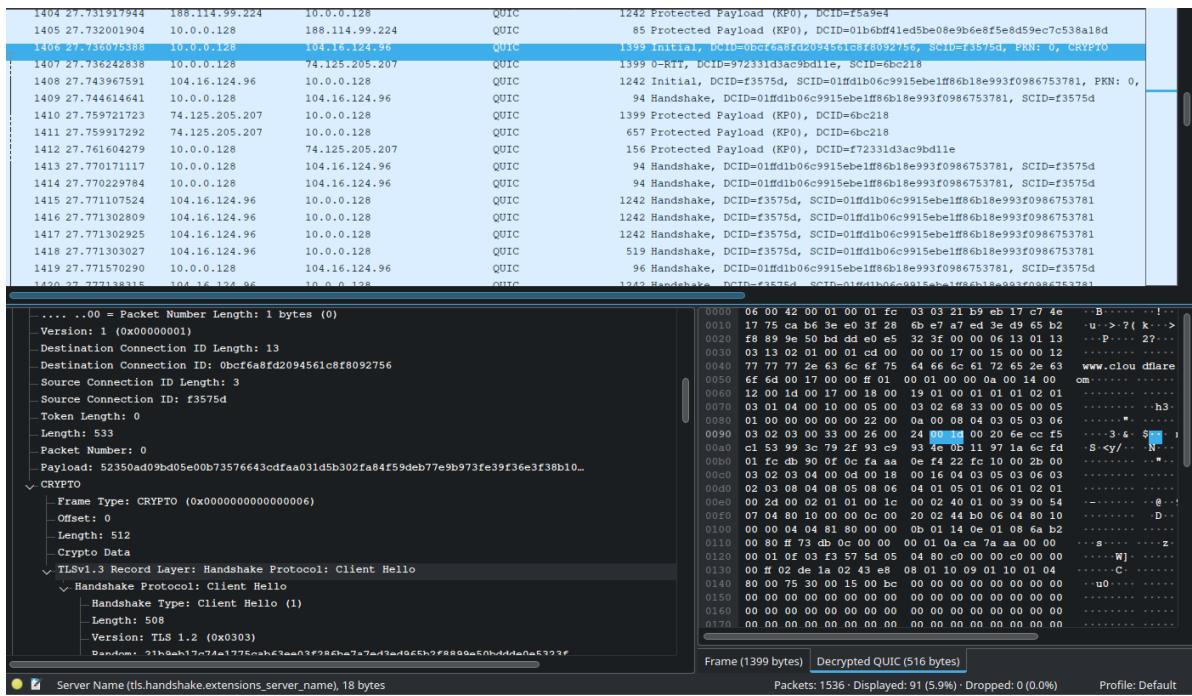


Рис. 3.16: quic

В QUIC соединения не связаны с TCP-соединениями, потому что QUIC работает по UDP. Вместо этого, соединения идентифицируются двумя значениями: SCID и DCID, соответственно Source Connection ID и Destination Connection ID. Благодаря этому, клиенты могут менять свой IP-адрес во время соединения, а также поддерживать много соединений параллельно. Мы видим, что когда сервер пишет нам, он говорит, что его SCID очень длинный – 20 байтов – в то время как для нас DCID всего лишь три байта. Это потому, что серверу может быть нужно поддерживать очень большое количество разговоров одновременно, и он может хранить какую-то важную для себя информацию в своих ID, в то время как нашему браузеру нужно держать в памяти сравнительно маленькое количество соединений.

Чтобы понять процедуру установления соединения в QUIC, я запустил свежую запись. В ней я получил одну транзакцию QUIC, которую сделал мой браузер на profile.accounts.firefox.com. Мы начинаем с пакета Initial, внутри которого мы кладем стандартный TLS Client Hello. Сервер отвечает с

Protected Payload (KP0), который можно расшифровать, если иметь доступ к Initial – в нем сервер отвечает своим TLS Server Hello. Также на этом уровне сервер подменяет выбранный клиентом DCID на тот, который сервер сам хочет использовать – когда клиент отправляет Initial, у него нет ID соединения, поэтому он генерирует его случайно. После этого клиент и сервер продолжают обмен Protected Payload (KP0), которые уже расшифровать нельзя, не зная деталей TLS-обмена ключами. Также существуют сообщения типа 0-RTT, которые видимо позволяют напомнить серверу о соединении, которое было открыто раньше, и не открывать это соединение заново.

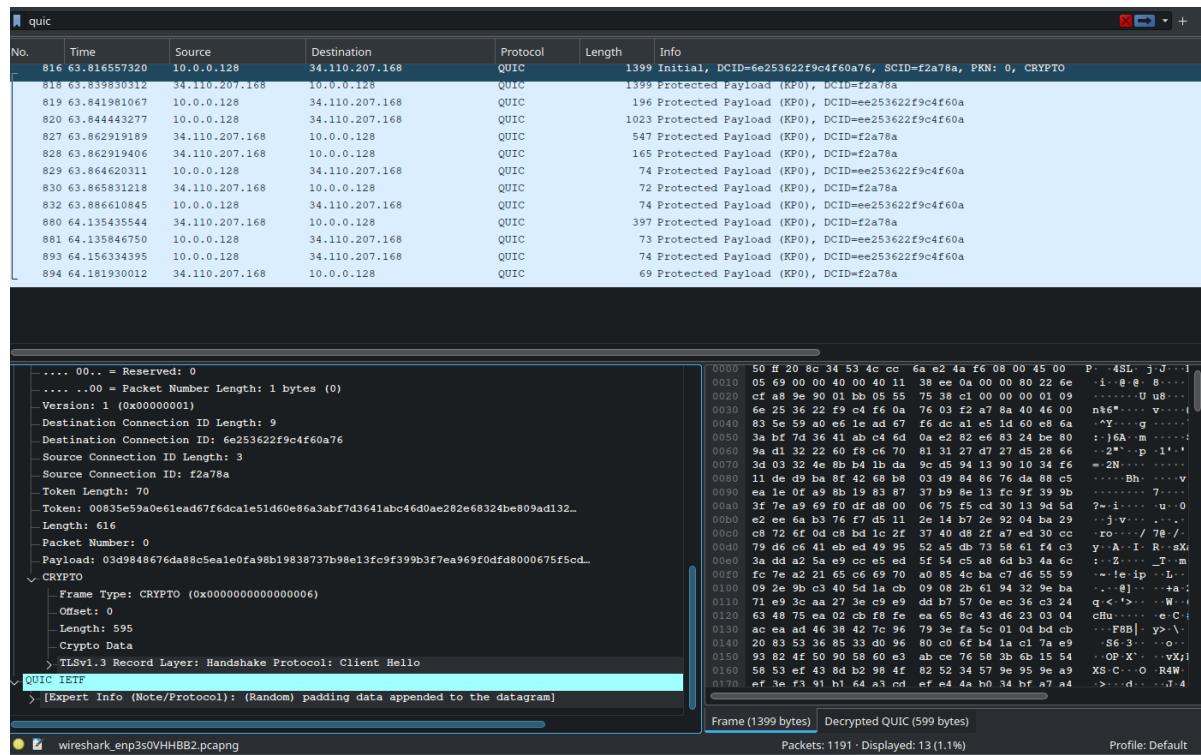


Рис. 3.17: quic

Теперь мы смотрим более детально на TCP-соединение, а именно то, как оно устанавливается. Для этого я сделал новую запись, в которой с помощью netcat переслал несколько пакетов между своим компьютером и своим домашним сервером на 10.0.0.2.

tcp.port == 12345						
No.	Time	Source	Destination	Protocol	Length	Info
51	8.141724470	10.0.0.128	10.0.0.2	TCP	74	60358 → 12345 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM TStamp=273789541 TSecr=62586258
52	8.142117857	10.0.0.2	10.0.0.128	TCP	74	12345 → 60358 [SYN, ACK] Seq=1 Win=65160 Len=0 MSS=1460 SACK_PERM TStamp=62586258
53	8.142176465	10.0.0.128	10.0.0.2	TCP	66	60358 → 12345 [ACK] Seq=1 Ack=1 Win=64256 Len=0 TStamp=273789542 TSecr=62586258
88	13.819632604	10.0.0.128	10.0.0.2	TCP	85	60358 → 12345 [PSH, ACK] Seq=1 Ack=1 Win=64256 Len=19 TStamp=273795219 TSecr=625826
89	13.819860747	10.0.0.2	10.0.0.128	TCP	66	12345 → 60358 [ACK] Seq=1 Ack=20 Win=65152 Len=0 TStamp=625832093 TSecr=273795219
247	20.661380638	10.0.0.2	10.0.0.128	TCP	85	12345 → 60358 [PSH, ACK] Seq=1 Ack=20 Win=65152 Len=19 TStamp=625838935 TSecr=27379
248	20.661405167	10.0.0.128	10.0.0.2	TCP	66	60358 → 12345 [ACK] Seq=20 Ack=20 Win=64256 Len=0 TStamp=273802061 TSecr=625838935
297	29.573083175	10.0.0.128	10.0.0.2	TCP	84	60358 → 12345 [PSH, ACK] Seq=20 Ack=20 Win=64256 Len=18 TStamp=273810972 TSecr=6258
298	29.573326876	10.0.0.2	10.0.0.128	TCP	66	12345 → 60358 [ACK] Seq=20 Ack=38 Win=65152 Len=0 TStamp=625847846 TSecr=273810972
306	30.349792776	10.0.0.128	10.0.0.2	TCP	66	60358 → 12345 [FIN, ACK] Seq=38 Ack=20 Win=64256 Len=0 TStamp=273811749 TSecr=62586
307	30.349817689	10.0.0.128	10.0.0.2	TCP	66	60358 → 12345 [RST, ACK] Seq=39 Ack=20 Win=64256 Len=0 TStamp=273811749 TSecr=62586

Рис. 3.18: tcp

Соединение начинается с SYN-пакета, который клиент отправляет серверу. В этом пакете он указывает, с каким портом он хочет общаться, в данном случае 12345.

No.	Time	Source	Destination	Protocol	Length	Info
51	8.141724470	10.0.0.128	10.0.0.2	TCP	74	60358 → 12345 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM TStamp=273789541 TSecr=62586258
52	8.142117857	10.0.0.2	10.0.0.128	TCP	74	12345 → 60358 [SYN, ACK] Seq=1 Win=65160 Len=0 MSS=1460 SACK_PERM TStamp=62586258
53	8.142176465	10.0.0.128	10.0.0.2	TCP	66	60358 → 12345 [ACK] Seq=1 Ack=1 Win=64256 Len=0 TStamp=273789542 TSecr=62586258
88	13.819632604	10.0.0.128	10.0.0.2	TCP	85	60358 → 12345 [PSH, ACK] Seq=1 Ack=1 Win=64256 Len=19 TStamp=273795219 TSecr=625826
89	13.819860747	10.0.0.2	10.0.0.128	TCP	66	12345 → 60358 [ACK] Seq=1 Ack=20 Win=65152 Len=0 TStamp=625832093 TSecr=273795219
247	20.661380638	10.0.0.2	10.0.0.128	TCP	85	12345 → 60358 [PSH, ACK] Seq=1 Ack=20 Win=65152 Len=19 TStamp=625838935 TSecr=27379
248	20.661405167	10.0.0.128	10.0.0.2	TCP	66	60358 → 12345 [ACK] Seq=20 Ack=20 Win=64256 Len=0 TStamp=273802061 TSecr=625838935
297	29.573083175	10.0.0.128	10.0.0.2	TCP	84	60358 → 12345 [PSH, ACK] Seq=20 Ack=20 Win=64256 Len=18 TStamp=273810972 TSecr=6258
298	29.573326876	10.0.0.2	10.0.0.128	TCP	66	12345 → 60358 [ACK] Seq=20 Ack=38 Win=65152 Len=0 TStamp=625847846 TSecr=273810972
306	30.349792776	10.0.0.128	10.0.0.2	TCP	66	60358 → 12345 [FIN, ACK] Seq=38 Ack=20 Win=64256 Len=0 TStamp=273811749 TSecr=62586
307	30.349817689	10.0.0.128	10.0.0.2	TCP	66	60358 → 12345 [RST, ACK] Seq=39 Ack=20 Win=64256 Len=0 TStamp=273811749 TSecr=62586

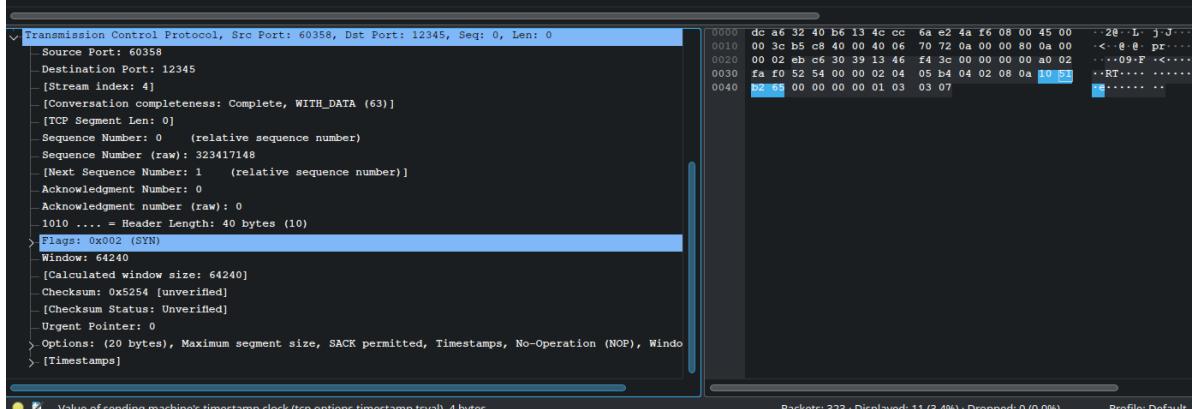


Рис. 3.19: tcp syn

Сервер отвечает пакетом SYN+ACK. Этим он говорит, что он получил запрос начать соединение, и готов это сделать.

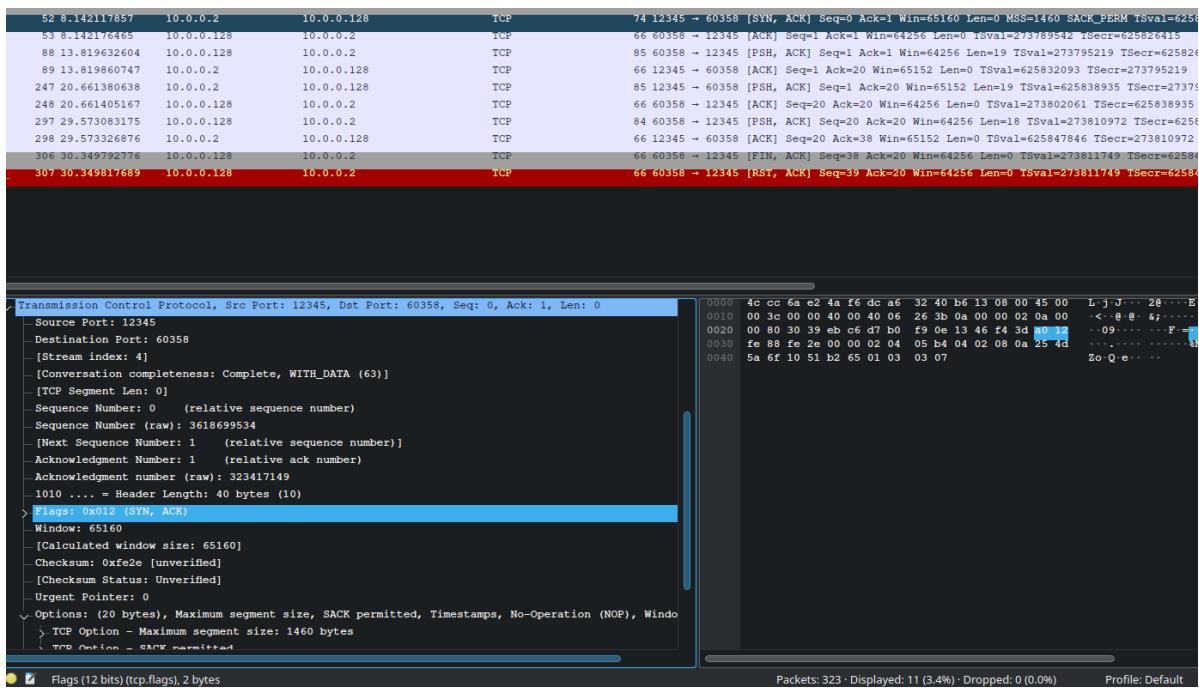


Рис. 3.20: tcp syn ack

Клиент затем отвечает пакетом ACK, что значит, что он теперь знает, что сервер подготовил соединение, и теперь клиент и сервер оба готовы отправлять пакеты.

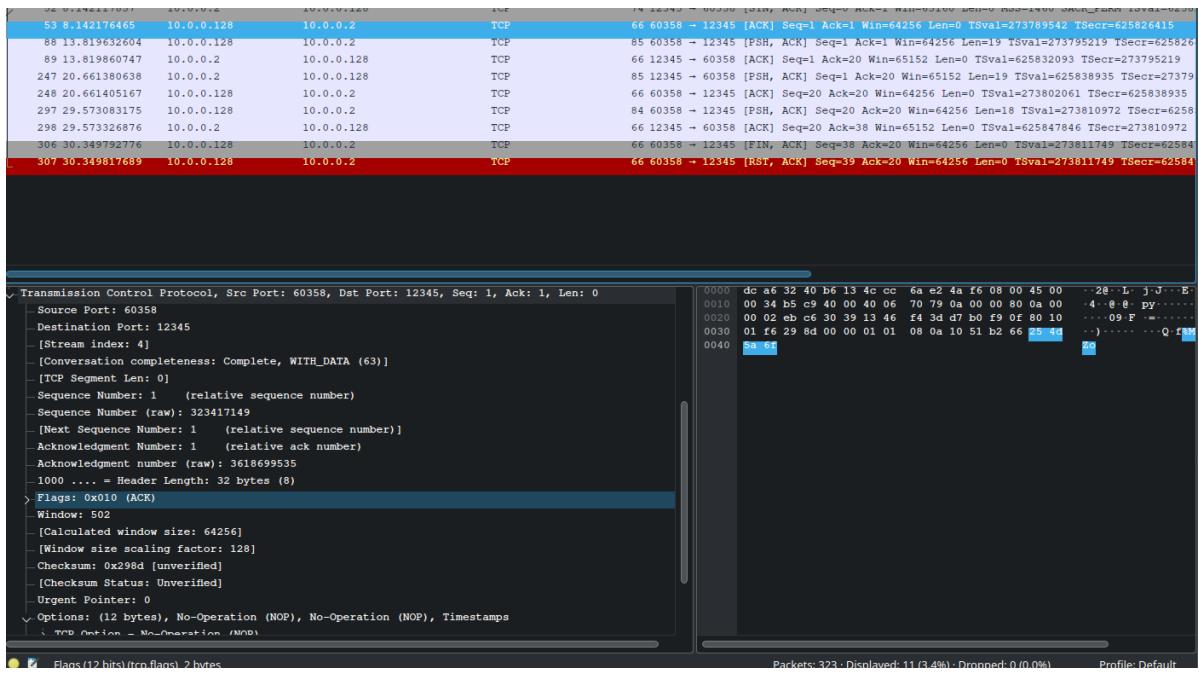


Рис. 3.21: tcp ack

После этого клиент может отправлять серверу пакеты, которые будут иметь флаги PSH+ACK, и этим он передает данные серверу и просит подтверждения получения. Сразу же после этого мы получаем от сервера пакеты ACK, которые говорят нам, что этот пакет был получен.

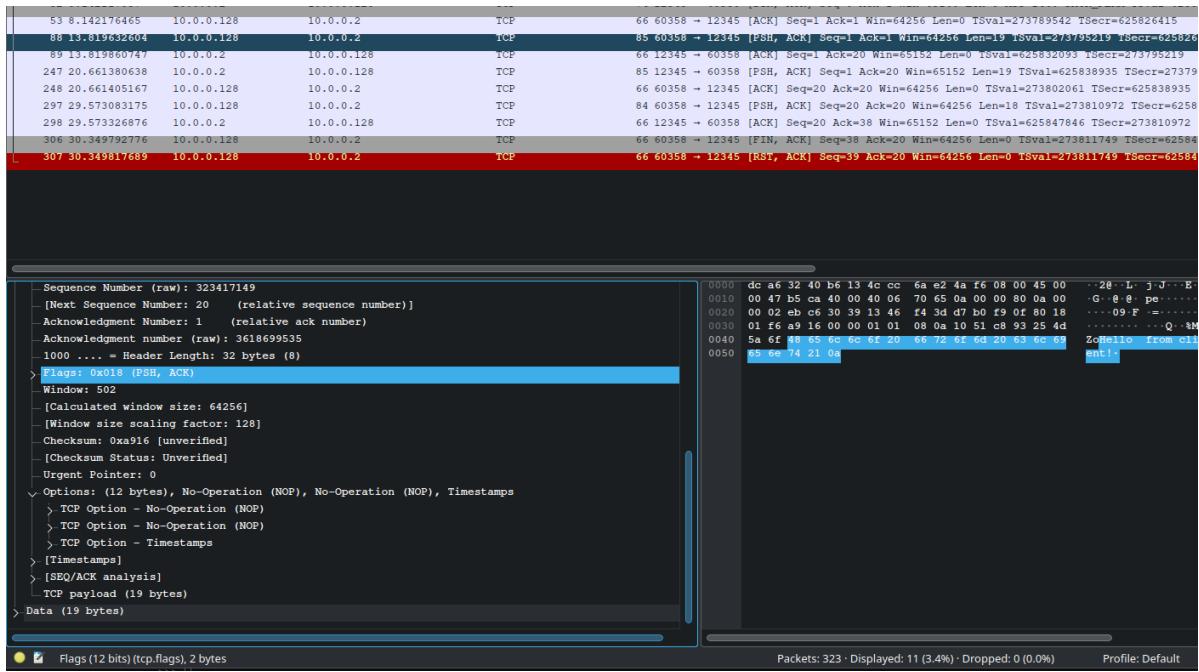


Рис. 3.22: tcp psh ack client

Сервер может делать то же самое, отправляя PSH+ACK-пакеты клиенту и передавая ему данные, и мы будем в ответ отправлять ACK.

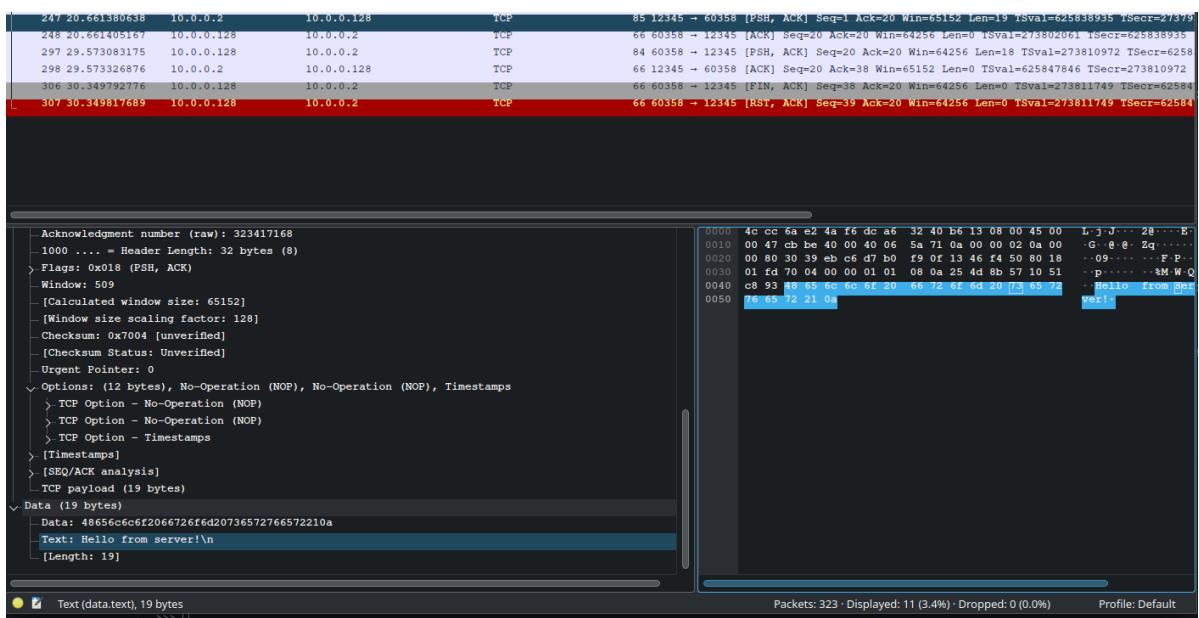


Рис. 3.23: tcp psh ack server

В конце концов, либо клиент, либо сервер закроют соединение. В данном случае это сделал клиент, отправив FIN+ACK-пакет, который говорит другому концу, что соединение завершается. Также, в этом случае, клиент после этого отправил RST+ACK-пакет, который говорит другому концу полностью забыть про это соединение. Это произошло потому, что я закрыл netcat с помощью Ctrl+C, и это оставило сетевой стек ядра с открытым TCP-соединением, которое больше никому не принадлежит. Если бы программа не закрылась так быстро, то мы бы увидели сообщение от сервера, которое подтверждает закрытие соединения.

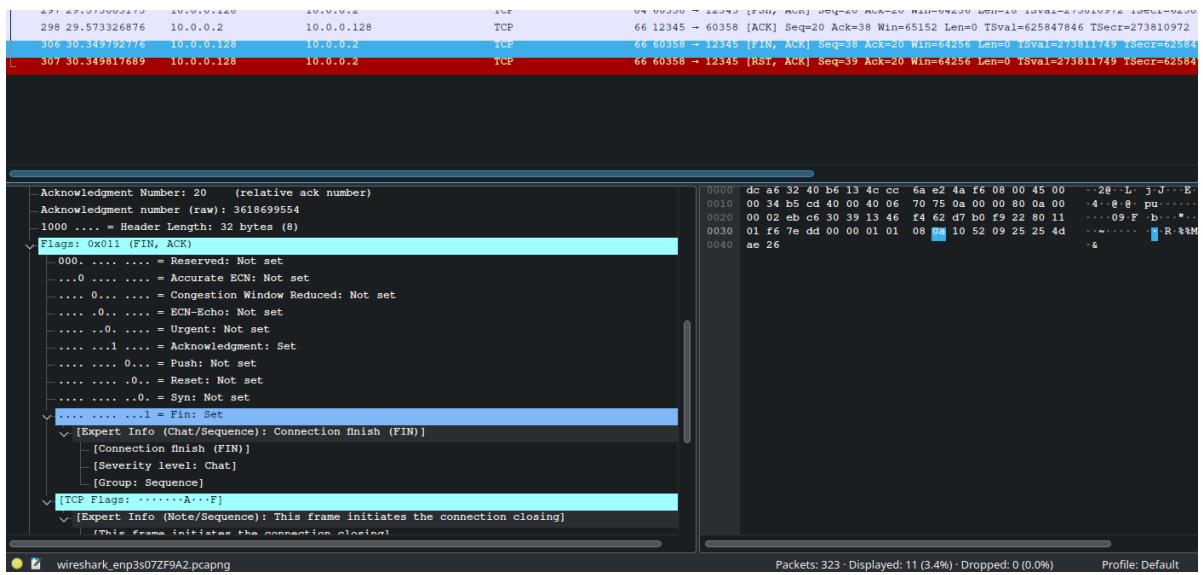


Рис. 3.24: tcp fin ack

4 Выводы

Я получил опыт работы с Wireshark для анализа пакетов в сети, а также с общими правилами работы различных сетевых протоколов.