

Сети

- TCP/UDP/ICMP, IP, Ethernet
- Маскарад, SNAT, DNAT
- DNS
- Linux и виртуализированные сети
- VXLAN

TCP/UDP/ICMP, IP, Ethernet

Статья про основные сетевые протоколы. Самое важное.

TCP

Попробуем отправить какой-нибудь текст через пс. "Слушатель" - 192.168.101.103, "клиент" - 192.168.101.104.

```
# Устанавливаем соединение
# Syn client -> server (привет, хочу подключиться к тебе)
14:14:32.476957 IP 192.168.101.104.40188 > 192.168.101.103.1234: Flags [S], seq 3022494785, win 64240,
options [mss 1460,sackOK,TS val 1927177028 ecr 0,nop,wscale 7], length 0
# Syn ack server -> client (привет, готов соединиться)
14:14:32.476984 IP 192.168.101.103.1234 > 192.168.101.104.40188: Flags [S.], seq 3324441448, ack
3022494786, win 65160, options [mss 1460,sackOK,TS val 3590889825 ecr 1927177028,nop,wscale 7], length 0
# ack client -> server (Я понял что ты готов соединиться)
14:14:32.476998 IP 192.168.101.104.40188 > 192.168.101.103.1234: Flags [.], ack 1, win 502, options
[nop,nop,TS val 1927177028 ecr 3590889825], length 0

# Отправляем данные
# Push data + ack client -> server (вот тебе данные, ответь что ты их принял)
14:15:24.138228 IP 192.168.101.104.40188 > 192.168.101.103.1234: Flags [P.], seq 1:3, ack 1, win 502,
options [nop,nop,TS val 1927228689 ecr 3590889825], length 2
# ack server -> client (да, я их принял)
14:15:24.138263 IP 192.168.101.103.1234 > 192.168.101.104.40188: Flags [.], ack 3, win 510, options
[nop,nop,TS val 3590941486 ecr 1927228689], length 0

# Закрываем соединение
# Final, ack client -> server (Был рад пообщаться, пока)
14:40:06.307980 IP 192.168.101.104.39352 > 192.168.101.103.1234: Flags [F.], seq 3, ack 1, win 502, options
[nop,nop,TS val 1928710859 ecr 3592422397], length 0
# Final, ack server -> client (Да, давай, пока)
14:40:06.308087 IP 192.168.101.103.1234 > 192.168.101.104.39352: Flags [F.], seq 1, ack 4, win 510, options
```

```
[nop,nop,TS val 3592423656 ecr 1928710859], length 0
```

```
# Ack client -> server (Услышал тебя. Конец)
```

```
14:40:06.308109 IP 192.168.101.104.39352 > 192.168.101.103.1234: Flags [.], ack 2, win 502, options
```

```
[nop,nop,TS val 1928710859 ecr 3592423656], length 0
```

Список флагов и их представление в tcpdump:

SYN	S	Флаг, который используется при запросе на соединение.
ACK	.	Используется при подтверждении пришедшего пакета.
FIN	F	Флаг устанавливается при нормальном закрытии соединения.
URGENT	U	Этот флаг нужен при передаче экстренных данных — например, при посылке <code>Ctrl+C</code> в telnet-соединении.
PUSH	P	Как правило, данный флаг устанавливается при передаче пользовательских данных.
RESET	R	Немедленный разрыв соединения.
Заполнитель	none	В случае, если в пакете отсутствует какой-либо флаг используется данный заполнитель.

Из чего состоит пакет:

- Порт src/dst
- Порядковый номер
- Номер подтверждения (ACK SN)
- Длина заголовка
- Флаги (см. таблицу выше)
- Размер окна
- Чексумма
- Уровень важности

UDP

Соединение не устанавливается и не закрывается! Мы просто кидаем данные. UDP не гарантирует, что пакет дойдет, но гарантирует, что пакет будет проверен, в пакете UDP предусмотрена чексумма.

```
# Запустили nc
# *Никакие пакеты никуда не полетели, открытия соединения не предусмотрено протоколом*

# Отправили текст
14:17:48.336527 IP 192.168.101.104.59808 > 192.168.101.103.1234: UDP, length 4

# Закрыли nc клиент
# *Опять же никаких пакетов, т.к. не предусмотрено закрытия соединения*
```

ICMP

Ping не подразумевает передачу данных. Он подразумевает передачу самих пакетов. Т.е. условно дошел-не дошел.

```
# Живой?
14:59:26.330559 IP 192.168.101.104 > 192.168.101.103: ICMP echo request, id 46223, seq 1, length 64
# Да, живой
14:59:26.330593 IP 192.168.101.103 > 192.168.101.104: ICMP echo reply, id 46223, seq 1, length 64
```

Также можно указать размер пакета в утилите ping. Если мы укажем размер больше Ethernet фрейма (MTU), то будет уже 2 пакета:

```
# MTU = 1500
root@test2:/# ip l show eth0
2: eth0@if7: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP mode DEFAULT group
default qlen 1000
    link/ether 00:16:3e:83:51:d8 brd ff:ff:ff:ff:ff:ff link-netnsid 0

root@test2:/# ping -s 1400 192.168.101.103
PING 192.168.101.103 (192.168.101.103) 1400(1428) bytes of data.
1408 bytes from 192.168.101.103: icmp_seq=1 ttl=64 time=0.042 ms
1408 bytes from 192.168.101.103: icmp_seq=2 ttl=64 time=0.063 ms
1408 bytes from 192.168.101.103: icmp_seq=3 ttl=64 time=0.048 ms

15:01:13.453126 IP 192.168.101.104 > 192.168.101.103: ICMP echo request, id 29861, seq 3, length 1480
15:01:13.453152 IP 192.168.101.104 > 192.168.101.103: ip-proto-1
15:01:13.453166 IP 192.168.101.103 > 192.168.101.104: ICMP echo reply, id 29861, seq 3, length 1480
```

Датаграмму можно посмотреть тут: <https://ru.wikipedia.org/wiki/ICMP>

Однако другие типы пакетов ICMP могут нести в себе какие-либо данные. [Вот хитрая статья о том как сделать проксирование через ICMP.](#)

IP (IPv4)

Дословно - Internet Protocol, межсетевой протокол. Если L4 (TCP/UDP) доставляет данные, то L3 (IP) объединяет сети и в целом **организует сетевое пространство**. С его помощью можно понять, как попасть к тому или иному хосту, как маршрутизировать трафик. **IP не гарантирует надежности доставки пакетов**, т.е. они могут продублироваться, повредиться, быть получены не в том порядке, что их отправили или вообще не прийти. За это отвечают протоколы более высокого уровня, например TCP.

Важные компоненты IP пакеты:

- TTL - время жизни пакета. Это количество узлов, которое может пройти пакет. Он полезен для предотвращения заикливания пакетов и чтобы пакет не искал нужный хост до бесконечности. При прохождении узла число уменьшается на единицу. Однако можно настроить маршрутизатор так, чтобы он не уменьшал TTL. Этой возможностью пользуются для обхода блокировки раздачи интернета через с мобильного телефона, так как некоторые операторы опираются на TTL при выявлении раздачи интернета. Также по этому принципу работаете traceroute/tracert - он отправляет пакеты каждый раз увеличивая TTL, пока не будет отвечать один и тот же хост. Когда приходит пакет с TTL = 0, то получивший его хост отбрасывает пакет обратно с сообщением `Time Exceeded` (ICMP тип 11 код 0).
- Длина пакета. Минимальное корректное значение для этого поля равно 20, максимальное — 65 535. В длине пакета учитывается заголовок и сами данные. Так как заголовок содержит 20 байт данных, то IP пакет **не может быть меньше 20 байт**.
- Протокол. TCP, UDP, ICMP.
- Контрольная сумма.

Ethernet (802.3)

Ethernet это довольно простой протокол. Состоит он из:

- MAC заголовок (14 байт): Отправитель, получатель, EtherType
- Данные (46 - 1500 байт): Например, IPv4 внутри
- CRC checksum (4 байта)

Надо не забывать, что Ethernet **ходит между хостами и не выходит за их пределы**. Т.е. буквально в пределах одного кабеля. Когда пакет приходит на роутер, он уже создает другой пакет к следующему хосту. И **бroadкаста в рамках Ethernet никакого нет**. Этот протокол работает на уровне сетевой карты/ сетевого оборудования, ОС не участвует в формировании Ethernet пакета.

VLAN

TODO

Сетевые модели

Чаще всего используют 2 модели: OSI из 7 уровней и DOD (TCP/IP) из 4х уровней. А иногда из 5 уровней. А иногда вообще из 3х уровней.

OSI

Хоть эта модель не особо совместима с сегодняшними реалиями, она используется повсеместно и строго стандартизирована.

Чаще всего упоминают 4 уровня:

- L7 - Прикладной уровень. Это уровень самого протокола, например http, smtp, ftp
- L4 - Транспортный уровень. Это TCP, UDP.
- L3 - Сетевой уровень. Это адресация, т.е. IPv4, IPv6.
- L2 - Канальный уровень. 802.3 или Ethernet (всеми любимая "медь"), PON (Оптоволокно, которое тянут сегодня в квартиры), 802.11 (вайфай), PPPoE, PPP, MPLS. **Fun fact:** WinBox, приложение для настройки оборудования Mikrotik (а также мобильное приложение), умеет подключаться на уровне L2, т.е., например, по MAC адресу Ethernet порта. Поэтому если вы примените неправильные настройки адресации, то можно будет подключиться по MAC адресу.

DOD

Это не очень популярная и стандартизированная модель, однако она более простая и актуальная. В ней есть всего 4 уровня:

- Прикладной (HTTP, FTP, SMTP)
- Транспортный (TCP, UDP)
- Сетевой (IPv4, IPv6)
- Канальный (802.3, 802.11)

В целом то, зачастую все эти 4 уровня и используются и большего не надо, но если заглянуть [вот сюда](#), то можно ужаснуться тому как модель DOD меняется от автора к автору. Хотя суть и одна, но везде они разные.

Смотрим трафик, отражаем атаки

Нам помогут 2 утилиты: tcpdump и ss. С помощью первой можно слушать трафик, с помощью второй смотреть соединения.

Краткий ликбез по tcpdump:

- `-n` : не резолвить адреса и домены, т.е. вместо yandex.ru:https будет 77.88.55.88:443. Это практически обязательная опция, так как резолвинг PTR записей адресов занимает неприлично много времени и практической пользы никакой, соотв-но трафик мы увидим с большой задержкой
- `-c` : количество IP пакетов (в штуках), которое мы хотим захватить
- `-i` : интерфейс, который мы хотим послушать. Можно указать any и мы будем слушать все существующие в системе интерфейсы
- `-v` : вербозность вывода. Можно указать -vv, -vvv, будет еще больше данных выводиться. При -v кроме Ethernet пакета будет выведен еще и TCP/UDP пакет.
- `-X` : будут выведены данные из пакета в ASCII и hex виде. Иногда может пригодиться, но более удобным будет параметр -A
- `-A` : будут выведены данные из пакетов в ASCII формате. Полезно для просмотра http/smtp/ftp или любого другого нешифрованного трафика.
- `-w file.dump` : Записать дамп в файл. Потом его можно будет открыть, например, в Wireshark или в том же tcpdump
- `-r file.dump` : Прочитать дамп из файла. Можно также применять различные фильтры и параметры, как при просмотре трафика вживую.
- `-t, -tt, -ttt, -tttt, -ttttt` : разные форматы вывода времени. Лучше посмотри их в man'e

Вот пример трафика:

```
losted@pg-new:~$ sudo tcpdump -v -n -i br0 port 1234
tcpdump: listening on br0, link-type EN10MB (Ethernet), snapshot length 262144 bytes
# Открыли соединение
17:46:20.818195 IP (tos 0x0, ttl 64, id 12548, offset 0, flags [DF], proto TCP (6), length 60)
    192.168.101.104.57202 > 192.168.101.103.1234: Flags [S], cksum 0x4c4f (incorrect -> 0x6a63), seq
524276269, win 64240, options [mss 1460,sackOK,TS val 1939885369 ecr 0,nop,wscale 7], length 0
17:46:20.818225 IP (tos 0x0, ttl 64, id 0, offset 0, flags [DF], proto TCP (6), length 60)
    192.168.101.103.1234 > 192.168.101.104.57202: Flags [S.], cksum 0x4c4f (incorrect -> 0xd5d5), seq
4126030036, ack 524276270, win 65160, options [mss 1460,sackOK,TS val 3603598166 ecr
1939885369,nop,wscale 7], length 0
17:46:20.818242 IP (tos 0x0, ttl 64, id 12549, offset 0, flags [DF], proto TCP (6), length 52)
    192.168.101.104.57202 > 192.168.101.103.1234: Flags [.], cksum 0x4c47 (incorrect -> 0x0135), ack 1, win
502, options [nop,nop,TS val 1939885369 ecr 3603598166], length 0

# Отправили данные
17:46:23.889145 IP (tos 0x0, ttl 64, id 12550, offset 0, flags [DF], proto TCP (6), length 58)
    192.168.101.104.57202 > 192.168.101.103.1234: Flags [P.], cksum 0x4c4d (incorrect -> 0xb14b), seq 1:7,
ack 1, win 502, options [nop,nop,TS val 1939888440 ecr 3603598166], length 6
17:46:23.889198 IP (tos 0x0, ttl 64, id 19860, offset 0, flags [DF], proto TCP (6), length 52)
    192.168.101.103.1234 > 192.168.101.104.57202: Flags [.], cksum 0x4c47 (incorrect -> 0xe928), ack 7, win
510, options [nop,nop,TS val 3603601237 ecr 1939888440], length 0

# Закрыли соединение
17:46:28.637162 IP (tos 0x0, ttl 64, id 12551, offset 0, flags [DF], proto TCP (6), length 52)
    192.168.101.104.57202 > 192.168.101.103.1234: Flags [F.], cksum 0x4c47 (incorrect -> 0xd6a3), seq 7, ack
1, win 502, options [nop,nop,TS val 1939893188 ecr 3603601237], length 0
17:46:28.637432 IP (tos 0x0, ttl 64, id 19861, offset 0, flags [DF], proto TCP (6), length 52)
    192.168.101.103.1234 > 192.168.101.104.57202: Flags [F.], cksum 0x4c47 (incorrect -> 0xc40e), seq 1, ack
8, win 510, options [nop,nop,TS val 3603605985 ecr 1939893188], length 0
17:46:28.637448 IP (tos 0x0, ttl 64, id 12552, offset 0, flags [DF], proto TCP (6), length 52)
    192.168.101.104.57202 > 192.168.101.103.1234: Flags [.], cksum 0x4c47 (incorrect -> 0xc416), ack 2, win
502, options [nop,nop,TS val 1939893188 ecr 3603605985], length 0
```

На первой строке - IP, на второй строке (которая с табуляцией) - TCP. Зачастую глупые атаки это флуд однотипными пакетами. Например, у них может быть один и тот же **ТТЛ, флаги, длина IP пакета, длина TCP пакета, длина окна**. Правила для iptables на основе этих параметров можно легко сформировать.

Также можно посмотреть данные внутри и заблокировать через модуль `string` в iptables.

TODO TCP Syn cookie

Маскарад, SNAT, DNAT

Зачем нам все это? Чтобы держать подсетку за одним адресом.

- SNAT - преобразование **исходящего трафика** 1 к 1. Т.е. локально на хосте 10.0.0.2, а во внешнюю сеть он ходит через 8.8.8.8. Несколько хостов за один адрес нельзя пустить, только 1.
- DNAT - преобразование **входящего трафика** (Много из внешки к Одному локальному). Можно пробросить как все порты, так и какой-то конкретный.
- PAT/NAPT/Masquerade - преобразование много адресов к одному + много портов. С помощью этой технологии работают домашние роутеры. Т.е. клиент за маскарадом может открыть у себя любой порт и установить с него соединение с хостом во внешней сети и его порт будет автоматически проброшен при соединении на такой же порт, только уже во внешней сети. Сокет (связка адрес+порт) должна быть уникальна, только тогда получится установить соединение.

SNAT+DNAT

Например, у нас есть host-vm, внутри которой есть внешний интерфейс eth0 с адресами 192.168.101.4/24

и 192.168.101.5/24 и бридж br0, на котором висит контейнер test2 со внутренним адресом 10.13.37.2/24. Мы хотим, чтобы по адресу 192.168.101.5 мы попадали на 10.13.37.2.

Так сделано, например, у облачных провайдеров, например AWS или Yandex.Cloud.

```
# Хост машина с контейнером внутри
losted@host-vm:~$ ip -4 a show eth0
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default qlen 1000
    altname enp0s18
    inet 192.168.101.4/24 brd 192.168.101.255 scope global eth0
        valid_lft forever preferred_lft forever
    inet 192.168.101.5/24 scope global secondary eth0
        valid_lft forever preferred_lft forever

losted@host-vm:~$ ip -h -4 a show br0
4: br0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group default qlen 1000
    inet 10.13.37.1/24 scope global br0
```

```
valid_lft forever preferred_lft forever
```

```
# Контейнер внутри
```

```
root@test2:/# ip a show eth0
```

```
2: eth0@if7: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group default qlen 1000
```

```
link/ether 00:16:3e:83:51:d8 brd ff:ff:ff:ff:ff:ff link-netnsid 0
```

```
inet 10.13.37.2/24 scope global eth0
```

```
valid_lft forever preferred_lft forever
```

```
# Пускаем трафик из сети 10.13.37.0/24 во вне через адрес 192.168.101.5
```

```
losted@host-vm:~$ sudo iptables -t nat -A POSTROUTING -o eth0 -s 10.13.37.0/24 -j SNAT --to-source 192.168.101.5
```

```
# Все что приходит на 192.168.101.5 перенаправляем на 10.13.37.2
```

```
losted@host-vm:~$ sudo iptables -t nat -A PREROUTING -d 192.168.101.5 -j DNAT --to-destination 10.13.37.2
```

```
# Проверяем
```

```
losted in ~ λ ping 192.168.101.5
```

```
PING 192.168.101.5 (192.168.101.5) 56(84) bytes of data.
```

```
64 bytes from 192.168.101.5: icmp_seq=1 ttl=63 time=3.32 ms
```

```
64 bytes from 192.168.101.5: icmp_seq=2 ttl=63 time=5.97 ms
```

```
root@test2:/# nc -l 123
```

```
dsf
```

```
losted in ~ λ nc 192.168.101.5 123
```

```
dsf
```

```
# Контейнер видит подключение с правильного адреса
```

```
root@test2:/# ss -tpn
```

State	Recv-Q	Send-Q	Local Address:Port	Peer
Address:Port				
				Process
ESTAB	0	0	10.13.37.2:123	
192.168.101.105:49352				users(("nc",pid=673,fd=4))

```
losted in ~ λ ip -h -4 a show wlp2s0
```

```
4: wlp2s0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group default qlen 1000
```

```
inet 192.168.101.105/24 brd 192.168.101.255 scope global dynamic noprefixroute wlp2s0
```

```
valid_lft 4294sec preferred_lft 4294sec
```

Теперь при обращении к любому порту по 192.168.101.5 будет перенаправлено на 10.13.37.2.

Masquerade + DNAT/SNAT

```
iptables -t nat -A POSTROUTING -o ppp0 -j MASQUERADE
```

DNS

TODO

<https://habr.com/ru/articles/137587/>

Linux и виртуализированные сети

veth, macvlan, etc

VXLAN

VXLAN позволяет соединить несколько нод между собой. Я посмотрел на то как работает фланель в кубе и мне понравилось.

Настроить ручками:

```
# host1:
ip l add vxlan0 type vxlan id 42 group 239.1.1.1 dev enp5s0 dstport 4789
ip a add 10.99.0.1/24 dev vxlan0
ip l set vxlan0 up
ping 10.99.0.2
ping 10.99.0.3

# host2:
ip l add vxlan0 type vxlan id 42 group 239.1.1.1 dev enp5s0 dstport 4789
ip a add 10.99.0.2/24 dev vxlan0
ip l set vxlan0 up
ip l add br0 type bridge
ip a add 10.99.0.3/24 dev br0
ip l set br0 up
ip link set vxlan0 master br0
ip -d l show vxlan0

# Между собой пингуются. Можно даже пинговать с первого хоста адреса на мосту второго хоста.
```

На третьем хосте можно тоже vxlan настроить и он тоже будет работать. Еще есть такая полезная вещь как vrf, чтобы объединить на третьем хосте vxlanы, мост, интерфейс, что угодно.

```
ip link add vrf0 type vrf table 10
ip link add br0 type bridge
ip link set br0 master vrf0
```