GeekBrains

Введение в UNIX-системы

Сетевые возможности Linux

Базовые знания о сетях и протоколах стека TCP/IP. Настройка сети вручную: настройка IP-адреса, шлюза по умолчанию, маски подсети. Встроенный фаервол iptables.

Оглавление

Введение

Темы сети и безопасности далеко выходят за рамки курса GNU / Linux. Это и понятно: далеко не одни UNIX-подобные системы используют сеть, а стек протоколов TCP/IP на данный момент является наиболее универсальным механизмом, реализованным во всех операционных системах. Глубокие знания по стеку TCP/IP и сетевой безопасности невозможно получить в рамках одного занятия, поэтому в блоке «Системное администрирование» предусмотрено два курса, необходимых для полноценного освоения профессии системного администратора (да и вообще любому профессиональному IT-разработчику):

1. TCP/IP, где подробно разбирается стек протоколов с самого нижнего уровня (физического в модели OSI) и до верхнего — прикладного. Подробно рассматриваются отличия между уровнями и выполняемыми на каждом из них задачами, сетевые протоколы, механизмы адресации (на канальном, сетевом уровне, идентификация приложений на транспортном).

2. Сетевая безопасность, где к знаниям по стеку протоколов добавляются представления и практика по сетевой безопасности: типы уязвимостей тех или иных протоколов, методах атак, навыках по поиску и закрытию уязвимостей.

Вышеописанные курсы тесно связаны между собой: чтобы работать с безопасностью сети, надо понимать, как сеть устроена.

В этом уроке мы рассмотрим самое общее представление о сети, а также разберем те вещи, которые касаются именно администрирования в GNU / Linux — то есть по большей части сетевые инструменты операционной системы и их использование.

Сеть

Существуют две популярные многоуровневые сетевые модели OSI/ISO (ЭМВОС) и TCP/IP.

Идея многоуровневой модели и стека как ее реализации заключается в том, что логика работы сетевых механизмов изолируется на каждом конкретном уровне. Это происходит таким образом, что каждый уровень на одной машине работает с аналогичным на другой, не задумываясь о реализации нижестоящих уровней. Каждый уровень имеет интерфейс на уровень ниже, обращаясь к которому он реализует собственные возможности. Соответственно, чтобы организовать надежную доставку, приложение открывает TCP-сокет, указывает IP-адрес и порт сервера. При успешном подключении появляется надежный способ передачи в обе стороны, который используется для реализации прикладного протокола. На прикладном уровне можно не беспокоиться о тройном рукопожатии, управлении окном, ретрансмиссиях и дупликациях. Напротив, на транспортном уровне, например при реализации протокола TCP, уже не имеет значения, какой прикладной протокол используется выше и какие протоколы будут использоваться ниже. TCP хорошо работает и через Ethernet, и через Wi-Fi, если взять канальный уровень.

Модель OSI/ISO

Модель OSI/ISO — это академическая попытка создать универсальную модель взаимодействия систем. На практике она потерпела поражение перед стеком TCP/IP, ставшим практической реализацией стека сетевых технологий. Тем не менее модель OSI/ISO иногда бывает удобна для описаний, и нижние 4 уровня применяются для описания сетевых устройств.

OSI/ISO расшифровывается как Open Systems Interconnection model (OSI model), стандарт ISO/IEC 7498-1.

Также есть русское соответствие, ЭМВОС — эталонная модель взаимодействия открытых систем, отечественный стандарт ГОСТ Р ИСО/МЭК 7498-1-99. Разработана в 1978 году и состоит из семи уровней, которые частично соотносятся с действительно применяемыми технологиями.

Уровень	Название
7 уровень	Прикладной уровень
6 уровень	Уровень представления
5 уровень	Сеансовый уровень
4 уровень	Транспортный уровень
3 уровень	Сетевой уровень
2 уровень	Канальный уровень
1 уровень	Физический уровень

Рассмотрим назначение уровней:

1. **Физический** уровень представлен физическими способами передачи информации. Витая пара (две пары, четыре пары), коаксиал, оптоволокно, радиоканал и т.д.

2. **Канальный** уровень предназначен для взаимодействия устройств через физическую среду передачи. Он принимает нужные кадры или фреймы (так называются блоки данных вместе с заголовками на канальном уровне), отбрасывает ненужные и контролирует целостность.

3. **Транспортный** уровень предназначен для передачи данных от отправителя к получателю. Позволяет управлять надежностью передачи. Запрашивает интерфейс у сетевого уровня.

4. **Сетевой** уровень предназначен для глобальной логической адресации узлов и для маршрутизации. Запрашивает интерфейс у канального.

5. **Сеансовый** уровень необходим для установки и управления сеансами связи. Запрашивает интерфейс у транспортного уровня.

6. Уровень **представления** предназначен для перекодирования и преобразования форматов данных. Запрашивает интерфейс у сеансового уровня.

7. **Прикладной** уровень предоставляет интерфейс к сетевым услугам для прикладного программного обеспечения. Сам же запрашивает интерфейс у уровня представления.

Протоколы TCP/IP не имеют однозначного сопоставления с моделью OSI/ISO.

Иногда протоколы RIP и BGP относятся к прикладному уровню, но чаще к сетевому. Протокол DNS — к прикладному, но иногда его относят к сеансовому. Протоколы SSL и TLS — чаще всего к уровню представления, но иногда и к сеансовому. Транспортный протокол TCP также обладает сеансовыми чертами. Протокол ARP относят либо к сетевому, либо к канальному уровню (на самом деле он в промежуточном положении). Консенсус есть по протоколам канального уровня (IEEE 802.3 Ethernet, IEEE 802;11 Wi-Fi, ITU G.992.1 ADSL, ITU G 991.1 HDSL), основных протоколов сетевого уровня (IP, ICMP, IGMP), протоколов транспортного уровня (TCP, UDP). Большинство протоколов относятся к

прикладному уровню (FTP/FTPS, SFTP/SSH, HTTP/HTTPS, SMTP, POP3, IMAP4 и т. д.). Туннелирующие протоколы часто относят к сеансовому уровню, хотя их местоположение в стеке OSI/ISO не может быть строго определенным — из-за дублирования уровней модели OSI/ISO внутри тоннеля и вне его (PPTP, L2TP, OpenVPN).

Примерная схема расположения протоколов в модели OSI/ISO:

Стек TCP/IP

В отличие от модели OSI/ISO, стек TCP/IP получил распространение на практике. Он состоит из четырех уровней. До стека TCP/IP был протокол NCP (Network Control Protocol), используемый в ARPANET — предшественнике интернета. TCP/IP назван по двум важным протоколам: TCP и IP, — но в стек входят и другие протоколы: UDP, ICMP и т. д.

Уровни стека TCP/IP:

Уровень	Название
4 уровень	Прикладной уровень
3 уровень	Транспортный уровень
2 уровень	Сетевой уровень
1 уровень	Уровень сетевых интерфейсов

Большинство протоколов, используемых приложениями, принадлежат к прикладному уровню. HTTP, FTP, SSH/SFTP, SMTP, POP3, IMAP4, XMPP, а также DNS, DHCP, NTP, SNMP — прикладные протоколы. По большей части любое приложение может реализовать свой протокол, и он будет протоколом прикладного уровня. Прикладные протоколы используют тот или иной протокол транспортного уровня в зависимости от задач. Если нужна надежная передача файлов большого

объема (архив, приложения) либо механизм сессии (как ssh), используется TCP. Если нужна отправка коротких сообщений или мультимедийного потока, для которого не критична потеря пакетов, — UDP.

Уровень	Название	Протоколы
4 уровень	Прикладной уровень	HTTP, HTTPS, SMTP, IMAP, POP3 DNS, DHCP, BOOTP, NTP, SNMP, SSH, SFTP, FTP, FTPS, Telnet
3 уровень	Транспортный уровень	TCP, UDP
2 уровень	Сетевой уровень	IP, ICMP, ARP
1 уровень	Уровень сетевых интерфейсов	IEEE 802.3 Ethernet, IEEE 802.11 Wi-Fi

На транспортном уровне решается вопрос идентификации приложений и надежности доставки. И TCP, и UDP в заголовке содержат двухбайтный порт получателя и порт отправителя. Существует два набора портов: TCP-порты (в системе именуются STREAM) и UDP-порты (DGRAM). Именно по номеру порта операционная система понимает, какому из работающих приложений должны быть доставлены полученные системой данные.

При использовании UDP данные помещаются полностью в UDP-дейтаграмму. При использовании TCP созданная сессия автоматически разбивает данные на набор TCP-сегментов, которые передаются по сети и собираются на другой стороне в исходный поток байт. Существуют и другие протоколы транспортного уровня, но фактически в большинстве случаев используются два: TCP и UDP.

На сетевом уровне решается вопрос идентификации хостов и маршрутизации. Основной протокол — IP (Internet Protocol). Данные снабжаются заголовком, где указываются IP-адреса получателя и отправителя. На сетевом уровне обычно говорят об IP-пакетах или IP-дейтаграммах (реже). Это синонимы. Также к сетевому уровню относятся вспомогательные протоколы, такие как ICMP (для идентификации о сетевых проблемах), IGMP (управление группами широковещательных рассылок, например для потокового вещания IPTV), ARP (преобразование IP-адресов в MAC-адреса), протоколы маршрутизации. При этом ICMP вкладывается в IP-пакет. А некоторые протоколы маршрутизации технически могут быть выполнены как протоколы прикладного уровня (могут использовать интерфейс транспортного уровня TCP, UDP), тем не менее при этом входить в механизм сетевого уровня. Протокол ARP работает сразу поверх канального уровня и служит для поиска MAC-адресов для искомых IP-адресов своей сети (или шлюза).

Уровень сетевых интерфейсов (соответствует канальному и физическому уровню модели OSI) представлен протоколами, работающими через конкретное сетевое оборудование. Задача протоколов данного уровня — получить информацию от вышестоящего уровня и передать ее через физическую среду. В отличие от сетевого, на данном уровне не решается задача глобальной доставки, только локальной (внутри своей сети). Для идентификации сетевых интерфейсов на данном уровне используются MAC-адреса. Блоки данных здесь называют кадрами, или фреймами (это синонимы): например, Ethernet-кадр. Популярные протоколы канального уровня: Ethernet, Wi-Fi. Что интересно, соответствующие стандарты IEEE 802.3 (Ethernet), IEEE 802.11 (Wi-Fi) описывают механизмы и канального, и физического уровня совместно — это соответствует принципам TCP/IP.

Сетевые устройства

Так сложилось, что при разработке программного обеспечения удобнее использовать модель TCP/IP, а при работе с сетевой инфраструктурой — модель OSI/ISO, по крайней мере ее нижние уровни.

По этому принципу выделяют устройства соответствующего уровня:

1. Устройства 1-го уровня: концентратор (хаб), репитер. Не анализирует кадры, передает данные побитно.

2. Устройства 2-го уровня: коммутаторы (свитчи), сетевые интерфейсы. Анализируют заголовки кадров, сопоставляют мак-адреса.

3. Устройства 3-го уровня: маршрутизаторы. Анализируют IP-заголовки, осуществляют маршрутизацию IP-пакетов.

4. Устройства 4-го уровня. Например, шлюз с NAPT-трансляцией, подменяющий «серые адреса» на «белые», используя сопоставление «динамический порт — IP-адрес» во внутренней сети.

Соотношение моделей

При сопоставлении нижних пяти уровней модели OSI и трех уровней TCP/IP в основном проблем не возникает. В выделении физического и канального уровней в уровне сетевых интерфейсов — тоже. Понятно, что MAC-адреса и сама структура кадра — это канальный уровень, а частота для радиоканала, физическое подключение коннектора в соответствующий порт, количество пар в витой паре и порядок их подключения — это физический уровень. Но при этом на практике невозможно в прикладном уровне стека TCP/IP выделить прикладной, представления и сеансовый уровень модели OSI.

С другой стороны, и у модели TCP/IP есть ограничения. Так, HTTPS является протоколом HTTP, вложенным в протокол шифрования TLS. Фактически, TLS занимает промежуточное положение в стеке протоколов TCP/IP между прикладным и транспортным, но такого уровня нет. Поэтому часто говорят, что TLS является протоколом сеансового или уровня представления (единственно верной позиции по этому поводу нет) в модели OSI/ISO. Таким образом, на практике может использоваться:

- и 4 уровня (TCP/IP),
- и 5 (TCP/IP с разделением уровня межсетевых интерфейсов на канальный и физический в соответствии с моделью OSI/ISO),
- и 6 уровней (SSL и TLS, занимающий место сеансового и уровня представления в модели OSI/ISO).

IP-адреса, маски, MAC-адреса

Более подробно данная тема раскрыта в отдельном курсе по сетям TCP/IP.

Упрощенно можно считать, что IPv4-адрес — это 4 байта (октета), записанные в десятичной системе. Например, 192.168.0.1 или 8.8.8.8.

Часть этих адресов используется в качестве адресов хостов, другие — в качестве адресов сетей, и еще часть в роли broadcast-адресов (широковещательных).

В классовой адресации первый адрес из диапазона будет являться адресом сети, а последний — broadcast-адресом. Маска позволяет определить по адресу, к какой сети он относится.

Например, если для адреса 192.168.1.1 указана маска 255.255.255.0, то это сеть 192.168.1.0, а широковещательный адрес — 192.168.1.255.

К слову, для адреса 192.168.1.1 может быть задана и маска 255.255.0.0. Тогда адресом сети будет уже 192.168.0.0, а бродкастом — 192.168.255.255. При этом адреса 192.168.1.0. и 192.168.1.255 могут быть адресами хостов, как и адрес 192.168.1.1. Существует еще более сложная система адресации — бесклассовая, она разбирается в курсе TCP/IP.

IP-адреса на нужной машине можно посмотреть, например, с помощью команды **ifconfig**:

```
user@ubuntu:~$ ifconfig
ens33     Link encap:Ethernet  HWaddr 00:0c:29:29:80:b7
          inet addr:192.168.1.2  Bcast:192.168.116.255  Mask:255.255.255.0
          inet6 addr: fe80::da89:c6b5:3098:4d72/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:127500 errors:0 dropped:0 overruns:0 frame:0
          TX packets:342871 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:147726823 (147.7 MB)  TX bytes:302244447 (302.2 MB)
          Interrupt:19 Base address:0x2000

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:1999 errors:0 dropped:0 overruns:0 frame:0
          TX packets:1999 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1
          RX bytes:665887 (665.8 KB)  TX bytes:665887 (665.8 KB)
```

Но в современных версиях ОС Linux ifconfig устарел, и лучше использовать команду **ip addr:**

```
user@ubuntu:~$ ip addr
9: enp0s3: <BROADCAST,MULTICAST,UP> mtu 1500 group default qlen 1
    link/ether 00:0c:29:29:80:b7
    inet 192.168.1.2/24 brd 192.168.116.255 scope global dynamic
       valid_lft forever preferred_lft forever
    inet6 fe80::da89:c6b5:3098:4d72/64 scope link dynamic
       valid_lft forever preferred_lft forever
1: lo: <LOOPBACK,UP> mtu 1500 group default qlen 1
    link/loopback 00:00:00:00:00:00
    inet 127.0.0.1/8 brd 127.255.255.255 scope global dynamic
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host dynamic
       valid_lft forever preferred_lft forever
```

Здесь видим два сетевых интерфейса: **enp0s3** — Ethernet, с MAC-адресом **00:0c:29:29:80:b7**, IP-адресом **192.168.1.2** и маской **255.255.255.0**. Так можем узнать, что наш хост принадлежит к сети **192.168.1.0**. Также есть статистика по сетевому интерфейсу, информация о локальном IPv6-адресе, MTU. Обратите внимание на MTU — это размер данных кадра (фактически IP-пакета), в который могут быть упакованы данные TCP-сегмента или UDP-дейтаграммы. Он не может быть превышен.

Также видим интерфейс локальной петли. Все адреса в сети **127.0.0.1** могут идентифицировать только данную машину и никогда не уходят в сеть. Любая посылка, отправленная на адрес **127.0.0.1**, не покинет компьютер (виртуальную машину). Такая адресация может быть использована для

взаимодействия между локальными приложениями (Nginx обращается к Apache2, который слушает TCP порт 80 по адресу 127.0.0.1, PHP обращается к MySQL, который слушает TCP-порт 3306 по адресу 127.0.0.1, и т. д.).

Неформально адреса делят на «белые», которые могут маршрутизироваться в интернет (например, 8.8.8.8, 5.255.255.55), и «серые», которые могут использоваться в локальных сетях, но не могут маршрутизироваться в интернет. Две разные локальные сети могут использовать одни и те же диапазоны «серых» адресов, так большинство DSL-модемов предоставляет по DHCP адреса в сети 192.168.1.1/255.255.255.0, а многие Wi-Fi-роутеры — в сети 192.168.0.1/255.255.255.0.

Но даже компьютеры с «серыми» адресами могут выходить в интернет, если используется:

- прокси-сервер — тогда будет доступен только интернет, так как все остальные TCP/IP службы будут ограничены локальной сетью. И даже если веб-адрес будет доступен по IP 5.255.255.55, команда ping 5.255.255.55 покажет, что связи нет;
- или NAT — точнее, NAPT, которая будет подменять «серый» адрес, предоставленный провайдером, на «белый» провайдера.

Команда **ping** позволяет проверить доступность узла:

```
user@ubuntu:~$ ping -c 4 5.255.255.55
PING 5.255.255.55 (5.255.255.55) 56(84) bytes of data.
64 bytes from 5.255.255.55: icmp_seq=1 ttl=128 time=41.1 ms
64 bytes from 5.255.255.55: icmp_seq=2 ttl=128 time=40.7 ms
64 bytes from 5.255.255.55: icmp_seq=3 ttl=128 time=41.3 ms
64 bytes from 5.255.255.55: icmp_seq=4 ttl=128 time=41.3 ms
--- 5.255.255.55 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3006ms
rtt min/avg/max/mdev = 40.798/41.168/41.375/0.222 ms
user@ubuntu:~$
```

Ключ **-c** указывает количество отправок. Если запустить без этого ключа, то будет осуществлять **ping**, пока вы не нажмете Ctrl-C.

ping с ключом **-b** позволяет сделать **ping** всех узлов в сети, используя broadcast-адрес:

```
# ping -b 192.168.1.255
```

Ответят все компьютеры, на которых не заблокирован ICMP-протокол или используемые ICMP UDP-порты.

Клиент-сервер и порты

По большей части в сетевых подключениях фигурирует клиент-серверное взаимодействие. Программа-клиент, используя стек TCP/IP, обращается к программе-демону и получает (а иногда отправляет) необходимый контент.

При этом серверы — это программы-демоны. Они постоянно находятся в памяти, их запуском управляет в нашем случае **systemd**. При запуске они занимают тот или иной TCP- или UDP-порт и слушают. В случае с UDP ожидают приема сообщений, с TCP-портом — ожидают приема сессии. Для этого сервер открывает соответствующие сокеты (STREAM для TCP или DGRAM для UDP) и слушает его.

Перечень прослушиваемых TCP-портов на машине можно посмотреть с помощью команды **netstat**:

```
$ netstat -tl
```

Просмотреть список прослушиваемых UDP-портов:

```
$ netstat -ul
```

Если хотим номера портов без расшифровки (только номер порта), добавляем **n**. Опции можно комбинировать.

```
$ netstat -ntul
```

Если хотим посмотреть список всех портов, добавляем **a**:

```
$ netstat -ntua
```

Просмотреть список прослушиваемых UNIX-сокетов:

```
$ netstat -xl
```

Но в современных версиях ОС Linux **netstat** считается устаревшей — лучше использовать команду **ss:**

```
$ ss -ntl
```

А чтобы увидеть имя процесса, нужно ее запустить через **sudo**:

```
# sudo ss -ntlp
```

Обратите внимание, что **netstat** и **ss** — это не команды для сканирования портов. Они лишь показывают информацию непосредственно от ядра операционной системы.

Для скана портов понадобится утилита **nmap**.Сначала ее надо установить:

```
# apt install nmap
```

Теперь можно посмотреть, какие порты открыты на каком-нибудь общедоступном сервере.

```
# nmap vk.com
```

Скорее всего, вы увидите 80 и 443 порты для веб-сервера. И это правильно, потому что лишний открытый порт — еще одна возможная проблема с безопасностью сервера. Особенно если это конфигурация по умолчанию.

Доменное имя преобразуется с помощью службы имен DNS. Это несложно посмотреть:

```
# dig vk.com
root@ubuntu:/etc/openvpn/ccd# dig vk.com
; <<>> DiG 9.10.3-P4-Ubuntu <<>> vk.com
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 39370
;; flags: qr rd ra; QUERY: 1, ANSWER: 2, AUTHORITY: 0, ADDITIONAL: 1
;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; MBZ: 0005 , udp: 512
;; QUESTION SECTION:
;vk.com.                                IN      A
;; ANSWER SECTION:
vk.com.                 5       IN      A       87.240.165.82
vk.com.                 5       IN      A       95.213.11.180
;; Query time: 43 msec
;; SERVER: 127.0.1.1#53(127.0.1.1)
;; WHEN: Fri Apr 14 23:59:46 MSK 2017
;; MSG SIZE  rcvd: 67
```

При каждом обращении к браузеру из URL берется следующая информация:

http://site.ru/path/file?param=something

Первая часть — это протокол:

- http — нешифрованый http, по умолчанию — порт 80;
- https — шифрованый http (http, упакованный в tls), порт 443;
- ftp — нешифрованый ftp, порт по умолчанию — 21.

Обычно других протоколов браузеры не знают.

Вторая часть — адрес сайта:

- доменное имя — преобразуется в IP-адрес. Именно на него будет установлена TCP-сессия;
- может быть сразу IP-адрес, например http://192.168.0.1/admin.

Третья часть — путь (как правило, путь ресурса):

- он будет передан на сервер. Это может быть действительно путь в файловой системе;
- может включать параметры, например **?param=something**, которые может обработать, например, **php**;
- может быть полностью произвольным: с помощью настроек Apache2 или Nginx переадресовываться скрипту (например, index.php), который уже сам разберет путь по собственным правилам.

Иногда (очень редко) используемый TCP-порт может быть указан явно.

Например: http://site.ru:8080/music.mp3 означает, что несмотря на использование http, порт будет подключен не 80, а 8080. Это может быть сервер радиостанции (например, icecast2). Соответственно, если TCP-сегменты приходят на порт 80, их получает приложение **apache2**, если на 8080 — то **icecast**.

Сами приложения используют динамические порты в верхнем диапазоне.

IANA зарезервировала для использования в качестве динамических порты 49152–65535.

На практике этот диапазон сильно разнится в зависимости от того, в какой операционной системе он используется. В Linux крайние значения диапазона можно посмотреть с помощью команды:

```
root@ubuntu:/etc/openvpn/ccd# cat /proc/sys/net/ipv4/ip_local_port_range
32768   60999
```

Отличие динамических портов от используемых серверами в том, что прослушиваемый порт может принимать много соединений. А браузер, подгружая несколько картинок одной страницы, как правило, использует несколько TCP-соединений, и на каждое будет отдельный динамический порт. Это легко посмотреть с помощью **tcpdump** или **wireshark**.

	Уровень	Адреса	Что адресуем
4	Прикладной уровень	URL, email-адреса (как правило, на основе доменных имен), длина переменная в символьном виде	пользователей, машины, ресурсы, приложения и т. д.
3	Транспортный уровень	TCP-порты и UDP-порты, длина — два байта	приложения (процессы)
2	Сетевой уровень	IP-адреса, для версии 4 — длина 4 байта	хосты
1	Уровень сетевых интерфейсов	MAC-адреса, длина 6 байт	сетевые интерфейсы в локальной сети

Минимум для настройки сети

Прежде чем перейти к командам, разберемся в теории работы сети. Это нужно, чтобы понять суть настройки локальной сети Ubuntu.

Компьютеры обмениваются информацией с помощью пакетов. Все данные в интернете передаются с помощью пакетов небольшого размера. Если не углубляться в подробности, то каждый пакет содержит адрес отправителя, адрес получателя и сами данные. Это привычные нам IP-адреса. Кроме IP, у компьютера есть физический адрес, который используется для общения между компьютерами в локальной сети. Это MAC-адрес, задается производителем сетевой карты.

Как только компьютер подключился к сети — независимо от того, проводное это соединение или беспроводное, — он может общаться только с компьютерами в локальной сети и только по физическим адресам. Чтобы получить доступ в глобальную сеть, машине в ней нужно получить IP-адрес. Для этого используется протокол DHCP. Если кратко: наш компьютер опрашивает все компьютеры в локальной сети о том, кто здесь DHCP-сервер. DHCP ему отвечает и выдает IP-адрес.

Таким же образом компьютер узнает IP маршрутизатора, через который он может получить доступ к сети, а затем пытается найти DNS-серверы или узнать стандартные у маршрутизатора. С теорией разобрались — перейдем к практике.

Для работы в TCP/IP в целом и в интернете в частности у машины должны быть следующие настройки:

- MAC-адрес у сетевой карты (присутствует по умолчанию);
- IP-адрес (даже «серый», если шлюз использует NAT);
- маска сети;
- маршрут по умолчанию (IP-адрес шлюза);
- IP-адреса одного или двух DNS-серверов для преобразования доменных имен в IP-адреса (часто DNS-сервером выступает шлюз, но могут быть указаны публичные кеширующие DNS-сервера от Google, 8.8.8.8 и 8.8.4.4).

Все настройки, кроме самого первого пункта, могут быть получены по DHCP при подключении компьютера к сети (или загрузке ОС, если к сети уже подключен).

В Ubuntu подключение к сети настраивается с помощью сервиса Network Manager в графическом интерфейсе. Чтобы подключиться к Wi-Fi-сети, достаточно пару раз кликнуть мышкой, выбрать соединение — и готово. То же самое и при использовании проводного соединения — тут интернет подключается автоматически, как только загрузился аплет.

Но не всегда доступен графический интерфейс: после неудачной установки драйвера или очередного обновления не запускается графическая оболочка, а на серверах она и вовсе не используется. В этой главе будет разобрана настройка сети Ubuntu из консоли. Мы поговорим о том, как настроить получение IP-адреса по DHCP, а также работу DNS. Рассмотрим ручную и автоматическую настройку, а также попробуем сделать это через системные конфигурационные файлы.

Автонастройка сети через DHCP

Рассмотрим автоматическую настройку сети для Ubuntu без Network Manager с помощью стандартных скриптов системы, которые остались от Upstart и пока все еще используются. Сначала определим, какие шаги нужны, чтобы все заработало:

1. Включаем сетевой интерфейс и подключаемся к сети;
2. Устанавливаем IP-адрес;
3. Получаем адреса DNS-серверов.

Готово — система все сделает сама. Нам нужно только выполнить нужные настройки. Но сначала посмотрим, какие сетевые интерфейсы подключены к системе. Команда **ip addr**:

```
user@ubuntu:~$ ip addr
9: enp0s3: <BROADCAST,MULTICAST,UP> mtu 1500 group default qlen 1
    link/ether 00:0c:29:29:80:b7
    inet 192.168.1.2/24 brd 192.168.116.255 scope global dynamic
       valid_lft forever preferred_lft forever
    inet6 fe80::da89:c6b5:3098:4d72/64 scope link dynamic
       valid_lft forever preferred_lft forever
1: lo: <LOOPBACK,UP> mtu 1500 group default qlen 1
    link/loopback 00:00:00:00:00:00
    inet 127.0.0.1/8 brd 127.255.255.255 scope global dynamic
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host dynamic
       valid_lft forever preferred_lft forever
```

В нашей системе только один интерфейс — это **enp0s3**, есть еще **lo**, но он виртуальный и указывает на эту машину.

Настройки сети находятся в файле **/etc/network/interfaces**:

```
user@ubuntu:~$ cat /etc/network/interfaces

# ifupdown has been replaced by netplan(5) on this system.  See
# /etc/netplan for current configuration.
# To re-enable ifupdown on this system, you can run:
#    sudo apt install ifupdown
```

Так что можем активировать традиционную настройку сети через:

```
root@ubuntu:~# sudo apt install ifupdown
```

И затем отредактируем наш конфиг — нас будут интересовать в этом файле строки **auto** и **iface**:

```
auto enp0s3
iface enp0s3 inet dhcp
```

Первая указывает, что нужно активировать интерфейс при загрузке. А вторая определяет, что настройки самого интерфейса нужно получить по DHCP.

После завершения настройки сохраните файл и перезапустите сетевой сервис:

```
root@ubuntu:~# sudo service networking restart
```

Если сетевой кабель подключен и вы все сделали правильно, то сеть будет работать.

Автонастройка статического адреса

При настройке статического IP-адреса компьютер не будет связываться с DHCP-сервером, поэтому здесь придется указать намного больше параметров.

Содержимое нашего конфигурационного файла будет выглядеть так:

```
auto eth0
iface eth0 inet static
address 192.168.1.7
gateway 192.168.1.1
netmask 255.255.255.0
network 192.168.1.0
broadcast 192.168.1.255
```

С первыми двумя строчками все понятно, а следующие задают параметры настройки интерфейса:

- **address** — наш IP-адрес;
- **gateway** — шлюз, через который будем получать доступ в интернет;
- **netmask** — маска сети;
- **network** — адрес сети (тот же адрес, что и шлюз, только с нулем вместо единицы);
- **broadcast** — широковещательный адрес сети, отправленный на него пакет придет всем компьютерам локальной сети.

Как видите, **network** и **broadcast** — это первый и последний IP-адреса сети. Теперь сохраните файл и перезапустите сеть:

```
root@ubuntu:~# sudo service networking restart
```

Если все параметры были указаны правильно, все будет работать. Но если допущена хоть одна ошибка, доступ к сети вы не получите.

Это была автоматическая настройка локальной сети Ubuntu. Еще рассмотрим, как все сделать вручную, без конфигурационных файлов.

Ручная настройка сети в Ubuntu

Включаем интерфейс:

```
root@ubuntu:~# sudo ip link set enp0s3 up
```

Устанавливаем IP-адрес, маску сети и broadcast-адрес для нашего интерфейса:

```
root@ubuntu:~# sudo ip addr add 192.168.1.2/255.255.255.0 broadcast
192.168.1.255 dev enp0s3
```

И указываем IP-адрес шлюза:

```
root@ubuntu:~# sudo ip route add default via 192.168.1.1
```

Здесь 192.168.1.2 — наш IP-адрес, 255.255.255.0 — маска сети, 192.168.1.255 — широковещательный адрес. Замените эти значения на свои, а 192.168.1.1 — это адрес нашего маршрутизатора (как правило, это адрес домашнего роутера. Подробнее о маршрутизации — в дополнительной части этого пособия).

Как видите, сеть работает.

Если хотите поэкспериментировать на машине с рабочей сетью, ее можно сбросить командой:

```
root@ubuntu:~# sudo ip -4 addr flush dev enp0s3
```

Ручная настройка DNS

Служба DNS используется для преобразования доменных имен сайтов в IP-адреса. При получении IP-адреса автоматически через DHCP мы используем правильные DNS-серверы. Но если выбрали статический IP, то DNS можно и не получить, поэтому придется сделать все вручную.

Если вам нужно настроить DNS так, чтобы он не сбивался после перезагрузки, необходимо использовать систему настройки сети Ubuntu. Для этого откройте файл **/etc/network/interfaces** и добавьте в него строчку после директив для нужного интерфейса:

```
dns-nameservers 8.8.8.8 4.4.4.4
```

Здесь **8.8.8.8** и **4.4.4.4** — это IP-адреса DNS-серверов. Можете заменить их на свои. И можно использовать один, а не два. Дальше сохраните файл и перезапустите сеть:

```
root@ubuntu:~# sudo service networking restart
```

Если же вам нужно настроить DNS только для этого сеанса, то добавьте строчку в **/etc/resolv.conf**:

```
root@ubuntu:~# sudo vi /etc/resolv.conf

nameserver 8.8.8.8
nameserver 4.4.4.4
```

После сохранения файла сеть будет работать полностью так, как нужно. Но последний способ пригоден только до перезагрузки, поскольку файл **/etc/resolv.conf** генерируется автоматически.

Сетевой фильтр

Сетевой фильтр служит для защиты вашей машины от внешних атак. То есть, если вы используете только TCP-порты 80 и 443 для веб-сервера, доступны извне должны быть только они. Также если у

вас используется 22 TCP-порт для ssh, то лучше перенести этот порт на любой другой, неизвестный злоумышленникам (например, на 49123 или 61666 — решать вам).

Кроме того, с помощью сетевого фильтра решаются задачи NAT, то есть:

- выход в интернет машин внутренней сети,
- предоставление доступа через NAT в интернет для виртуальных машин и контейнеров,
- а также DNAT (Destination NAT) — проброс портов в соответствующие Docker или LXC-контейнеры.

Рассмотрим подробнее команды **iptables** — это пользовательская утилита, задающая правила для сетевого фильтра **Netfilter**, работающего уже в пространстве ядра Linux.

Netfilter

Стек TCP/IP по большей части реализован в пространстве ядра. Если использовать Linux-машину в качестве шлюза, транзитные (forward) пакеты будут перенаправляться из одного сетевого интерфейса в другой, и пользовательское пространство в этом процессе задействовано не будет.

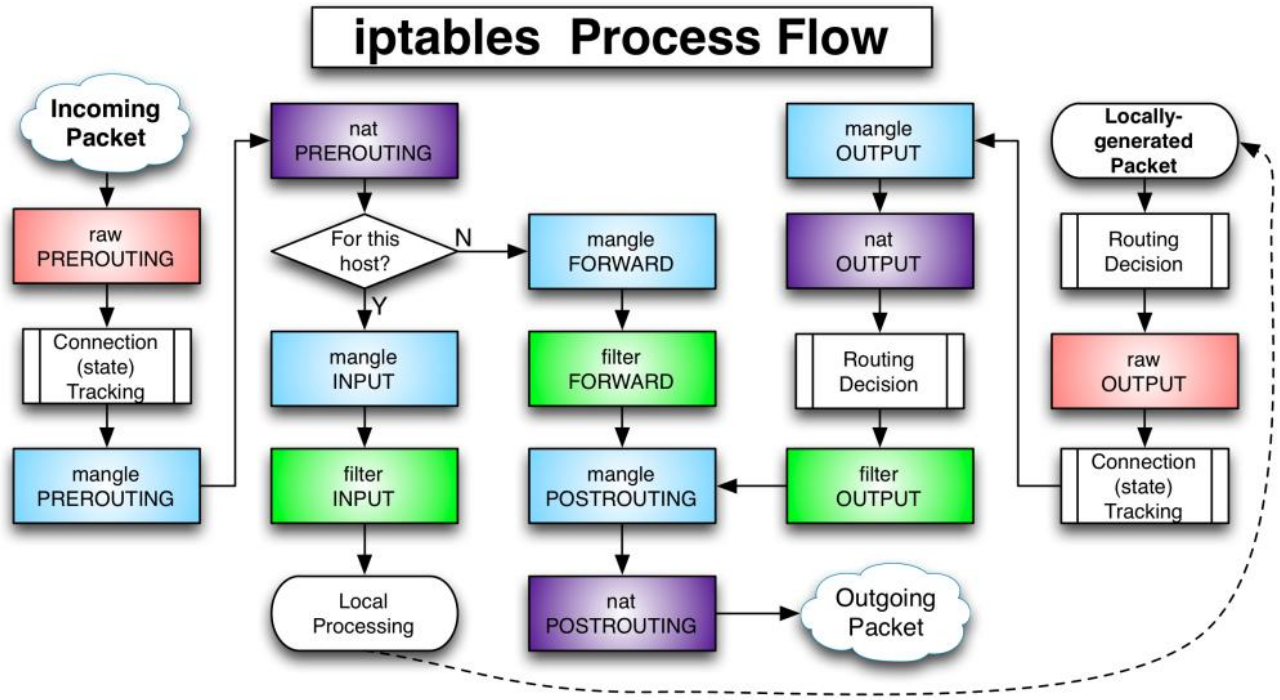
Более того, доставка сообщения соответствующему приложению данной машины проходит ряд этапов на канальном, сетевом и транспортном уровне, и все это осуществляется на уровне ядра.

Сетевой интерфейс, принимая кадр, определяет, надо ли его отбросить (отключен promisc-режим, MAC-адреса назначения и сетевой карты не совпадают) или принять (MAC-адрес сетевой карты совпадает или включен promisc-режим). На практике promisc-режим нужен, чтобы проанализировать чужой трафик через **tcpdump** или **WireShark**.

На сетевом уровне анализируется IP-адрес назначения (наложением IP-масок и сравнением с IP-адресами сетей либо прямым указанием IP-хостов). В зависимости от того, какой сети он принадлежит, определяется, надо ли отправить пакет в другой сетевой интерфейс, в адрес шлюза или передать локально. В последнем случае он передается соответствующему приложению.

Обрабатывается транспортный уровень. Если это UDP, он передается приложению, слушающему UDP-порт или отправившему с него сообщение и ожидающему ответа. Если это TCP-порт, ядро обрабатывает механизмы установки соединения, контроля целостности, сборки TCP-сегментов в единый поток данных / файл. Данные уже побайтово передаются в соответствующее приложение, которое либо слушает TCP-порт, либо установило TCP-соединение.

На каждом из этих этапов могут производиться действия как по фильтрации, так и по преобразованию пакетов / дейтаграмм / TCP-сегментов.



Iptables

Iptables — утилита для работы с правилами, которые использует сетевой фильтр **netfilter**.

Netfilter — компонент ядра, а **iptables** — утилита, работающая в пользовательском пространстве.

Формат запуска:

```
iptables [-t таблица] команда [критерий] [ -j цель]
```

Команды:

- **-L** — посмотреть список правил (можно указать цепочку, иначе выводятся все);
- **-F** — удалить все правила (можно указать цепочку, иначе очистятся все);
- **-A** — добавить правило (критерий и цель) в заданную цепочку;
- **-D** — удалить правило (критерий и цель, либо порядковый номер в заданной цепочке);
- **-P** — установить политику по умолчанию для заданной цепочки;
- **-I** — вставка нового правила в цепочку;
- **-N** — создать новую цепочку с заданным именем в таблице;
- **-X** — удалить заданную цепочку (но сначала нужно удалить в ней правила, и на нее не должны ссылаться правила из других цепочек);

Другие команды посмотрите самостоятельно. Пример:

```
root@ubuntu:~# iptables -L
```

```
Chain INPUT (policy ACCEPT)
target     prot opt source               destination
Chain FORWARD (policy ACCEPT)
target     prot opt source               destination
Chain OUTPUT (policy ACCEPT)
target     prot opt source               destination
root@ubuntu:~#
```

Видим цепочки в выводе команды. Вспомним про этапы обработки пакетов на уровне ядра. Обратите внимание, что политика по умолчанию — **ACCEPT**, то есть все пакеты по умолчанию пропускаются.

Существуют и другие политики (например, **DROP**):

```
root@ubuntu:~# iptables -P INPUT DROP
```

Она разорвет ваше соединение через **ssh** (не стоит так делать на настоящих машинах). Если это произошло на виртуальной машине, осталось только зайти в нее и вернуть политику по умолчанию:

```
root@ubuntu:~# iptables -P INPUT ACCEPT
```

Политика по умолчанию задает порядок работы с пакетами, для которых нет ни одного правила. Если в цепочке правил ни одно не подошло, в итоге применяется действие по умолчанию, которое и задается политикой. Если правило подошло, действие по умолчанию не выполняется.

Цепочка — это набор правил, которые проверяются по пакетам поочередно (напоминает язык программирования). В таблице **filter** видим цепочки **INPUT**, **FORWARD** и **OUTPUT**. Но есть и другие таблицы (их нужно указывать явно): таблица **nat**, когда нам необходима трансляция адресов и портов, и **mangle** — когда требуется внести изменения в пакет (например, установить **ttl** в 64 и скрыть от провайдера использование подключения как шлюза к собственной сети).

```
root@ubuntu:~# iptables -t nat -L
Chain PREROUTING (policy ACCEPT)
target     prot opt source               destination
Chain INPUT (policy ACCEPT)
target     prot opt source               destination
Chain OUTPUT (policy ACCEPT)
target     prot opt source               destination
Chain POSTROUTING (policy ACCEPT)
target     prot opt source               destination
```

Для критериев можно использовать следующие параметры:

- **-p** — тип протокола: **tcp**,**udp**, **icmp** (полный список здесь: **cat /etc/protocols**);
- **-s** — IP-адрес отправителя (можно использовать адреса сетей с маской 10.0.0.0/255.0.0.0 или 10.0.0.0/8). Можно применять логическое отрицание, поставив символ **!** перед адресом;
- **-d** — IP-адрес получателя (синтаксис аналогичный);
- **-i** — сетевой интерфейс, с которого получен пакет (по умолчанию пакеты обрабатываются независимо от того, с какого интерфейса они пришли);

- **-o** — сетевой интерфейс, в который предполагается отправка пакета. Может применяться только в цепочках **OUTPUT**, **FORWARD** и **POSTROUTING**;
- **-m** — лимит, устанавливает предельное число пакетов в единицу времени, которое способно пропустить правило;
- **--sport** — порт (или диапазон портов) отправителя;
- **--dport** — порт назначения (куда предназначен пакет).

Остальные параметры изучите самостоятельно.

В качестве цели можно использовать следующий действия:

- **ACCEPT** — принять пакет;
- **REJECT** — отбросить пакет, но выдать сообщение об ошибке на хост отправителя;
- **DROP** — отбросить пакет, не уведомляя отправителя;
- **SNAT** (для таблицы NAT) — преобразовать IP-адрес отправителя в заголовке пакета, подменив адрес, указанный в критерии, на свой «белый» IP-адрес, указанный после SNAT (**--to-source=...**);
- **DNAT** (для таблицы NAT) — **destination NAT** (когда надо перенаправить пакеты с одной машины на несколько разных в зависимости от критериев);
- **MASQUERADE** (для таблицы NAT) — преобразовать IP-адрес отправителя в заголовке пакета, подменив адрес, указанный в критерии, на свой (например, если он динамический);
- **LOG** — журналировать пакеты и события;
- **MARK** (для таблицы mangle) — пометить пакет;
- **TTL** (для таблицы mangle) — изменить TTL (например, установить его снова в 64, скрыв движения по локальной сети от провайдера).

Другие действия изучите самостоятельно.

Сохранение значений

Обратите внимание, что все значения, вводимые через **iptables**, не будут сохранены при перезагрузке. Для их сохранения можно воспользоваться двумя способами:

- установить демон **iptables-persistent**, который будет сохранять введенные значения;
- использовать утилиты **iptables-save** и **iptables-restore**. Но вызывать их нужно самостоятельно и при сохранении, и при старте.

По умолчанию **iptables-save** выводит список правил в **stdout**, а **iptables-restore** читает из **stdin**. Поэтому надо перенаправить в файл:

```
# iptables-save > /etc/iptables.rules
```

Чтобы восстановить, нужно вызвать **iptables-restore**:

```
# iptables-restore < /etc/iptables.rules
```

Поднятые сетевые интерфейсы, маршруты тоже не сохранятся. Для сохранения в Ubuntu используйте файл **/etc/network/interfaces** либо **iptables-persistent**, или можно добавить **iptables-restore < /etc/iptables.rules** в **/etc/rc.local**.

Пример настройки **interfaces** — в **/usr/share/doc/ifupdown/examples/network-interface**.

Так можно настроить статический сетевой интерфейс **/etc/network/interfaces**:

```
iface ens33 inet static
     address 192.168.200.2
     network 192.168.200.0
     netmask 255.255.255.0
     broadcast 192.168.200.255
     dns-nameservers 192.168.200.1 8.8.8.8
     up route add -net 192.168.220.0 netmask 255.255.255.0 gw 192.168.200.220
     up route add default gw 192.168.200.1
     down route del default gw 192.168.200.1
     down route del -net 192.168.220.0 netmask 255.255.255.0 gw 192.168.200.220
     pre-up iptables-restore < /etc/iptables.rules
     post-down iptables-restore < /etc/iptables.downrules
```

Или так — для динамического интерфейса:

```
auto ens33
iface ens33 inet dhcp
  pre-up iptables-restore < /etc/iptables.rules
  post-down iptables-restore < /etc/iptables.downrules
```

Файл **/etc/reslov.conf** в **/etc/resolv.conf.d** использовать не следует. Настройки DNS теперь находятся в вышеописанном файле. Обратите внимание, что в RedHat-подобных системах (**CentOS**) сетевые настройки хранятся совсем в других файлах и форматах.

Итоги

В этом уроке мы рассмотрели автонастройку сети Ubuntu с помощью DHCP и ручную — из консоли. Теперь вы знаете, что нужно делать, если у вас нет доступа к графическому интерфейсу, но срочно нужно попасть в сеть.

Также рассмотрели вопросы защиты с помощью встроенного файервола Linux — **iptables**.

Если остались вопросы, посмотрите дополнительные материалы и ссылки на статьи в конце методички.

Практическое задание

1. Произвести ручную настройку сети в Ubuntu, на каждом шаге сделать скриншоты.
2. Переключить настройку сети на автоматическую через DHCP, проверить получение адреса.
3. Изменить адрес DNS на 1.1.1.1 и проверить доступность интернета, например, открыв любой браузер на адрес https://geekbrains.ru.
4. * Настроить правила iptables, чтобы из внешней сети можно было обратиться только к портам 80 и 443. Запросы на порт 8080 перенаправлять на порт 80.
5. * Дополнительно к предыдущему заданию настроить доступ по ssh только из указанной сети.
6. * Настроить OpenVPN, связать несколько виртуальных машин с помощью OpenVPN-туннеля.
7. * Сделать одну из настроенных в задании выше машин шлюзом доступа в интернет. Настроить NAT.

Примечание. *Задания с 4–7 даны для тех, кому упражнений 1–3 показалось недостаточно.*

Используемая литература

Для подготовки данного методического пособия были использованы следующие ресурсы:

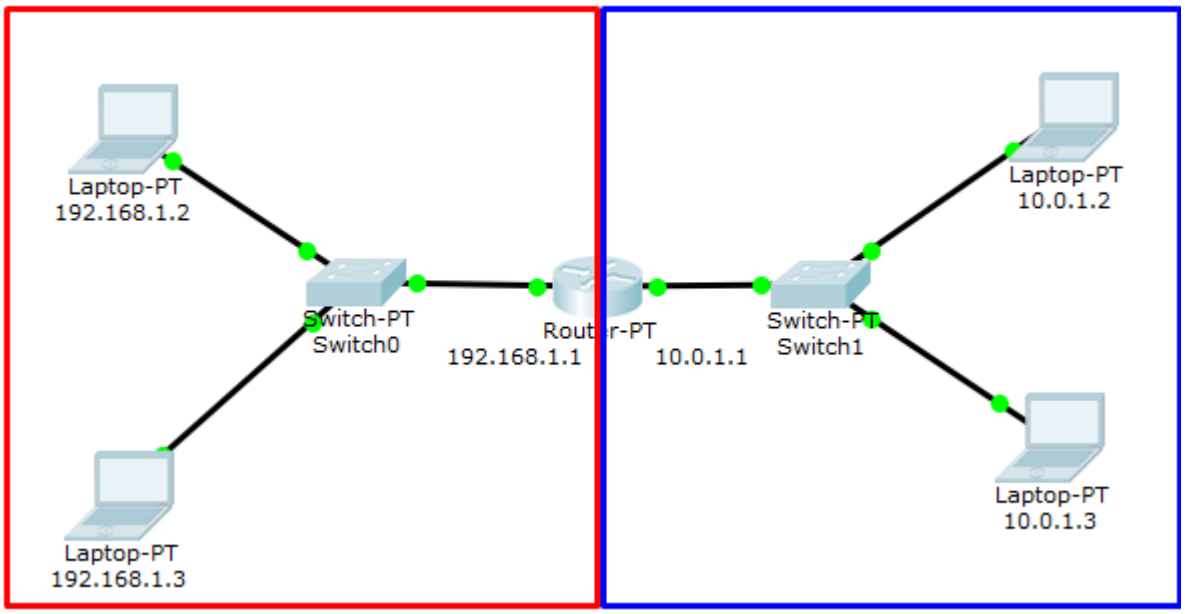
1. https://ru.wikipedia.org/w/index.php?title=Сетевая_модель_OSI&oldid=83830272
2. https://ru.wikipedia.org/wiki/Динамический_порт
3. https://losst.ru/nastrojka-seti-iz-konsoli-ubuntu
4. http://www.resoo.org/docs/iptables_ipchains/iptables-help.pdf
5. https://www.cyberciti.biz/faq/setting-up-an-network-interfaces-file/
6. https://help.ubuntu.com/community/IptablesHowTo

Приложение 1. Маршрутизация

Напрямую связь между хостами, адресуемыми IP-адресами, может быть только в рамках одной сети. Если адрес хоста 192.168.0.1, невозможно отправить сообщение на адрес 10.0.0.1. Для этих целей применяется маршрутизация. Она бывает статическая (задаваемая с помощью правил **route**) и динамическая (с помощью алгоритмов маршрутизации, таких как BGP, RIP, OSPF и т. д.)

У каждого компьютера есть таблица маршрутизации. Она показывает:

- в какой сетевой интерфейс следует отправлять пакет с тем или иным IP-адресом, если пакет предназначен получателю, доступному через канальный уровень напрямую (в той же физической среде);
- или на какой шлюз (который, сам находясь в этой сети, будет перенаправлять пакеты в другую сеть, к которой он также подключен).



Если на компьютере 192.168.1.2 посмотрим таблицу маршрутизации, увидим:

```
user@ubuntu:~$ route
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
default         192.168.1.1     0.0.0.0         UG    100    0        0 ens33
192.168.1.0     *               255.255.255.0   U     100    0        0 ens33
```

Но это если **Router** имеет и подключение к внешней сети. Может быть и такая картина:

```
user@ubuntu:~$ route
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
10.0.1.0        192.168.1.1     255.255.255.0   UG    100    0        0 ens33
192.168.1.0     *               255.255.255.0   U     100    0        0 ens33
```

Таблица маршрутизации есть на каждой машине с поддержкой стека TCP/IP, даже на Windows-машинах. Когда операционная система получает к отправке пакет, она по IP-адресу и маске сравнивает, в какой сетевой интерфейс и какой шлюз его отправить. В первом случае у нас было два маршрута:

- один — для устройств, подключенных к той же сети, 192.168.1.0, в сетевой интерфейс **ens33**;
- другой маршрут по умолчанию — на шлюз 192.168.1.1, подключенный (обратите внимание) к той же сети. Пакеты на все другие IP-адресы будут отправляться туда.

Во втором случае тоже два маршрута:

- пакеты на адреса в сети 192.168.1.0 отправляются в сетевой интерфейс ens33, так как доступны через канальный уровень;
- Пакеты на адреса в сети 10.0.1.0 отправляются на адрес шлюза с адресом 192.168.1.0 в этой же сети;

Остальные пакеты в данной конфигурации будут отброшены с сообщением **No Route to host**.

Чтобы заработало, необходимо включить **IP-forwarding**:

```
user@router:~$ echo 1 >/etc/proc/sys/network/ipv4/ip_forward
```

Сложнее настройка статической маршрутизации между двумя маршрутизаторами.

Предположим, у нас уже есть два маршрутизатора (в данном случае в их роли выступают Linux-машины).

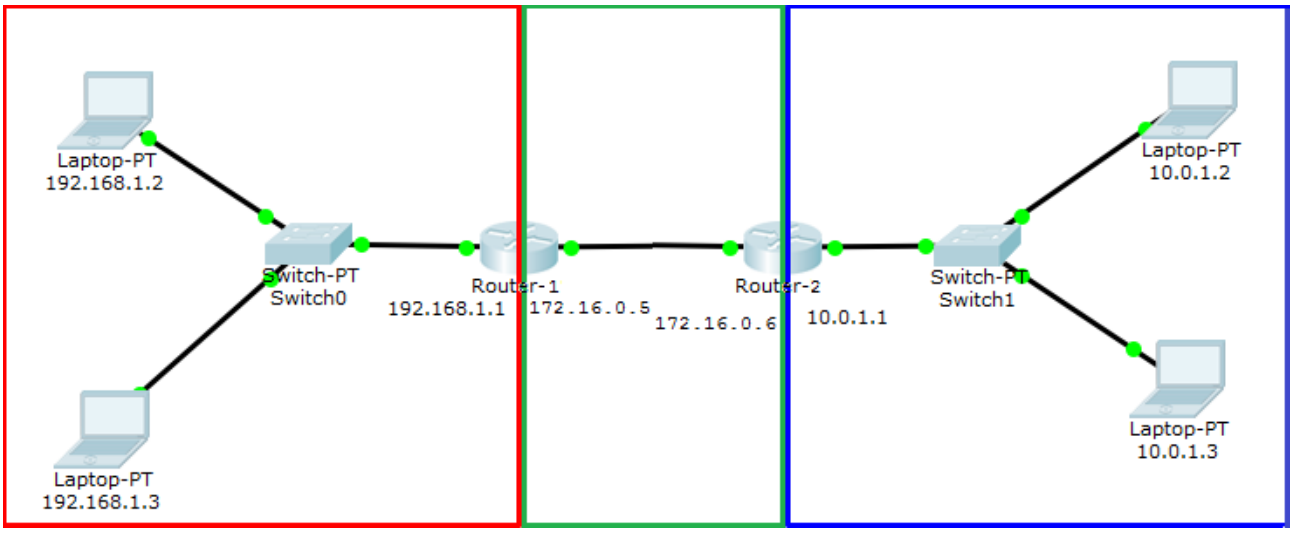
У каждой из машин есть по два сетевых интерфейса. У клиентских машин ближняя машина-маршрутизатор указана в маршруте по умолчанию.

Но при этом машина 192.168.1.1, получая сообщение от, например, 192.168.1.2, ничего не знает о сети 10.0.1.0. В прошлом случае знала, так как подобный маршрут возникает уже при поднятии соответствующего сетевого интерфейса.

Вы в этом легко можете убедиться, подняв нужный сетевой интерфейс (настоящий или, как в примере, псевдоним) и посмотрев таблицу маршрутов:

```
# ifconfig ens33:dummy 172.30.1.1 netmask 255.255.255.0 up
# route
# ifconfig ens33:dummy down
```

Оба маршрутизатора должны быть настроены на пересылку пакетов (IP Forwarding необходимо разрешить на обеих машинах).



Но также надо указать соответствующие маршруты и на маршрутизаторах.

На **Router-1**:

```
# route add -net 10.0.1.0 netmask 255.255.255.0 gw 172.16.0.6
```

Мы указали, что маршрут в сеть 10.0.1.0 знает машина 172.16.0.6. Обратите внимание, мы указываем адрес в нашей сети.

На **Router-2**:

```
# route add -net 192.168.1.0 netmask 255.255.255.0 gw 172.16.0.5
```

После этого машины из сетей 192.168.1.0 и 10.0.1.0 смогут работать друг с другом.

Если же между Router-1 и Router-2 нет прямой связи, а мы хотим адресовать частные адреса, то уже понадобится использовать тоннель, например с помощью OpenVPN.

Приложение 2. Туннели

Задача — связать две сети предприятия, находящиеся в разных городах. В рамках клиент-серверного взаимодействия шлюз одной сети (VPN-клиент) будет подключаться к шлюзу другой (VPN-сервер). Транспортировка будет осуществляться через внешнюю сеть, для чего нам необходимо выполнить три задачи:

- обеспечить аутентификацию, чтобы удостовериться, что VPN-клиент — действительно тот, за кого себя выдает;
- обеспечить шифрование трафика, чтобы никто из злоумышленников в интернете не мог его перехватить;
- обеспечить туннелирование — следование IP-пакетов, упакованных в конечном итоге в другие пакеты. Таким образом все хопы (прыжки), которые делаются при не туннельном соединении, в данном случае будут не видны, и между шлюзами по туннелю будет всего лишь один

прыжок. Это позволит настроить маршрут через туннель и действительно объединить две физически непосредственно не связанные между собой сети в единую безопасную сеть.

Сети, построенные таким образом, называют виртуальными частными сетями (VPN — Virtual Private Networks).

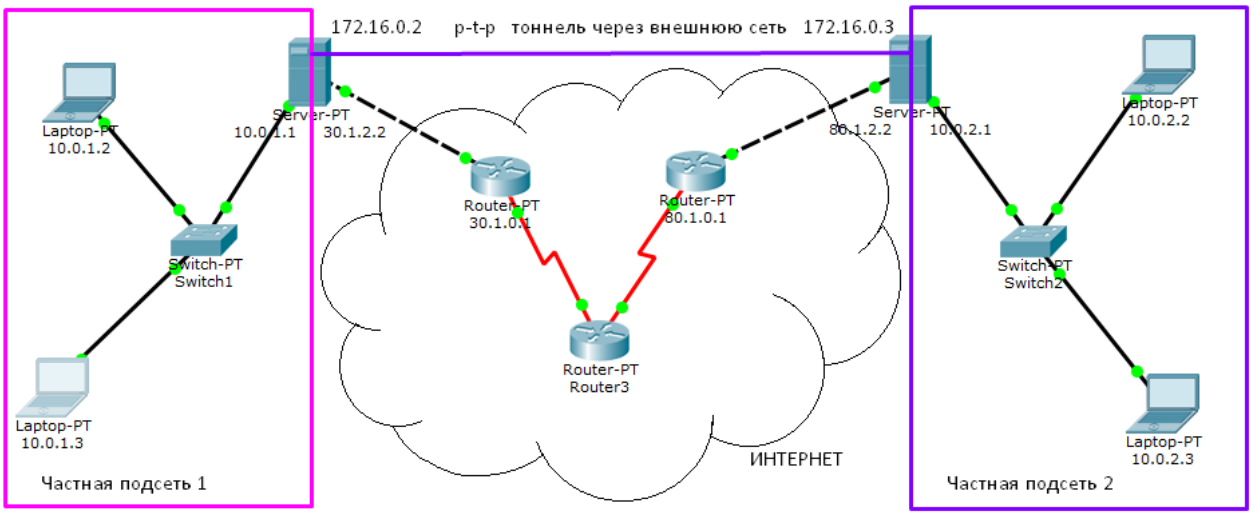
Существуют разные туннелирующие протоколы. Например, протокол PPPoE является туннелирующим, но работает на канальном уровне. С его помощью можно соединить клиент и сервер (как правило, шлюз в интернете), но только если машины присутствуют в одной сети. Можно настроить и в виртуальной машине, но тип сетевого соединения должен быть «мост» (bridge). Главная причина использования PPPoE провайдерами — в возможности аутентификации и логирования действий пользователя.

PPTP формально является защищенным туннелирующим протоколом. Но на практике он не криптостойкий. Используется там, где безопасность не требуется. В отличие от PPPOE, протокол PPTP работает поверх сетевого уровня и не налагает требование находиться в одной физической сети. Поэтому он используется провайдерами для предоставления «белого» IP-адреса клиенту. Дело в том, что обычно инфраструктура провайдера состоит из «серых» IP-адресов — отсюда потребность в туннелировании. В остальном PPTP предоставляет аналогичные PPPoE возможности — аутентифицировать и логировать действия пользователя. Также PPPoE используется в DSL-модемах.

Оба вышеуказанных протокола поддерживаются в Windows, но их можно настроить и в Linux.

Одной из полноценных реализаций защищенного VPN является OpenVPN. Он позволяет надежно передавать данные, присоединять отдельных удаленных пользователей к офисным сетям и связывать географически удаленные сети предприятия в одну защищенную сеть. Для доступа к OpenVPN в Windows потребуется установка дополнительного программного обеспечения.

На картинке видна схема объединения двух офисных сетей в одну частную сеть.



Подобное объединение может быть выполнено с помощью OpenVPN или PPTP (но невозможно с помощью PPPoE). С точки зрения логической инфраструктуры (в этом несложно убедиться с помощью команды **traceroute**) для абонентов частной подсети 1 и частной подсети 2 соединение выглядит так, словно шлюзы напрямую соединены кабелем. На самом деле кабель виртуальный (соединение «точка — точка», туннель), и один прыжок туннелированного пакета обходится несколькими прыжкам несущего его пакета. PPTP и OpenVPN предоставляют аналогичные возможности, но с точки зрения безопасности PPTP не устойчив к взлому.

OpenVPN

Предположим, у нас есть две машины, играющие роль шлюза. На них может быть установлен DHCP-сервер, кеширующий DNS-сервер, настроен сетевой фильтр. В любом случае остальные машины локальной сети выходят в интернет через него.

Наша задача — связать две сети так, чтобы можно было работать в безопасной среде, объединяющей обе подсети предприятия.

Напрямую прописать маршрут в сеть 10.0.2.0/255.255.255.0 через 80.1.2.2 мы не можем, потому что:

- небезопасно внутренний трафик передавать через внешнюю сеть;
- адреса вида 10.0.2.0 и 10.0.1.0 не маршрутизируется в интернете и будут отброшены сетевыми маршрутизаторами;
- третья причина еще более веская. Мы не можем назначить шлюзом адрес машины, недоступной на канальном уровне.

Поэтому нам понадобится туннелирование. Туннелирование построено по клиент-серверному взаимодействию. VPN-сервер слушает указанный порт, например UDP-порт 6666, а клиент подключается к этому порту. В случае разрыва соединения переподключаться будет именно клиент. Но после того как соединение установлено, обе сети будут равноправными (по крайней мере можно настроить, чтобы связанные подсети маршрутизировались в обе стороны. Не всегда это нужно: например, когда задача — всего лишь подключить к частной сети компьютер удаленного сотрудника).

Предположим, что компьютер с адресами 10.0.1.1 (в локальную сеть) и 30.1.2.2 (во внешнюю) будет клиентом. Компьютер с адресами 10.0.2.1 (во внутреннюю сеть) и 80.1.2.2 (во внешнюю) — сервером.

Итак, подключаемся по ssh на компьютер 80.1.2.2 (VPN-сервер).

Устанавливаем **openvpn**:

```
# sudo apt-get install openvpn
```

Создаем сертификаты: воспользуемся программой **easy-rsa** для генерации сертификатов. Ранее она входила в состав **openssh**, а сейчас это отдельный проект — скачаем его дополнительно.

Создадим директорию для ключей и перейдем в нее:

```
# mkdir /etc/openvpn/keys
# cd /etc/openvpn/keys
```

Скачиваем и распаковываем утилиту **easy-rsa**:

```
# wget https://github.com/OpenVPN/easy-rsa/archive/master.zip
# unzip master.zip
```

Создаем структуру публичных PKI-ключей:

```
# cd /etc/openvpn/keys/easy-rsa-master/easyrsa3
```

```
# mv vars.example vars
# ./easyrsa init-pki
```

Будет создана папка **/etc/openvpn/keys/easy-rsa-master/easyrsa3/pki**

Создаем удостоверяющий центр CA:

```
# ./easyrsa build-ca
```

Вводим и запоминаем пароль для центра сертификации, он нам понадобится. Получим два ключа:

- секретный (приватный), который будет храниться на сервере, и не должен его никогда покидать: **/etc/openvpn/keys/easy-rsa-master/easyrsa3/pki/private/ca.key**
- и открытый, его вместе с пользовательскими сертификатами будем передавать клиентам: **/etc/openvpn/keys/easy-rsa-master/easyrsa3/pki/ca.crt**

Создаем запрос сертификата для сервера без пароля с помощью опции **nopass**:

```
# ./easyrsa gen-req server nopass
```

Подписываем запрос на получение сертификата у нашего CA:

```
# ./easyrsa sign-req server server
```

Вводим пароль от CA, который указывали раньше, и отвечаем на вопрос **yes**.

Результат — подписанный удостоверяющим центром сертификат для сервера — **/etc/openvpn/keys/easy-rsa-master/easyrsa3/pki/issued/server.crt**

Также понадобится файл с параметрами для работы алгоритма Диффи — Хеллмана (протокол Диффи — Хеллмана позволяет получить общий секретный симметричный ключ при работе через незащищенный от прослушивания канал). Создаем его (потребует некоторого времени):

```
# ./easyrsa gen-dh
```

Получаем файл dh-сертификата — **/etc/openvpn/keys/easy-rsa-master/easyrsa3/pki/dh.pem**.

Копируем в папку **/etc/openvpn** все необходимые для работы OpenVPN-сервера ключи:

```
# cp pki/ca.crt /etc/openvpn/ca.crt
# cp pki/dh.pem /etc/openvpn/dh.pem
# cp pki/issued/server.crt /etc/openvpn/server.crt
# cp pki/private/server.key /etc/openvpn/server.key
```

Создадим ключи для клиента:

```
# ./easyrsa gen-req client nopass
# ./easyrsa sign-req client client
```

Будет запрошен **Common name** клиента. Рекомендуется указать имя хоста клиента. Это понадобится в настройках. В результате получаем:

/etc/openvpn/keys/easy-rsa-master/easyrsa3/pki/issued/client.crt

/etc/openvpn/keys/easy-rsa-master/easyrsa3/pki/private/client.key

Далее создаем папку, например:

```
# mkdir /etc/openvpn/clientfiles
```

Копируем туда оба клиентских файла и **ca.crt**:

```
# cp pki/ca.crt /etc/openvpn/clientfiles
# cp pki/issued/client.crt /etc/openvpn/clientfiles
# cp pki/private/client.key /etc/openvpn/clientfiles
```

Запаковываем в архив:

```
# cd /etc/openvpn/
# tar -cf clientfiles.tar clientfiles
```

С помощью **scp** (а на наших машинах, вы помните, установлен **ssh**) отправляем на клиентскую машину:

```
# scp clientfiles.tar user@30.1.2.2:/home/user
```

Настроим конфигурационный файл для сервера:

```
# mcedit /etc/openvpn/server.conf
```

Практически создаем с нуля:

```
port 6666 # 6666 — если предпочитаете использовать нестандартные порты для работы
                # либо используйте стандартный 1194
proto udp # протокол может быть и tcp, если есть необходимость в этом (слишком много потерь)
dev tun     # tun работает на сетевом уровне, tap на канальном. Нам достаточно tun
                # файлы сертификатов и ключей
ca /etc/openvpn/ca.crt
cert /etc/openvpn/server.crt
key /etc/openvpn/server.key
dh /etc/openvpn/dh.pem
server 172.16.0.0 255.255.255.0 # подсеть для туннеля, может быть любой
route 10.0.2.0 255.255.255.0           # указываем подсеть, к которой будем обращаться через vpn,
#это клиентская подсеть
#Следующие три опции можно пропустить, если мы создаем удаленное рабочее место.
#Без них только клиент будет иметь доступ к серверным ресурсам
#Для соединения офисов — они понадобятся
ifconfig-pool-persist ipp.txt            # файл с записями соответствий client - ip
client-to-client                                    # позволяет клиентам OpenVPN подключаться друг к другу
client-config-dir /etc/openvpn/ccd # директория с индивидуальными настройками клиентов
keepalive 10 120
comp-lzo
persist-key
persist-tun
status /var/log/openvpn/openvpn-status.log
log /var/log/openvpn/openvpn.log
verb 3
```

Создаем необходимые директории:

```
# mkdir /etc/openvpn/ccd && mkdir /var/log/openvpn
```

Создаем файл конфигурации клиента в директории, указанной в параметре **client-config-dir** — с учетом того, что было сказано выше. Эти настройки нужны, чтобы связать две подсети, т. е. чтобы клиентская сеть тоже маршрутизировалась со стороны сервера:

```
# mcedit /etc/openvpn/ccd/client
```

Обратите внимание, что **client** — имя сервера (**common name**). В идеале оно должно совпадать с доменным именем, прописанным в том числе и при генерации клиентского сертификата. Если же указали в качестве **common name** IP-адрес, придется использовать его.

```
iroute 10.0.2.0 255.255.255.0
```

Запускаем сервер:

```
# systemctl start openvpn@server.service
# systemctl enable openvpn@server.service
```

Проверяем:

```
# netstat -tulnp | grep 6666
```

Если все правильно сделали, то увидим прослушиваемый порт.

Так как мы рассматриваем задачу для соединения локальных сетей, нам необходимо разрешить форвардинг пакетов и добавить в IPTABLES правила, разрешающие пересылку пакетов между подсетями (если это не было сделано ранее).

```
# Включить IP-форвардинг в ядре
echo "1" > /proc/sys/net/ipv4/ip_forward
# Разрешить входящие соединения на порт OpenVPN
iptables -A INPUT -p UDP --dport 6666 -j ACCEPT
# Разрешить форвардинг между подсетью OpenVPN и локальной
# Правила следует поставить в начале цепочки, в случае если уже определены другие правила
iptables -A FORWARD -s 10.0.1.0/24 -d 10.0.2.0/24 -j ACCEPT
iptables -A FORWARD -d 10.0.2.0/24 -s 10.0.1.0/24 -j ACCEPT
```

Также не забудьте изменить опцию IP-forwarding в **/etc/sysctl.conf**

Рассмотрим работу на клиенте. Подключаемся по **ssh** на компьютер 30.1.2.2 (VPN-клиент). Устанавливаем **openvpn**:

```
# sudo apt-get install openvpn
```

Распаковываем принятые ранее файлы:

```
# cp /home/user/clientfiles.tgz /etc/openvpn
# cd /etc/openvpn
# tar -xzf clientfiles.tgz
```

Настраиваем файл конфигурации **client.conf**:

```
dev tun
proto udp
remote 80.1.2.2 6666                # Здесь IP-адрес сервера, видимый на данный
момент
client
resolv-retry infinite
ca /etc/openvpn/ca.crt
cert /etc/openvpn/client.crt
key /etc/openvpn/client.key
route 10.0.1.0 255.255.255.0 # Подсеть со стороны сервера
persist-key
persist-tun
comp-lzo
verb 3
status /var/log/openvpn/openvpn-status.log 1
status-version 3
log-append /var/log/openvpn/openvpn-client.log
```

Создаем каталог для логов:

```
# mkdir /var/log/openvpn
```

Запускаем:

```
# systemctl start openvpn@client.service
# systemctl enable openvpn@client.service
```

Также можно проверить:

```
# ifconfig
# route
# ping 10.0.1.1
```

Чтобы наши абоненты тоже могли подключаться к другой половине сети:

```
# Включить IP-форвардинг в ядре
echo "1" > /proc/sys/net/ipv4/ip_forward
# Разрешить форвардинг между подсетью OpenVPN и локальной
# Правила следует поставить в начале цепочки,если определены другие правила
iptables -A FORWARD -s 10.0.1.0/24 -d 10.0.2.0/24 -j ACCEPT
iptables -A FORWARD -d 10.0.2.0/24 -s 10.0.1.0/24 -j ACCEPT
```

Если мы хотим, чтобы клиентская сеть еще пользовалась OpenVPN-сервером для шлюза доступа в интернет, понадобится на сервере настроить NAT через **iptables**.

Дополнительные материалы

1. Настройка сети из консоли в Ubuntu.
2. Настройка Wi-Fi в Ubuntu.
3. Настройка iptables и fail2ban.
4. iptables в примерах.
5. iptables-persistent.