Chapter 12

Network Configuration

Chapter Objectives

1. Describe the purpose and types of networks, protocols, and media access methods.
2. Explain the basic configuration of IP.
3. Configure a network interface to use IP.
4. Configure a PPP interface.
5. Describe the purpose of host names and how they are resolved to IP addresses.
6. Configure IP routing.
7. Identify common network services.
8. Use command-line and graphical utilities to perform remote administration.

Throughout this book, you have examined the installation and administration of local Linux services. This chapter focuses on configuring Linux to participate on a network. First, you become acquainted with some common network terminology, then you learn about IP and the procedure for configuring a network interface. Next, you learn about the Domain Name Space and the processes by which host names are resolved to IP addresses. Finally, you learn how to configure IP routing as well as how to set up and use various utilities to perform remote administration of a Linux system.

Networks

Most functions that computers perform today involve the sharing of information between computers. Information is usually transmitted from computer to computer via media such as fiber optic, telephone, coaxial, or unshielded twisted pair (UTP) cable, but it can also be transmitted via wireless media such as radio, micro, or infrared waves. These media typically interact directly with a peripheral device on the computer, such as a network interface or modem.

Two or more computers connected via media that can exchange information are called a **network**. Networks that connect computers within close proximity are called **local area networks (LANs)**, whereas networks that connect computers separated by large distances are called **wide area networks (WANs)**.

Many companies use LANs to allow employees to connect to databases and other shared resources such as shared files and printers. Home users can also use LANs to connect several home computers together. Alternatively, home users can use a WAN to connect home computers to an Internet service provider (ISP) to gain access to resources such as websites on the worldwide public network called the Internet.

Note 1

The Internet (the name is short for "internetwork") is merely several interconnected public networks. Both home and company networks can be part of the Internet. Recall from Chapter 1 that special computers called routers transfer information from one network to another.

Network media serve as the conduit for information as it travels across a network. But sending information through this conduit is not enough. For devices on the network to make sense of this information, it must be organized according to a set of rules, or protocols. A network **protocol** breaks information down into **packets** that can be recognized by workstations, routers, and other devices on a network.

While you can configure many network protocols in Linux, nearly all Linux computers use the following three protocols by default:

- **Transmission Control Protocol/Internet Protocol (TCP/IP)**, which provides reliable communication of packets across the Internet.
- **User Datagram Protocol/Internet Protocol (UDP/IP)**, which provides fast, yet unreliable communication of packets across the Internet.
- **Internet Control Message Protocol (ICMP)**, which is used to send network-related information and error messages across the Internet.

Note 2

While both TCP/IP and UDP/IP can send information packets across the Internet, TCP/IP is the most common network protocol used today, and the one that we'll focus on within this chapter.

Note 3

When transmitting information across a WAN, you might use a WAN protocol in addition to a specific LAN protocol to format packets for safer transmission. The most common WAN protocol is **Point-to-Point Protocol (PPP)**.

Another important part of the puzzle, the **media access method**, is a set of rules that govern how the various devices on the network share the network media. The media access method is usually contained within the hardware on the network interface or modem. Although many media access methods are available, the one most used to send packets onto network media is called **Ethernet**. It ensures that any packets are retransmitted onto the network if a network error occurs.

Note 4

Wireless-Fidelity (Wi-Fi) LANs also use Ethernet to place packets onto the network medium (in this case, the air).

The IP Protocol

TCP/IP is actually a set, or suite, of protocols with two core components: TCP and IP. Together, these two protocols ensure that information packets travel across a network as quickly as possible, without getting lost or mislabeled.

When you transfer information across a network such as the Internet, that information is often divided into many thousands of small IP packets. Each of these packets may take a different physical route when reaching its destination as routers can transfer information to multiple interconnected

networks. TCP ensures that packets can be assembled in the correct order at their destination regardless of the order in which they arrive. Additionally, TCP ensures that any lost packets are retransmitted.

Note 5

UDP is an alternative to TCP that does not provide packet ordering or retransmission.

IP is responsible for labeling each packet with the destination address. Recall from Chapter 1 that each computer that participates on an IP network must have a valid Internet Protocol (IP) address that identifies itself to the IP protocol. Many computers on the Internet use a version of the IP protocol called **IP version 4 (IPv4)**. However, nearly all IoT devices and an increasing number of computers now use a next-generation IP protocol called **IP version 6 (IPv6)**. We will examine the structure and configuration of IPv4 and IPv6 in this chapter.

The IPv4 Protocol

To participate on an IPv4 network, your computer must have a valid IP address as well as a **subnet mask**. Optionally, you can configure a **default gateway** to participate on larger networks such as the Internet.

IPv4 Addresses

An IP address is a unique number assigned to the computer that identifies itself on the network, similar to a unique postal address that identifies your location in the world. If any two computers on the same network have the same IP address, it is impossible for information to be correctly delivered to them. Directed communication from one computer to another single computer using IP is referred to as a **unicast**.

The most common format for IPv4 addresses is four numbers called **octets** that are separated by periods. Each octet represents an 8-bit binary number (0–255). An example of an IP address in this notation is 192.168.5.69.

You can convert between decimal and binary by recognizing that an 8-bit binary number represents the decimal binary powers of two in the following order:

```
128 64 32 16 8 4 2 1
```

Thus, the number 255 is 11111111 (128+64+32+16+8+4+2+1) in binary, and the number 69 is 01000101 (64+4+1) in binary. When the computer looks at an IP address, the numbers are converted to binary. To learn more about binary/decimal number conversion, visit wikihow.com/Convert-from-Decimal-to-Binary.

All IPv4 addresses are composed of two parts: the network ID and the host ID. The **network ID** represents the network on which the computer is located, whereas the **host ID** represents a single computer on that network. No two computers on the same network can have the same host ID; however, two computers on different networks can have the same host ID.

The network ID and the host ID are similar to postal mailing addresses, which are made up of a street name and a house number. The street name is like a network ID. No two streets in the same city can have the same name, just as no two networks can have the same network ID. The host ID is like the house number. Two houses can have the same house number as long as they are on different streets, just as two computers can have the same host ID as long as they are on different networks.

Only computers with the same network ID can communicate with each other without the use of a router. This allows administrators to logically separate computers on a network; computers in the Accounting Department could use one network ID, whereas computers in the Sales Department could use a different network number. If the two departments are connected by a router, computers in the Accounting Department can communicate with computers in the Sales Department and vice versa.

Note 6

If your IP network is not connected to the Internet, the choice of IP address is entirely up to you. However, if your network is connected to the Internet, you might need to use preselected IP addresses for the computers on your network. IP addresses that can be used on the public Internet are assigned by your Internet service provider.

Note 7

The IP address 127.0.0.1 is called the loopback IP address. It always refers to the local computer. In other words, on your computer, 127.0.0.1 refers to your computer. On your coworker's computer, 127.0.0.1 refers to your coworker's computer.

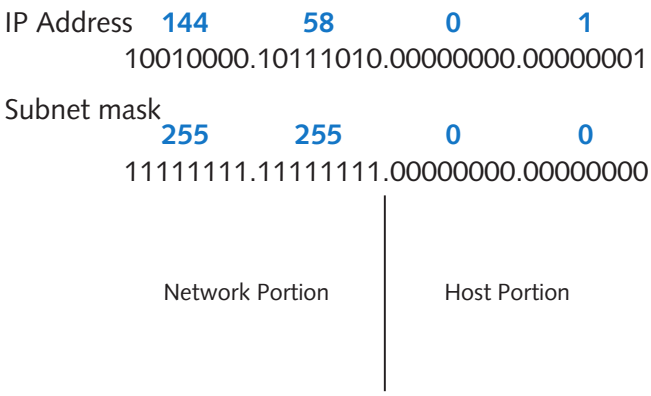
Subnet Masks

Each computer with an IPv4 address must also be configured with a subnet mask to define which part of its IP address is the network ID and which part is the host ID. Subnet masks are composed of four octets, just like an IP address. The simplest subnet masks use only the values 0 and 255. An octet in a subnet mask containing 255 is part of the network ID. An octet in a subnet mask containing 0 is part of the host ID. Your computer uses the binary process called **ANDing** to find the network ID. ANDing is a mathematical operation that compares two binary digits and gives a result of 1 or 0. If both binary digits being compared have a value of 1, the result is 1. If one digit is 0 and the other is 1, or if both digits are 0, the result is 0.

When an IP address is ANDed with a subnet mask, the result is the network ID. Figure 12-1 shows an example of how the network ID and host ID of an IP address can be calculated using the subnet mask.

Thus, the IP address shown in Figure 12-1 identifies the first computer (host portion 0.1) on the 144.58 network (network portion 144.58).

Figure 12-1 A sample IP address and subnet mask



Note 8

IP addresses and their subnet masks are often written using the **classless interdomain routing (CIDR) notation**. For example, the notation 144.58.0.1/16 refers to the IP address 144.58.0.1 with a 16-bit subnet mask (255.255.0.0).

The IP addresses 0.0.0.0 and 255.255.255.255 cannot be assigned to a host computer because they refer to all networks and all computers on all networks, respectively. Similarly, using the number 255 (all 1s in binary format) in an IP address can specify many hosts. For example, the IP address 192.168.131.255 refers to all hosts on the 192.168.131 network; this IP address is also called the **broadcast** address for the 192.168.131 network.

Note 9

A computer uses its IP address and subnet mask to determine what network it is on. If two computers are on the same network, they can deliver packets directly to each other. If two computers are on different networks, they must use a router to communicate.

Default Gateway

Typically, all computers on a LAN are configured with the same network ID and different host IDs. A LAN can connect to another LAN by means of a router, which has IP addresses for both LANs and can forward packets to and from each network. Each computer on a LAN can contain the IP address of a router in its IP configuration; any packets that are not destined for the local LAN are then sent to the router, which can forward the packet to the appropriate network or to another router. The IP address of the network interface on the router to which you send packets is called the default gateway.

A router is often a dedicated hardware device from a vendor such as Cisco, D-Link, or HP. Other times, a router is a computer with multiple network cards. The one consistent feature of routers, regardless of the manufacturer, is that they can distinguish between different networks and move (or route) packets between them. A router has an IP address on every network to which it is attached. When a computer sends a packet to the default gateway for further delivery, the address of the router must be on the same network as the computer, as computers can send packets directly to devices only on their own network.

IPv4 Classes and Subnetting

IPv4 addresses are divided into classes to make them easier to manage. The class of an IP address defines the default subnet mask of the device using that address. All the IP address classes can be identified by the first octet of the address, as shown in Table 12-1.

Table 12-1 IP address classes

Class	Subnet Mask	First Octet	Maximum Number of Networks	Maximum Number of Hosts	Example IP Address
A	255.0.0.0	1–127	127	16,777,214	3.4.1.99
B	255.255.0.0	128–191	16,384	65,534	144.129.188.1
C	255.255.255.0	192–223	2,097,152	254	192.168.1.1
D	N/A	224–239	N/A	N/A	224.0.2.1
E	N/A	240–254	N/A	N/A	N/A

Class A addresses use 8 bits for the network ID and 24 bits for the host ID. You can see this is true by looking at the subnet mask, 255.0.0.0. The value of the first octet will always be somewhere in the range 1 to 127. This means there are only 127 potential Class A networks available for the entire Internet. Class A networks are only assigned to very large companies and Internet providers.

Class B addresses, which are identified by the subnet mask 255.255.0.0, use 16 bits for the network ID and 16 bits for the host ID. The value of the first octet ranges from 128 to 191. There are 16,384 Class B networks, with 65,534 hosts on each network. Class B networks are assigned to many larger organizations, such as governments, universities, and companies with several thousand users.

Class C addresses, which are identified by the subnet mask 255.255.255.0, use 24 bits for the network ID and 8 bits for the host ID. The value of the first octet ranges from 192 to 223. There are 2,097,152 Class C networks, with 254 hosts on each network. Although there are very many Class C networks, they have a relatively small number of hosts; thus, they are suited only to smaller organizations.

Class D addresses are not divided into networks, and they cannot be assigned to computers as IP addresses; instead, Class D addresses are used for multicasting. **Multicast** addresses are used by groups of computers. A packet addressed to a multicast address is delivered to each computer in the multicast group. This is better than a broadcast message because routers can be configured to allow multicast traffic to move from one network to another. In addition, all computers on the network process broadcasts, while only computers that are part of that multicast group process multicasts. Streaming media and network conferencing software often use multicasting to communicate to several computers at once.

Like Class D addresses, Class E addresses are not typically assigned to a computer. Class E addresses are considered experimental and are reserved for future use.

Notice from Table 12-1 that Class A and B networks can have many thousands or millions of hosts on a single network. Because this is not practically manageable, Class A and B networks are typically subnetted. **Subnetting** is the process in which a single large network is subdivided into several smaller networks to control traffic flow and improve manageability. After a network has been subnetted, a router is required to move packets from one subnet to another.

Note 10

You can subnet any Class A, B, or C network.

To subnet a network, you take some bits from the host ID and give them to the network ID. Suppose, for example, that you want to divide the 3.0.0.0/8 network into 17 subnets. The binary representation of this network is:

```
3.0.0.0 = 00000011.00000000.00000000.00000000
255.0.0.0 = 11111111.00000000.00000000.00000000
```

You then borrow some bits from the host portion of the subnet mask. Because the number of combinations of binary numbers can be represented in binary powers of two, and because valid subnet masks do not contain all 0s or 1s, you can use the equation 2^n to represent the minimum number of subnets required, where n is the number of binary bits that are borrowed from the host portion of the subnet mask. For our example, this is represented as:

$$2^n \geq 15$$

Thus, $n = 4$ (because $2^3 = 8$ is less than 15, but $2^4 = 16$ is greater than 15). Following this, our subnet mask borrows four bits from the default Class A subnet mask:

```
255.240.0.0 = 11111111.11110000.00000000.00000000
```

Similarly, because there are 20 zeros in the preceding subnet mask, you can use the $2^n - 2$ equation to identify the number of hosts per subnet (the –2 accounts for the broadcast and network address for the subnet):

$$2^{20} - 2 = \text{number of hosts per subnet}$$
$$= 1{,}048{,}574 \text{ hosts per subnet}$$

You can then work out the IP address ranges for each of the network ranges. Because four bits in the second octet were not borrowed during subnetting, the ranges of IP addresses that can be given to each subnet must be in ranges of $2^4 = 16$. Thus, the first five ranges that can be given to different subnets on the 3.0.0.0/8 network that use the subnet mask 255.240.0.0 are as follows:

```
3.0.0.1–3.15.255.254
3.16.0.1–3.31.255.254
3.32.0.1–3.47.255.254
3.48.0.1–3.63.255.254
3.64.0.1–3.79.255.254
```

From the preceding ranges, a computer with the IP address 3.34.0.6/12 cannot communicate directly with the computer 3.31.0.99/12 because they are on different subnets. To communicate, there must be a router between them.

Note 11

When subnetting a Class C network, ensure that you discard the first and last IP address in each range to account for the broadcast and network address for the subnet.

The IPv6 Protocol

As the Internet grew in the 1990s, ISPs realized that the number of IP addresses available using IPv4 was inadequate to accommodate future growth. As a result, the IPv6 protocol was designed in 1998 to accommodate far more IP addresses. IPv6 uses 128 bits to identify computers, whereas IPv4 only uses 32 bits (4 octets). This allows IPv6 to address up to 340,282,366,920,938,463,463,374,607,431,768,211,456 (or 340 trillion trillion trillion) unique computers.

IPv6 IP addresses are written using 8 colon-delimited 16-bit hexadecimal numbers—for example, 2001:0db8:3c4d:0015:0000:0000:adb6:ef12. If an IPv6 IP address contains 0000 segments, they are often omitted in most notation, thus 2001:0db8:3c4d:0015:::adb6:ef12 is equivalent to 2001:0db8:3c4d:0015:0000:0000:adb6:ef12. The IPv6 loopback address is 0000:0000:0000:0000:0000:0000:0000:0001, but it is often referred to as ::1 for simplicity.

Note 12

Unlike our traditional decimal numbering scheme, hexadecimal uses an expanded numbering system that includes the letters A through F in addition to the numbers 0–9. Thus, the number 10 is called A in hexadecimal, the number 11 is called B in hexadecimal, the number 12 is called C in hexadecimal, the number 13 is called D in hexadecimal, the number 14 is called E in hexadecimal, and the number 15 is called F in hexadecimal.

Although IPv6 addresses can be expressed several ways, the first half (64 bits) of an IPv6 address identifies your network (the network ID); the first 46 bits are typically assigned by your ISP and identify your organization uniquely on the public Internet, and the following 16 bits can be used to identify unique subnets within your organization. The last 64 bits of an IPv6 address is used to uniquely identify a computer in your LAN (the host ID) and is often generated from the unique hardware address on each computer's network interface.

Note 13

The hardware address on a network interface is a 48-bit hexadecimal number called the **Media Access Control (MAC) address** that is unique for each network interface manufactured. Ethernet translates IPv4- and IPv6-addressed packets into MAC-addressed frames before sending it to the nearest host or router.

Although all operating systems today support IPv6, it has not replaced IPv4 on most server and client computers because the addressing is more difficult to work with compared to IPv4. Instead, IPv6 has primarily allowed the plethora of IoT devices available today to participate on the Internet without affecting the available IPv4 address space. Some example IoT devices include the NEST smart thermostat and the Google Home personal assistant; both of which run the Linux operating system. IoT devices often use an automatically configured IPv6 address that can be accessed by an online app, and the IPv6 traffic they send is often encapsulated in IPv4 traffic using a protocol such as **Teredo** to allow it to work within IPv4-only networks.

This slow adoption of IPv6 among server and client computers is primarily the result of two technologies introduced in Chapter 1 that allow IPv4 to address many more computers than was previously possible: proxy servers and Network Address Translation (NAT) routers.

Proxy servers and NAT routers are computers or hardware devices that have an IP address and access to a network such as the Internet. Other computers on the network can use a proxy server or NAT router to obtain network or Internet resources on their behalf. Moreover, there are three reserved ranges of IPv4 addresses that are not distributed to computers on the Internet and are intended only for use behind a proxy server or NAT router:

- The entire 10.0.0.0 Class A network (10.0.0.0/8)
- The 172.16 through 172.31 Class B networks (172.16–31.0.0/16)
- The 192.168 Class C networks (192.168.0–255.0/24)

Thus, a computer behind a proxy server in Iceland and a computer behind a NAT router in Seattle could use the same IPv4 address—say, 10.0.5.4—without problems because each of these computers only requests Internet resources using its own proxy server or NAT router. A company may use a Cisco NAT router, for example, to allow other networks and computers in the company to gain access to the Internet. Similarly, a high-speed home Internet modem typically functions as a NAT router to allow multiple computers in your home to access the Internet.

Most computers in the world today obtain Internet access via a proxy server or NAT router. Because these computers share IPv4 addresses on a reserved network range, rather than using a unique IP address, the number of available IPv4 addresses has remained high and slowed the adoption of IPv6.

Configuring a Network Interface

Linux computers in a business environment typically connect to the company network via a wired or wireless network interface. At home, people typically connect to the Internet by means of a network interface, using technologies such as Fiber optic, cellular wireless, Wi-Fi, Digital Subscriber Line (DSL), and Broadband Cable Networks (BCNs).

Recall from Chapter 6 that network interface drivers are provided by modules that are inserted into the Linux kernel at boot time and given an alias that can be used to refer to the network interface afterwards. To view the driver modules and associated aliases for network interfaces in your system, you can use the `lshw -class network` command. On legacy Linux systems, the first wired Ethernet network interface is called eth0, the second wired Ethernet network interface is called eth1, and so on. Similarly, the first wireless Ethernet network interface on a legacy Linux system is called wlan0, the second wireless Ethernet network interface in your system is called wlan1, and so on. While this naming convention can also be used on modern Linux systems, some modern Linux distributions that use Systemd instead provide a more descriptive name that reflects the location of the hardware in the system. For example, enp0s3 refers to the wired Ethernet network interface on PCI bus 00 slot 03, and wlp8s0 refers to the wireless Ethernet network interface on PCI bus 08 slot 00. The information reflected by these names will match the PCI information displayed by the `lspci` command for the network interface hardware.

Note 14

You can also use the **`ethtool` command** to display detailed information for network hardware. For example, `ethtool -i enp0s3` will display driver information for the enp0s3 network interface.

After a driver module for a network interface has been loaded into the Linux kernel and given an alias, you can configure it to use IP. Legacy Linux systems use the UNIX **`ifconfig` (interface configuration) command** to assign an IP configuration to a network interface. For example, to assign eth0 the IPv4 address of 3.4.5.6 with a subnet mask of 255.0.0.0 and broadcast address of 3.255.255.255,

you can use the `ifconfig eth0 3.4.5.6 netmask 255.0.0.0 broadcast 3.255.255.255` command. Modern Linux systems use the **`ip` command** to configure IP. For example, you can use the `ip addr add 3.4.5.6/8 dev eth0` command to configure eth0 with the IPv4 address 3.4.5.6 and an 8-bit subnet mask (255.0.0.0).

Note 15

You can install the net-tools package on a modern Linux distribution to obtain the `ipconfig` command or install the iproute or iproute2 package on a legacy Linux distribution to obtain the `ip` command.

Alternatively, you can receive IP configuration from a Dynamic Host Configuration Protocol (DHCP) or Boot Protocol (BOOTP) server on the network. To obtain and configure IP information from a server on the network, you can use the **`dhclient` command**; for example, `dhclient eth0` would attempt to obtain IP configuration from the network connected to eth0.

The process of obtaining an IP address for your network interface varies, depending on whether your computer is on an IPv4 or IPv6 network. If you attempt to obtain IPv4 configuration for your network interface from a DHCP or BOOTP server and no DHCP or BOOTP server exists on your network, your system will assign an IPv4 address of 169.254.*x.x* where *.x.x* is a randomly generated host ID. This automatic assignment feature is called **Automatic Private IP Addressing (APIPA)**. If your network has IPv6-configured routers, an IPv6 address is automatically assigned to each network interface. This is because network interfaces use **Internet Control Message Protocol version 6 (ICMPv6)** router discovery messages to probe their network for IPv6 configuration information. Alternatively, you can obtain your IPv6 configuration from a DHCP server on the network. If there are no IPv6-configured routers or DHCP servers on your network from which you can obtain an IPv6 configuration for your network interface, your system will assign an IPv6 APIPA address that begins with fe80.

Note 16

A single network interface can have both an IPv4 and an IPv6 address. Each address can be used to access the Internet using the IPv4 and IPv6 protocols, respectively.

To view the configuration of all interfaces, you can use the `ifconfig` command without arguments, or the `ip addr` command (a shortcut to `ip addr show`), as shown in the following output:

```
[root@server1 ~]# ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 3.4.5.6 netmask 255.0.0.0 broadcast 3.255.255.255
        inet6 fe80::bbc3:3b03:591:801b prefixlen 64 scopeid 0x20<link>
        ether 00:15:5d:01:6e:d6  txqueuelen 1000  (Ethernet)
        RX packets 955  bytes 111767 (109.1 KiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 272  bytes 61744 (60.2 KiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        inet6 ::1  prefixlen 128  scopeid 0x10<host>
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 74  bytes 26434 (25.8 KiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 74  bytes 26434 (25.8 KiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

```
[root@server1 ~]# ip addr
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue
    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
       valid_lft forever preferred_lft forever
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq state UP
    link/ether 00:15:5d:01:6e:d6 brd ff:ff:ff:ff:ff:ff
    inet 3.4.5.6/8 brd 3.255.255.255 scope global dynamic eth0
       valid_lft 258581sec preferred_lft 258581sec
    inet6 fe80::bbc3:3b03:591:801b/64 scope link noprefixroute
       valid_lft forever preferred_lft forever
[root@server1 ~]#_
```

The previous output shows that the eth0 network interface has an IPv4 address of `3.4.5.6` and an IPv6 address of `fe80::bbc3:3b03:591:801b` that you can tell was automatically configured by the system because it starts with `fe80`. The special loopback adapter (`lo`) is configured automatically using the IPv4 address 127.0.0.1 and IPv6 address ::1; these IP addresses represent the local computer and are required on all computers that use IP. The `ifconfig` command also displays receive (RX) and transmit (TX) statistics for each network interface.

Note 17

You can also use the `-i` option to the **`netstat` command** to show interface statistics.

If you restart the computer, the IP information configured for eth0 will be lost. To allow the system to activate and configure the IP information for an interface at each boot time, you can place entries in a configuration file that is read at boot time by your Linux distribution when activating the network. On legacy Ubuntu systems you could add entries for eth0 to a file called /etc/network/interfaces, and on legacy Fedora systems you could add entries to a /etc/sysconfig/network-scripts/ifcfg-eth0 file. Following this, you could run `ifconfig eth0 down ; ifconfig eth0 up` or `ifdown eth0 ; ifup eth0` or `ip link set eth1 down ; ip link set eth0 up` to deactivate and then reactivate your IP configuration based on the lines within these files. However, on modern Linux systems, the configuration of IP is typically performed at system initialization by a **network renderer** service, such as **NetworkManager** or **Systemd-networkd**.

On modern Fedora Workstation systems, NetworkManager configures network interfaces at system initialization using IP configuration information stored within *.nmconnection files under the /etc/NetworkManager/system-connections directory. If no files exist under this directory, NetworkManager obtains IP configuration for each network interface using DHCP. A sample /etc/NetworkManager/system-connections/Wired1.nmconnection file that configures IP on the eth0 network interface is shown in the following output:

```
[root@server1 ~]# cd /etc/NetworkManager/system-connections
[root@localhost system-connections]# cat Wired1.nmconnection
[connection]
id=Wired1
uuid=e4006c02-9479-340a-b065-01a328727a75
type=ethernet
interface-name=eth0

[ipv4]
address1=3.4.5.6/8,3.0.0.1
dns=8.8.8.8;1.1.1.1;
```

```
method=manual

[ipv6]
addr-gen-mode=stable-privacy
method=auto
[root@server1 ~]#_
```

The entries in the preceding output indicate that IPv6 configuration will be obtained automatically using DHCP (`method=auto`) if it has not already been obtained via ICMPv6. IPv4 is configured manually (`method=manual`) using the IP address 3.4.5.6, an 8-bit subnet mask, and a default gateway of 3.0.0.1 (`address1=3.4.5.6/8,3.0.0.1`). To resolve Internet names, the first DNS server queried will be 8.8.8.8 followed by 1.1.1.1 if the first DNS server is unavailable (`dns=8.8.8.8;1.1.1.1;`). After making changes to a *.nmconnection file, you must instruct NetworkManager to activate your changes using the **`nmcli` command**. For example, after making changes to the Wired1.nmconnection file shown in the preceding output, you can run the `nmcli connection down Wired1` to deactivate eth0, and then run the `nmcli connection up Wired1` command to activate eth0 again using the new IP configuration information in the /etc/NetworkManager/system-connections/Wired1.nmconnection file.

On modern Ubuntu Server systems, the **`netplan` command** configures network interfaces at system initialization using either the NetworkManager or Systemd-networkd renderer from IP configuration information stored within *.yaml files under the /etc/netplan directory. A sample /etc/netplan/00-installer-config.yaml file that instructs Systemd-networkd to configure IP on eth0 is shown in the following output:

```
[root@server1 ~]# cat /etc/netplan/00-installer-config.yaml
network:
 version: 2
 renderer: networkd
 ethernets:
   eth0:
     dhcp4: no
     dhcp6: yes
     addresses: [3.4.5.6/8]
     gateway4: 3.0.0.1
     nameservers:
       addresses: [8.8.8.8,1.1.1.1]
[root@server1 ~]#_
```

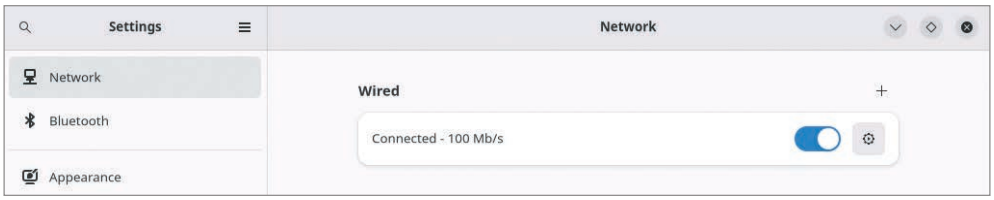
The entries in the preceding output indicate that IPv6 configuration will be obtained automatically using DHCP (`dhcp6: yes`) if it has not already been obtained via ICMPv6. IPv4 is configured manually (`dhcp6: no`) using the IP address 3.4.5.6 alongside an 8-bit subnet mask (`addresses: [3.4.5.6/8]`) and a default gateway of 3.0.0.1 (`gateway4: 3.0.0.1`). The DNS servers (`nameservers:`) used to resolve Internet names will be 8.8.8.8 followed by 1.1.1.1 if 8.8.8.8 is unavailable (`addresses: [8.8.8.8,1.1.1.1]`). After modifying the /etc/netplan/00-installer-config.yaml file, you can activate your new configuration by using the `netplan apply` command.

Note 18

YAML (YAML Ain't Markup Language) is a file format that uses the `attribute: value` syntax defined by JSON (JavaScript Object Notation), but with additional support for comments. Due to its simplicity, YAML is increasingly being used to provide configuration information for modern Linux systems and cloud-related services.

To make the configuration of a network interface easier, you may access your configuration from the Settings utility within a desktop environment. In GNOME, you can navigate to Activities, Show Applications, Settings to open the Settings utility, navigate to Network, and click the settings (cog wheel) button next to your wired or wireless network interface, as shown in Figure 12-2.

Figure 12-2 Network interface configuration within GNOME Settings



Most workstation-focused Linux distributions use NetworkManager as the network renderer. NetworkManager was designed to keep track of the many different wired and wireless networks that a portable workstation may connect to over time and is supported by all desktop environments that allow you to switch between different networks using graphical tools. If your system is running NetworkManager, you can use the `nmcli` command without arguments to display information about each network interface, including the active connection as shown in the following output:

```
[root@server1 ~]# nmcli
wlan0: connected to CLASSWIFI
      "Qualcomm Atheros AR93xx Wireless Network Adapter (Killer
           Wireless-N 1103 Half-size Mini PCIe Card [AR9380])"
      wifi (ath9k), B8:76:3F:16:F9:F4, hw, mtu 1500
      ip4 default
      inet4 192.168.1.101/24
      route4 0.0.0.0/0
      route4 192.168.1.0/24
      inet6 fe80::3157:d86d:5da1:2292/64
      route6 ff00::/8
      route6 fe80::/64
      route6 fe80::/64

eth0: unavailable
      "Qualcomm Atheros AR8151 v2.0 Gigabit Ethernet"
      ethernet (atl1c), B8:CA:3A:D5:61:A9, hw, mtu 1500

lo: unmanaged
      "lo"
      loopback (unknown), 00:00:00:00:00:00, sw, mtu 65536

DNS configuration:
      servers: 192.168.1.1
      interface: wlan0

Use "nmcli device show" to get complete information about known devices
and "nmcli connection show" to get an overview on active connection profiles.

Consult nmcli(1) and nmcli-examples(5) manual pages for complete usage details.
[root@server1 ~]#_
```

In the previous output, the wireless Ethernet network interface wlan0 is connected to a wireless network called CLASSWIFI. On a Fedora system, NetworkManager will store the network configuration and Wi-Fi password for this connection in a separate file called /etc/NetworkManager/system-connections/CLASSWIFI.nmconnection, such that it can be reused if the system connects to the network in the future. This is especially useful if you manually configure IP addresses for a wireless network, such as a home or work network; each time you connect to the wireless network, the

IP address information you configured previously will automatically be applied from the associated configuration file. To see a list of all wired and wireless networks that NetworkManager has connected to in the past, you can run the `nmcli conn show` command as shown in the following output:

```
[root@server1 ~]# nmcli conn show
NAME        UUID                                  TYPE      DEVICE
CLASSWIFI   562a0a70-7944-4621-ac4d-35572f08b1ed  wifi      wlan0
Starbucks   59683e0b-5e1d-405c-b88f-8289899b60bd  wifi      --
McDonalds   0dc84378-4e04-4b43-a7fc-2ce25cff9401  wifi      --
HomeWifi    1966c022-8cc5-422c-9fcf-1fdfd8cdef4e  wifi      --
Wired1      f1b34bb3-b621-36d0-bdd0-c23e646c9a51  ethernet  --
[root@server1 ~]#_
```

While Systemd-networkd can also be used to provide the same functionality as NetworkManager, it is more commonly used by Linux server distributions that do not use a desktop environment. You can use the **`networkctl` command** to view and modify connection information for Systemd-networkd; without arguments, it displays information about each network interface, including the active connection. The following output indicates that eth0 is actively connected and managed by Systemd-networkd:

```
[root@server1 ~]# networkctl
IDX LINK             TYPE               OPERATIONAL SETUP
  1 lo               loopback           carrier     unmanaged
  2 eth0             ether              routable    configured
  3 wlan0            ether              no-carrier  unmanaged

3 links listed.
[root@server1 ~]#_
```

Note 19

Both NetworkManager and Systemd-networkd are optional. If used, only one can be active at any time.

After a network interface has been configured to use IP, you should test the configuration by using the **`ping` (Packet Internet Groper) command**. The `ping` command sends an ICMP packet to another IP address and awaits a response. By default, the `ping` command sends packets continuously every second until the Ctrl+c key combination is pressed; to send only five ping requests to the loopback interface, you can use the `-c` option to the `ping` command, as shown in the following example:

```
[root@server1 ~]# ping -c 5 127.0.0.1
PING 127.0.0.1 (127.0.0.1) 56(84) bytes of data.
64 bytes from 127.0.0.1: icmp_seq=0 ttl=64 time=0.154 ms
64 bytes from 127.0.0.1: icmp_seq=1 ttl=64 time=0.109 ms
64 bytes from 127.0.0.1: icmp_seq=2 ttl=64 time=0.110 ms
64 bytes from 127.0.0.1: icmp_seq=3 ttl=64 time=0.119 ms
64 bytes from 127.0.0.1: icmp_seq=4 ttl=64 time=0.111 ms

--- 127.0.0.1 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 3999ms
rtt min/avg/max/mdev = 0.109/0.120/0.154/0.020 ms
[root@server1 ~]#_
```

Note 20

If the `ping` command fails to receive any responses from the loopback interface, there is a problem with IP itself.

Next, you need to test whether the Linux computer can ping other computers on the same network; the following command can be used to send five ping requests to the computer that has the IP address 3.0.0.2 configured:

```
[root@server1 ~]# ping -c 5 3.0.0.2
PING 3.0.0.2 (3.0.0.2) 56(84) bytes of data.
64 bytes from 3.0.0.2: icmp_seq=0 ttl=128 time=0.448 ms
64 bytes from 3.0.0.2: icmp_seq=1 ttl=128 time=0.401 ms
64 bytes from 3.0.0.2: icmp_seq=2 ttl=128 time=0.403 ms
64 bytes from 3.0.0.2: icmp_seq=3 ttl=128 time=0.419 ms
64 bytes from 3.0.0.2: icmp_seq=4 ttl=128 time=0.439 ms

--- 3.0.0.2 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4001ms
rtt min/avg/max/mdev = 0.401/0.422/0.448/0.018 ms
[root@server1 ~]#_
```

Note 21

If the `ping` command fails to receive any responses from other computers on the network, there is a problem with the network media.

You can use the **`ping6` command** to send an ICMP6 message to an IPv6 address.

Configuring a PPP Interface

Instead of configuring IP to run on a network interface to gain network access, you can run IP over serial lines (such as telephone lines) using the PPP WAN protocol. Three common technologies use PPP to connect computers to the Internet or other networks:

- Modems
- ISDN
- DSL

Modem (modulator-demodulator) devices use PPP to send IP information across normal telephone lines; they were the most common method for home users to gain Internet access in the 1990s. Modem connections are considered slow today compared to most other technologies; most modems can only transmit data at 56 Kb/s. Because modems transmit information on a serial port, the system typically makes a symbolic link called /dev/modem that points to the correct serial port device, such as /dev/ttyS0 for COM1.

Integrated Services Digital Network (ISDN) is a set of standards designed for transmitting voice, video, and data over normal copper telephone lines. It allows data to be transferred at 128 Kb/s. ISDN uses an ISDN modem device to connect to a different type of media than regular phone lines.

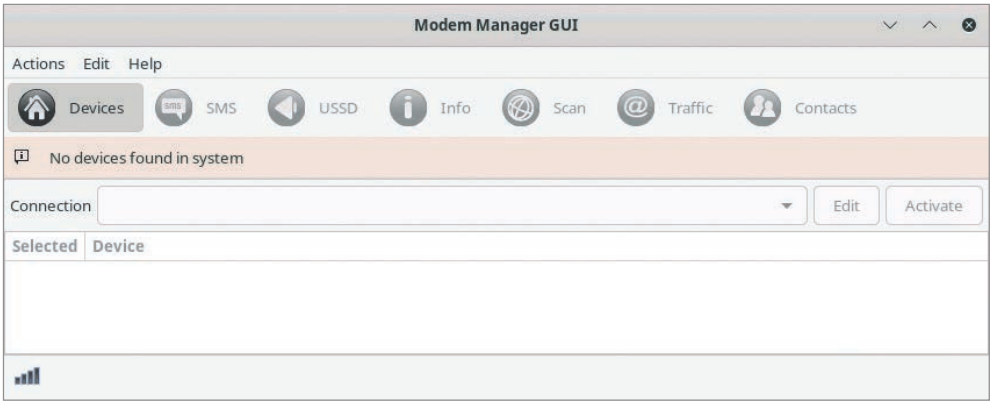
While modems and ISDN are rare today, many home and rural networks connect to the Internet using DSL. DSL has many variants, such as Asynchronous DSL (ADSL), which is the most common DSL used in homes across North America, and High-bit-rate DSL (HDSL), which is common in business environments; for simplification, all variants of DSL are referred to as xDSL. You use an Ethernet network interface to connect to a DSL modem using IP and PPP; as a result, DSL connections are said to use **PPP over Ethernet (PPPoE)**. The DSL modem then transmits information across normal telephone lines at speeds that can exceed 100 Mb/s.

Because modem, ISDN, and DSL connections require additional configuration information that is specific to the ISP, they are not normally configured during the Linux installation and must be configured manually.

Configuring a PPP connection requires support for PPP compiled into the kernel or available as a module, the PPP daemon (pppd), and a series of supporting utilities such as the chat program, which is used to communicate with a modem. PPP configuration in the past was tedious at best; you needed to create a chat script that contained the necessary information to establish a PPP connection (user name, password, and so on), a connection script that contained device parameters used by the PPP daemon, as well as use a program such as minicom to initiate network communication. Because the IP configuration is typically assigned by the ISP to which you connect, it rarely needs to be configured during the process.

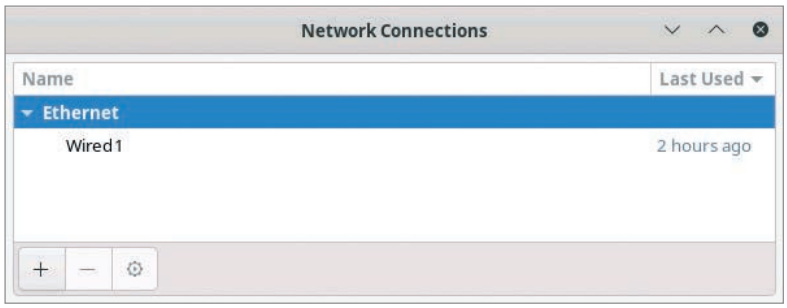
Because modems and ISDN modems are relatively rare today, modern Linux distributions typically don't ship with a graphical configuration tool by default. However, you can download and install the **Modem Manager utility** to configure modems and ISDN modems. On a Fedora system, you can run the `dnf install modem-manager-gui` command as the root user to install this package, and then navigate to Activities, Show Applications, Modem Manager GUI to start the Modem Manager shown in Figure 12-3. The Modem Manager automatically detects any modem or ISDN hardware in your computer and allows you to configure the associated ISP configuration, including the ISP account user name and password.

Figure 12-3 The Modem Manager utility



DSL connections are very common today. As a result, nearly all modern Linux workstation distributions contain a utility that can configure your network interface to work with a DSL modem. On Fedora Workstation, you can configure a DSL connection by running the **`nm-connection-editor` command** within a terminal in a desktop environment. This command starts the graphical **Network Connections utility** shown in Figure 12-4, which is a component of NetworkManager. If you click the + button shown in Figure 12-4 and choose your device type (DSL/PPPoE), the Network Connections tool will attempt to detect your DSL modem and prompt you to supply the ISP account user name and password.

Figure 12-4 The Network Connections utility



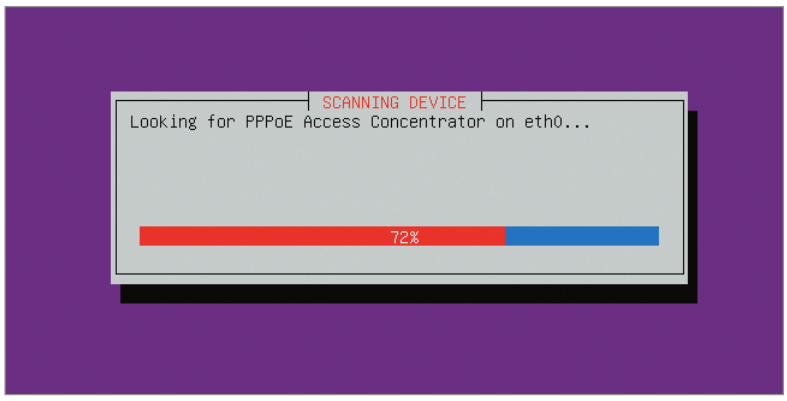
Note 22

In addition to DSL, the Network Connections tool can also be used to configure many other network technologies, including:

- Overlay networks, including **Virtual Private Network (VPN)** connections that provide an encrypted virtual IP network on top of the existing unencrypted IP network.
- Specialized technologies, such as Infiniband, that allow network interfaces to communicate directly to each other using **Remote Direct Memory Access (RDMA)**.
- **Bonding** (also called **aggregation**), in which two separate network interfaces connected to the same network can be combined to provide fault tolerance (in an active/passive configuration) or load balancing of network traffic to achieve higher transmission rates.
- **Bridging**, in which two network interfaces connected to separate networks provide for seamless connectivity between the networks.

To configure a DSL connection on a system without a desktop environment that contains NetworkManager, you can run the `nmcli` command with the appropriate options. If NetworkManager is not installed (the default on Ubuntu Server), you can instead execute the **`pppoeconf` command**, which will open a basic graphical screen within your terminal and scan for DSL devices, as shown in Figure 12-5. If a DSL device is found, you will be prompted to supply the ISP account user name and password to complete the configuration.

Figure 12-5 Scanning for DSL devices using pppoeconf



After a PPP modem, ISDN, or DSL connection has been configured, it will normally be activated automatically at boot time like other network interfaces on the system. On a Fedora Workstation system, you will notice a new *.nmconnection file for the connection under the /etc/NetworkManager/system-connections directory. On an Ubuntu Server system, there will be an additional YAML configuration file within the /etc/netplan directory. Other configuration used by the PPP daemon is stored within the /etc/ppp and /etc/isdn directories. It is good form to double-check the passwords used to connect to the ISP because incorrect passwords represent the most common problem with PPP connections. These passwords are stored in two files: /etc/ppp/pap-secrets (Password Authentication Protocol secrets) and /etc/ppp/chap-secrets (Challenge Handshake Authentication Protocol secrets). If the ISP accepts passwords sent across the network in text form, the /etc/ppp/pap-secrets file is consulted for the correct password; however, if the ISP requires a more secure method for validating the identity of a user, the passwords in the /etc/ppp/chap-secrets file are used. When you configure a PPP connection, this information is automatically added to both files, as shown in the following output:

```
[root@server1 ~]# cat /etc/ppp/pap-secrets
# Secrets for authentication using PAP
# client        server  secret                  IP addresses
```

```
####### system-config-network will overwrite this part!!! (begin) ####
"user1"         "isp"   "secret"
####### system-config-network will overwrite this part!!! (end) ######
[root@server1 ~]# cat /etc/ppp/chap-secrets
# Secrets for authentication using CHAP
# client        server  secret                  IP addresses
####### system-config-network will overwrite this part!!! (begin) ####
"user1"         "isp"   "secret"
####### system-config-network will overwrite this part!!! (end) ######
[root@server1 ~]#_
```

After a PPP device has been configured, the output of the `ifconfig` and `ip addr` commands indicate the PPP interface using the appropriate name; ppp0 is typically used for the first modem or xDSL device, and ippp0 is typically used for the first ISDN device.

Name Resolution

Computers that communicate on an IP network identify themselves using unique IP addresses; however, this identification scheme is impractical for human use because it is difficult to remember IP addresses. As a result, every computer on a network is identified by a name that makes sense to humans, such as "Accounting1" or "ReceptionKiosk." Because each computer on a network is called a host, the name assigned to an individual computer is its **host name**.

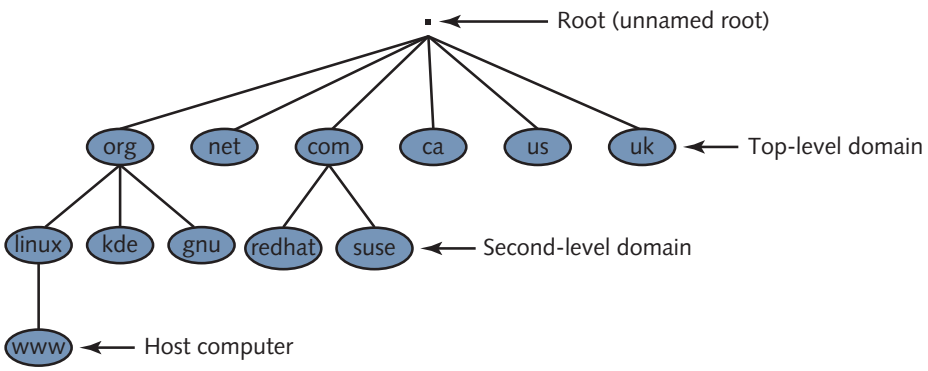
For computers that require a presence on the Internet, simple host names are rarely used. Instead, they are given a host name called a fully qualified domain name (FQDN) according to a hierarchical naming scheme called Domain Name Space (DNS), as discussed in Chapter 1. At the top of the Domain Name Space is the root domain, which is just a theoretical starting point for the branching, tree-like structure. Below the root domain are the top-level domain names, which identify the type of organization in which a network is located. For example, the com domain is primarily used for business, or commercial, networks. Several second-level domains exist under each top-level domain name to identify the name of the organization, and simple host names are listed under the second-level domains. Figure 12-6 shows a portion of the Domain Name Space.

Note 23

For simplicity, FQDNs are often referred to as host names.

Thus, the host computer shown in Figure 12-6 has an FQDN of www.linux.org.

Figure 12-6 The Domain Name Space



Note 24

The host name www (World Wide Web) has traditionally been used by servers that host webpages.

Second-level domains must be purchased and registered with an ISP in order to be recognized by other computers on the Internet. You can use the **whois command** to obtain registration information about any domain within the Domain Name Space. For example, to obtain information about the organization responsible for maintaining the linux.org domain, you can use the following command:

```
[root@server1 ~]# whois linux.org | head -22
[Querying whois.pir.org]
[whois.pir.org]
Domain Name: linux.org
Registry Domain ID: eac371bee0f24a089943a58c8b182080-LROR
Registrar WHOIS Server: http://whois.networksolutions.com
Registrar URL: http://www.networksolutions.com
Updated Date: 2019-01-08T09:43:32Z
Creation Date: 1994-05-10T04:00:00Z
Registry Expiry Date: 2027-05-11T04:00:00Z
Registrar: Network Solutions, LLC
Registrar IANA ID: 2
Registrar Abuse Contact Email: domain.operations@web.com
Registrar Abuse Contact Phone: +1.8777228662
Domain Status: clientTransferProhibited https://icann.org/epp#clientTransferProhibited
Registry Registrant ID: REDACTED FOR PRIVACY
Registrant Name: REDACTED FOR PRIVACY
Registrant Organization: Linux Online, Inc
Registrant Street: REDACTED FOR PRIVACY
Registrant City: REDACTED FOR PRIVACY
Registrant State/Province: NY
Registrant Postal Code: REDACTED FOR PRIVACY
Registrant Country: US
[root@server1 ~]#_
```

You can view or set the host name for a Linux computer using the **hostname command**, as shown in the following output:

```
[root@server1 ~]# hostname
server1.class.com
[root@server1 ~]# hostname computer1.sampledomain.com
[root@server1 ~]# hostname
computer1.sampledomain.com
[root@server1 ~]# bash
[root@computer1 ~]#_
```

Note that the new hostname isn't shown in your shell prompt until a new shell is started. To configure the host name shown in the preceding output at system startup, add the desired hostname to the /etc/hostname file, or run the **hostnamectl command**. For example, the commands in the following output set the host name to computer1.sampledomain.com and verify the configuration within /etc/hostname:

```
[root@computer1 ~]# hostnamectl set-hostname computer1.sampledomain.com
[root@computer1 ~]# cat /etc/hostname
computer1.sampledomain.com
[root@computer1 ~]#_
```

Note 25

Many network services record the system host name in their configuration files during installation. As a result, most Linux server distributions prompt you for the host name during the Linux installation to ensure that you don't need to modify network service configuration files afterwards.

Although host names are easier to use when specifying computers on the network, IP cannot use them to identify computers. Thus, you must map host names to their associated IP addresses so that applications that contact other computers across the network can find the appropriate IP address for a host name.

The simplest method for mapping host names to IP addresses is by placing entries into the /etc/hosts file, as shown in the following example:

```
[root@server1 ~]# cat /etc/hosts
127.0.0.1   server1 server1.class.com localhost localhost.localdomain
::1         server1 server1.class.com localhost6 localhost6.localdomain6
3.0.0.2     ftp.sampledomain.com fileserver
10.3.0.1    alpha
[root@server1 ~]#_
```

Note 26

You can also use the `getent hosts` command to view the contents of the /etc/hosts file.

The entries in the preceding output identify the local computer, 127.0.0.1, by the host names server1, server1.class.com, localhost, and localhost.localdomain. Similarly, you can use the host name ftp.sampledomain.com or fileserver to refer to the computer with the IP address of 3.0.0.2. Also, the computer with the IP address of 10.3.0.1 can be referred to using the name alpha.

Because it would be cumbersome to list names for all hosts on the Internet in the /etc/hosts file, ISPs can list FQDNs in DNS servers on the Internet. Applications can then ask DNS servers for the IP address associated with a certain FQDN. To configure your system to resolve names to IP addresses by contacting a DNS server, you can specify the IP address of the DNS server in the /etc/resolv.conf file. This file can contain up to three DNS servers; if the first DNS server is unavailable, the system attempts to contact the second DNS server, followed by the third DNS server listed in the file. An example /etc/resolv.conf file is shown in the following output:

```
[root@server1 ~]# cat /etc/resolv.conf
nameserver 209.121.197.2
nameserver 192.139.188.144
nameserver 6.0.4.211
[root@server1 ~]#_
```

On Linux distributions that use Systemd, the **Systemd-resolved** service handles name resolution requests. In this case, /etc/resolv.conf is merely a symlink to /run/systemd/resolve/stub-resolv.conf, which contains a single entry that redirects name resolution requests to the special 127.0.0.53 loopback address as shown below:

```
[root@server1 ~]# ls -l /etc/resolv.conf
lrwxrwxrwx. 1 root root 39 Sep 20 11:33 /etc/resolv.conf ->
../run/systemd/resolve/stub-resolv.conf
[root@server1 ~]# cat /run/systemd/resolve/stub-resolv.conf
nameserver 127.0.0.53
[root@server1 ~]#_
```

The Systemd-resolved service listens to name resolution requests sent to 127.0.0.53. It then queries the DNS servers configured by the network renderer (NetworkManager or Systemd-networkd) to resolve the name and responds to the original request with the answer. Systemd-networkd caches the results of names resolved by DNS servers to speed future queries. You can manage Systemd-resolved using the **`resolvectl` command**. For example, `resolvectl dns eth0 8.8.8.8` will configure eth0 to use a DNS server of 8.8.8.8, overriding the configuration provided by the network renderer. You can also use `resolvectl status` to display the current Systemd-resolved configuration, as well as `resolvectl flush-caches` to clear cached results.

Note 27

To test the DNS configuration by resolving a host name or FQDN to an IP address, you can supply the host name or FQDN as an argument to the `resolvectl query`, **`nslookup`**, **`dig`**, or **`host` command** at a command prompt.

When you specify a host name while using a certain application, that application must then resolve that host name to the appropriate IP address by searching either the local /etc/hosts file or a DNS server. The method that applications use to resolve host names is determined by the "hosts:" line in the /etc/nsswitch.conf file; an example of this file is shown in the following output:

```
[root@server1 ~]# grep hosts /etc/nsswitch.conf
hosts:      files dns
[root@server1 ~]#_
```

The preceding output indicates that applications first try to resolve host names using the /etc/hosts file (`files`). If unsuccessful, applications contact the DNS servers listed in the /etc/resolv.conf file (`dns`).

Legacy Linux systems used the /etc/host.conf file instead of /etc/nsswitch.conf. The /etc/host.conf file still exists on modern Linux systems to support older programs and should contain the same name resolution order as /etc/nsswitch.conf if older programs are installed. An example /etc/host.conf file that tells applications to search the /etc/hosts file (`hosts`) followed by DNS servers (`bind`) is shown in the following output:

```
[root@server1 ~]# cat /etc/host.conf
multi on
order hosts,bind
[root@server1 ~]#_
```

Routing

Every computer on a network maintains a list of IP networks so that packets are sent to the appropriate location; this list is called a **route table** and is stored in system memory. To see the route table, you can use the **`route` command** if the net-tools package is installed on your system, or the `ip route` command (a shortcut to `ip route show`) if your system has the iproute or iproute2 package. The following illustrates sample output from these commands:

```
[root@server1 ~]# 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 eth0
192.168.1.0     0.0.0.0         255.255.255.0   U     100    0        0 eth0
10.0.0.0        0.0.0.0         255.255.255.0   U     101    0        0 wlan0
[root@server1 ~]# ip route
```

```
default via 192.168.1.1 dev eth0 proto src 192.168.1.105 metric 100
192.168.1.0/24 dev eth0 proto kernel src 192.168.1.105 metric 100
10.0.0.0/8 dev wlan0 proto kernel src 10.0.0.5 metric 101
[root@server1 ~]#_
```

Note 28

The `netstat -r` command is equivalent to the `route` command.

Note 29

The /etc/networks file contains aliases for IP networks. The "default" line in the previous output refers to the 0.0.0.0 network because the /etc/networks file provides an alias for the 0.0.0.0 network called "default." To view the route table without aliases, supply the `-n` option to the `route` or `netstat -r` command, or the `-N` option to the `ip route` command.

The route tables shown in the preceding output indicates that all packets destined for the 10.0.0.0 network will be sent to the wlan0 network interface (which has an IPv4 address of 10.0.0.5). Similarly, all packets destined for the 192.168.1.0 network will be sent to the eth0 network interface (which has an IPv4 address of 192.168.1.105). Packets that must be sent to any other network will be sent to the default gateway (which has an IPv4 address of 192.168.1.1) via the eth0 network interface.

Note 30

The default gateway is normally specified within the IP configuration file for the network interface as shown earlier in this chapter and loaded when the network interface is activated.

If your computer has more than one network interface configured, the route table will have more entries that define the available IP networks; computers that have more than one network interface are called **multihomed hosts**. Multihomed hosts can be configured to forward packets from one interface to another to aid a packet in reaching its destination; this process is commonly called **routing** or **IP forwarding**. To enable routing on your Linux computer, place the number 1 in the file /proc/sys/net/ipv4/ip_forward for IPv4 or /proc/sys/net/ipv6/conf/all/forwarding for IPv6, as shown in the following output:

```
[root@server1 ~]# cat /proc/sys/net/ipv4/ip_forward
0
[root@server1 ~]# cat /proc/sys/net/ipv6/conf/all/forwarding
0
[root@server1 ~]# echo 1 > /proc/sys/net/ipv4/ip_forward
[root@server1 ~]# echo 1 > /proc/sys/net/ipv6/conf/all/forwarding
[root@server1 ~]# cat /proc/sys/net/ipv4/ip_forward
1
[root@server1 ~]# cat /proc/sys/net/ipv6/conf/all/forwarding
1
[root@server1 ~]#_
```

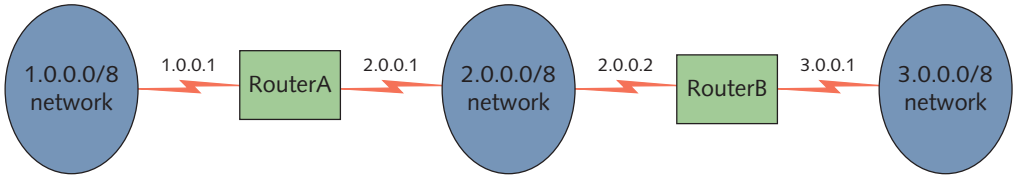
Note 31

The **`sysctl` command** can also be used to modify the contents of files under the /proc/sys directory. For example, `sysctl net.ipv4.ip_forward=1` would be equivalent to the `echo 1 > /proc/sys/net/ipv4/ip_forward` command.

To enable IPv4 routing at every boot, ensure that the line `net.ipv4.ip_forward = 1` exists in the /etc/sysctl.conf file or within a file under the /etc/sysctl.d directory. To enable IPv6 routing at every boot, ensure that the line `net.ipv6.conf.default.forwarding = 1` exists in the /etc/sysctl.conf file or within a file under the /etc/sysctl.d directory.

If your computer has more than one network interface and routing is enabled, your computer will route packets only to networks for which it has a network interface. On larger networks, however, you might have several routers, in which case packets might have to travel through several routers to reach their destination. Because routers only know the networks to which they are directly connected, you might need to add entries to the route table on a router so that it knows where to send packets that are destined for a remote network. Suppose, for example, your organization has three IPv4 networks (1.0.0.0/8, 2.0.0.0/8, and 3.0.0.0/8) divided by two routers, as shown in Figure 12-7.

Figure 12-7 A sample routed network



By default, RouterA will have two entries in its route table that are established when IP is configured. The first entry will identify that it is connected to the 1.0.0.0/8 network via the network interface that has the IP address 1.0.0.1, and the second entry will identify that it is connected to the 2.0.0.0/8 network via the network interface that has the IP address 2.0.0.1. If RouterA receives a packet that is destined for the 3.0.0.0/8 network, it does not know where to forward it because it does not have a route for the 3.0.0.0/8 network in its routing table. Thus, you must add a route to the route table on RouterA that allows it to forward packets destined for the 3.0.0.0/8 network to Router2 (2.0.0.2). To add this route on RouterA, you can run either the `route add -net 3.0.0.0 netmask 255.0.0.0 gw 2.0.0.2` command or the `ip route add 3.0.0.0/8 via 2.0.0.2` command.

Similarly, for RouterB to forward packets it receives destined for the 1.0.0.0/8 network, it must have a route that sends those packets to RouterA via the interface 2.0.0.1. To add this route on RouterB, you can run either the `route add -net 1.0.0.0 netmask 255.0.0.0 gw 2.0.0.1` command or the `ip route add 1.0.0.0/8 via 2.0.0.1` command.

Note 32

You can use the `route del <route>` or `ip route del <route>` command to remove entries from the route table.

Note 33

The contents of the route table are lost when the computer is powered off; to load the additional routes to the route table at every boot time, you must add the appropriate lines to the IP configuration file for the network interface as shown earlier in this chapter. For example, to add a single additional route to the 1.0.0.0/8 network via the 2.0.0.1 router on a Fedora system, you could add the line `route1=1.0.0.0/8,2.0.0.1` to the `[ipv4]` section of the /etc/NetworkManager/system-connections/Wired1.nmconnection file. For an Ubuntu system, you can instead add the following lines under the network interface section of the YAML file under the /etc/netplan directory:

```
routes:
   - to: 1.0.0.0/8
     via: 2.0.0.1
```

Note 34

You can also use a routing protocol on routers within your network to automate the addition of routes to the routing table. Two common routing protocols are Routing Information Protocol (RIP) and Open Shortest Path First (OSPF). If you install the FRRouting (frr) package, you can configure the RIP and OSPF routing protocols using the **vtysh command**.

Because the list of all routes on large networks such as the Internet is too large to be stored in a route table on a router, most routers are configured with a default gateway. Any packets that are addressed to a destination that is not listed in the route table are sent to the default gateway, which is a router that can forward the packet to the appropriate network or to the router's own default gateway and so on until the packets reach their destination.

If computers on your network are unable to connect to other computers on a remote network, the problem is likely routing related. A common utility used to troubleshoot routing is the **traceroute command**; it displays all routers between the current computer and a remote computer. To trace the path from the local computer to the computer with the IP address 3.4.5.6, you can use the following command:

```
[root@server1 ~]# traceroute 3.4.5.6
traceroute to 3.4.5.6 (3.4.5.6), 30 hops max, 38 byte packets
1  linksys (192.168.0.1)  2.048 ms  0.560 ms  0.489 ms
2  apban.pso.com (7.43.111.2)  2.560 ms  0.660 ms  0.429 ms
3  tfs.ihtfcid.net (3.0.0.1)  3.521 ms  0.513 ms  0.499 ms
4  sr1.lala.com (3.4.5.6)  5.028 ms  0.710 ms  0.554 ms
[root@server1 ~]#_
```

Note 35

Two common alternatives to the `traceroute` command include the **tracepath command** and the **mtr command**.

Note 36

To trace an IPv6 route, you can use the `mtr`, **traceroute6**, or **tracepath6 command**.

Network Services

Recall from Chapter 1 that Linux provides a wide variety of services that are available to users across a network. Before you can configure the appropriate network services to meet your organization's needs, you must first identify the types and features of network services.

Network services are processes that provide some type of valuable service for other computers on the network. They are often represented by a series of daemon processes that listen for certain requests on the network. Daemons identify the packets to which they should respond using a **port** number that uniquely identifies each network service. Different daemons listen for different port numbers. A port number is like an apartment number for the delivery of mail. The network ID of the IP address ensures that the packet is delivered to the correct street (network); the host ID ensures that the packet is delivered to the correct building (host); and the port number ensures that the packet is delivered to the proper apartment in the building (service).

Ports and their associated protocols are defined in the /etc/services file. To see to which port the telnet daemon listens, you can use the following command:

```
[root@server1 ~]# grep telnet /etc/services
telnet          23/tcp
telnet          23/udp
rtelnet         107/tcp             # Remote Telnet
rtelnet         107/udp
telnets         992/tcp
telnets         992/udp
skytelnet       1618/tcp            # skytelnet
skytelnet       1618/udp            # skytelnet
hp-3000-telnet  2564/tcp            # HP 3000 NS/VT block mode telnet
hp-3000-telnet  2564/udp            # HP 3000 NS/VT block mode telnet
tl1-telnet      3083/tcp            # TL1-TELNET
tl1-telnet      3083/udp            # TL1-TELNET
telnetcpcd      3696/tcp            # Telnet Com Port Control
telnetcpcd      3696/udp            # Telnet Com Port Control
scpi-telnet     5024/tcp            # SCPI-TELNET
scpi-telnet     5024/udp            # SCPI-TELNET
ktelnet         6623/tcp            # Kerberos V5 Telnet
ktelnet         6623/udp            # Kerberos V5 Telnet
[root@server1 ~]#_
```

The preceding output indicates that the telnet daemon listens on port 23 using both TCP/IP and UDP/IP.

Ports range in number from 0 to 65534. The ports 0–1023 are called **well-known ports** because they represent commonly used services. Table 12-2 provides a list of common well-known ports.

Table 12-2 Common well-known ports

Service	Port
FTP	TCP 20, 21
Secure Shell (SSH)	TCP 22
Telnet	TCP 23
SMTP	TCP 25
HTTP/HTTPS	TCP 80/TCP 443
rlogin	TCP 513
DNS	TCP 53, UDP 53
Trivial FTP (TFTP)	UDP 69
NNTP/NNTPS	TCP 119/TCP 563
POP3/POP3S	TCP 110/TCP 995
IMAP4/IMAP4S	TCP 143/TCP 993
NTP	TCP 123, UDP 123
SNMP	TCP 161, TCP 162, UDP 161, UDP 162
NetBIOS	TCP 139, UDP 139
SMB/CIFS	TCP 445, UDP 445
Syslog	TCP 514, UDP 514
LDAP/LDAPS	TCP 389, UDP 389/TCP 636, UDP 636

Note 37

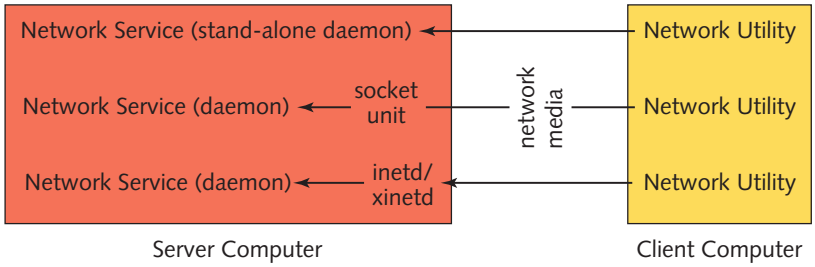
Many protocols have a secure version that uses encrypted communication. For example, secure HTTP is called HTTPS and uses a different port number as a result.

Note 38

You can use the `netstat` or **`ss` (socket statistics) command** to display active TCP/IP and UDP/IP connections on your system; the `netstat -t` or `ss -t` command will display active TCP/IP connections, whereas the `netstat -u` or `ss -u` command will display active UDP/IP connections.

Network utilities can connect to daemons that provide network services directly; these daemons are called **stand-alone daemons**. Alternatively, network utilities can connect to network services via the **Internet Super Daemon (inetd)** or the **Extended Internet Super Daemon (xinetd)**, which starts the appropriate daemon to provide the network service as needed. Because inetd and xinetd only start daemons on demand to conserve memory and processor cycles, they were more common on legacy Linux systems with few system resources. However, you may still find inetd or xinetd on smaller Linux systems today, such as IoT devices. Modern Linux systems with Systemd use **socket units** to provide the same functionality. Socket units are similar to service units but start daemons only when packets are received from other computers that request the associated port number. This structure is shown in Figure 12-8.

Figure 12-8 Interacting with network services



The inetd daemon is configured via entries within the /etc/inetd.conf file. For example, if you install the telnet daemon on a system that uses inetd, you can configure the following line in /etc/inetd.conf to start it:

```
[root@server1 ~]# grep telnet /etc/inetd.conf
telnet stream tcp nowait telnetd /usr/sbin/in.telnetd
[root@server1 ~]#_
```

The xinetd daemon is configured via entries within the /etc/xinetd.conf file. Normally, this file incorporates all of the files in the /etc/xinetd.d directory as well. Most daemons that are managed by xinetd are configured by files in the /etc/xinetd.d directory named after the daemons. For example, if you install the telnet daemon on a system that uses xinetd, you can configure it to be started via the /etc/xinetd.d/telnet file, as shown in the following output:

```
[root@server1 ~]# cat /etc/xinetd.d/telnet
service telnet
{
   flags           = REUSE
   socket_type     = stream
   wait            = no
   user            = root
   server          = /usr/sbin/in.telnetd
```

```
    log_on_failure  += USERID
    disable         = no
}
[root@server1 ~]#_
```

Many network daemons also have one or more configuration files that control how they operate. Most of these configuration files contain many commented lines that indicate the usage of certain configuration parameters. As a result, these configuration files can be very large; the main configuration file used by the Apache Web Server is often several hundred lines long. In addition to this, most stand-alone network daemons do not use the system log daemon (rsyslogd) or Systemd journal daemon (journald) to log information related to their operation. Instead, they log this information themselves to subdirectories of the same name under the /var/log directory. For example, log files for the Samba daemon are located under the /var/log/samba directory.

Table 12-3 lists the names and features of network services that are commonly found on Linux computers that participate in a network environment. You'll learn how to configure many of these network services in this chapter as well as within Chapter 13.

Table 12-3 Common network services

Network Service	Type	Port	Description
Apache Web Server (httpd)	stand-alone	TCP 80 TCP 443	Serves webpages using HTTP/HTTPS to other computers on the network that have a web browser Configuration file: /etc/httpd/conf/httpd.conf or /etc/apache2/apache2.conf
BIND/DNS Server (named)	stand-alone	TCP 53 UDP 53	Resolves fully qualified domain names to IP addresses for a certain namespace on the Internet Configuration file: /etc/named.conf
DHCP Server (dhcpd)	stand-alone	UDP 67 UDP 68	Provides IP configuration for computers on a network Configuration file: /etc/dhcp/dhcpd.conf
Washington University FTP Server (in.ftpd)	inetd/xinetd or socket unit	TCP 20 TCP 21 UDP 69	Transfers files to and accepts files from other computers on the network with an FTP utility Configuration file: /etc/ftpaccess Hosts denied FTP access: /etc/ftphosts Users denied FTP access: /etc/ftpusers FTP data compression: /etc/ftpconversions
Very Secure FTP Server (vsftpd)	stand-alone	TCP 20 TCP 21 UDP 69	Transfers files to and accepts files from other computers on the network with an FTP utility Configuration file: /etc/vsftpd/vsftpd.conf or /etc/vsftpd.conf Users denied FTP access: /etc/vsftpd/ftpusers or /etc/ftpusers
Internetwork News Server (innd)	stand-alone	TCP 119 TCP 563	Accepts and manages newsgroup postings and transfers them to other news servers Configuration file: /etc/news/inn.conf
NFS Server (rpc.nfsd)	stand-alone	TCP 2049	Shares files to other computers on the network that have an NFS client utility Configuration file: /etc/exports
POP3 Server (ipop3d)	stand-alone	TCP 110 TCP 995	Allows users with an email reader to obtain email from the server using the Post Office Protocol version 3

(*continues*)

Table 12-3 Common network services (*continued*)

Network Service	Type	Port	Description
IMAP4 Server (imapd)	stand-alone	TCP 143 TCP 993	Allows users with an email reader to obtain email from the server using the Internet Message Access Protocol
Sendmail Email Server (sendmail)	stand-alone	TCP 25	Accepts and sends email to users or other email servers on the Internet using the Simple Mail Transfer Protocol (SMTP) Configuration file: /etc/sendmail.cf
Postfix Email Server (postfix)	stand-alone	TCP 25	Accepts and sends email to users or other email servers on the Internet using the Simple Mail Transfer Protocol (SMTP) Configuration file: /etc/postfix/main.cf
rlogin Daemon (in.rlogind)	inetd/xinetd or socket unit	TCP 513	Allows users who use the `rlogin` and `rcp` utilities the ability to copy files and obtain shells on other computers on the network
rsh Daemon (in.rshd)	inetd/xinetd or socket unit	TCP 514	Allows users who use the `rsh` utility the ability to run commands on other computers on the network
Samba Server (smbd & nmbd)	stand-alone	TCP 137 TCP 138 TCP 139 TCP 445	Allows Windows users to view shared files and printers on a Linux server Configuration file: /etc/samba/smb.conf
Secure Shell Daemon (sshd)	stand-alone	TCP 22	Provides a secure alternative to the `telnet`, `rlogin`, and `rsh` utilities by using encrypted communication Configuration file: /etc/ssh/sshd_config
Squid Proxy Server (squid)	stand-alone	TCP 3128	Allows computers on a network to share one connection to the Internet. It is also known as a proxy server Configuration file: /etc/squid/squid.conf
telnet Daemon (in.telnetd)	inetd/xinetd or socket unit	TCP 23	Allows users who have a telnet utility the ability to log in to the system from across the network and obtain a shell
X.org (X11)	stand-alone	TCP 6000	Allows users who use X.org to obtain graphics from another X.org computer across the network using the XDMCP protocol. You can specify the hosts that can connect to X.org using the `xhost` command. Alternatively, you can use the `xauth` command to generate credentials (stored in ~/.Xauthority) that are used to connect to a remote X.org server.

To ensure that a service is responding to client requests, you can use the **`ncat` (net cat) command** to interact with it. For example, to interact with the sshd service running on port 22 on the local computer (127.0.0.1), you could run the `ncat 127.0.0.1 22` command and view the output. If the service doesn't display any output, you can often restart the daemon (sshd) to fix the problem. The `ncat` command can be used on two separate computers to communicate across a network on any port. For example, if you run `ncat -l -p 1234 > destination` on server1 and then run `ncat -w 3 server1 1234 < source` on server2, the source file on server2 will be copied to a new file called destination on server1 using port 1234.

Note 39

Different distributions often have different names for the `ncat` command. As a result, most Linux distributions create a symlink to their version of the `ncat` command called `nc`.

Remote Administration

As we discussed in Chapter 6, Linux servers are typically installed on a rackmount server system that is located on a rack in a server room and administered remotely. There are several ways to perform command-line and graphical administration of remote Linux servers, including telnet, remote commands, Secure Shell (SSH), and Virtual Network Computing (VNC).

Telnet

The easiest way to perform administration on a remote Linux computer is via a command-line interface. The **`telnet` command** has traditionally been used on Linux, macOS, and UNIX systems to obtain a command-line shell on remote servers across the network that run a telnet server daemon. For Windows systems, you can download the free **Putty** program at www.chiark.greenend.org.uk/~sgtatham/putty/ to start a telnet session to another computer.

The `telnet` command and telnet server daemon are not installed by default on most modern Linux distributions but can easily be installed from a software repository. On Linux systems that use Systemd, the telnet server daemon is usually managed using a socket unit (telnet.socket). On Linux systems that use SysV init, the telnet server daemon is managed by inetd or xinetd.

After the telnet daemon has been configured, you can connect to it from a remote computer. To do this, specify the host name or IP address of the target computer to the `telnet` command and log in with the appropriate user name and password. A shell obtained during a telnet session runs on a pseudo terminal (a terminal that does not obtain input directly from the computer keyboard) rather than a local terminal, and it works much the same way a normal shell does; you can execute commands and use the `exit` command to kill the shell and end the session. A sample telnet session is shown in the following output using a computer with a host name of server1:

```
[root@server1 ~]# telnet server1
Trying 192.168.1.105...
Connected to server1.
Escape character is '^]'.

Kernel 5.19.11-200.fc36.x86_64 on an x86_64 (1)
server1 login: root
Password: LINUXrocks!
Last login: Fri Oct 4 16:55:33 from 192.168.1.113
[root@server1 ~]# who
root     tty5         2023-10-09 14:23
root     pts/0        2023-10-09 18:49 (192.168.1.113)
[root@server1 ~]# exit
logout
Connection closed by foreign host.
[root@server1 ~]#_
```

Secure Shell (SSH)

Although the `telnet` command can be quickly used to perform remote administration, it doesn't encrypt the information that passes between computers. **Secure Shell (SSH)** was designed as a secure replacement for telnet (and other legacy commands, such as `rsh`, `rlogin`, and `rcp`) that encrypts information that passes across the network. As a result, the SSH daemon (sshd) is installed by default on most Linux distributions.

Note 40

On Linux workstation distributions, sshd is often installed by default but not set to start automatically at system initialization.

To connect to a remote Linux computer running sshd, you can use the **ssh command** followed by the host name or IP address of the target computer. For example, you can connect to a computer with the host name of appserver using the `ssh appserver` command. Your local user name will be passed to the server automatically during the SSH request, and you will be prompted to supply the password for the same user on the target computer. If you need to log in using a different user name on the remote appserver computer, you can instead use the `ssh -l username appserver` command or the `ssh username@appserver` command. A sample `ssh` session is shown in the following output:

```
[root@server1 ~]# ssh root@appserver
root@appserver's password: LINUXrocks!
Last login: Sun Oct  9 19:36:42 2023 from 192.168.1.113
[root@appserver ~]# who
root     tty5         2023-10-09 14:23
root     pts/0        2023-10-09 19:36 (192.168.1.113)
[root@appserver root]# exit
logout
Connection to appserver closed.
[root@appserver ~]#_
```

Note 41

You can also use the Putty program on a Windows computer to connect to sshd running on a Linux, macOS, or UNIX computer.

SSH can also be used to transfer files between computers. For example, to transfer the /root/sample file on a remote computer called appserver to the /var directory on the local computer, you could run the following command:

```
[root@server1 ~]# ssh root@appserver cat /root/sample > /var/sample
root@appserver's password: LINUXrocks!
[root@server1 ~]#_
```

Similarly, to transfer the /root/sample file on the local computer to the /var directory on a remote computer called appserver, you could run the following command:

```
[root@server1 ~]# ssh root@appserver cat </root/sample ">" /var/sample
root@appserver's password: LINUXrocks!
[root@server1 ~]#_
```

Alternatively, you can use the **scp command** to copy files using SSH. For example, to transfer the /root/sample file on a remote computer called appserver to the /var directory on the local computer, you could run the following `scp` command:

```
[root@server1 ~]# scp root@appserver:/root/sample /var
root@appserver's password: LINUXrocks!
[root@server1 ~]#_
```

Similarly, to copy the /root/sample file to the /var directory on appserver, you could use the following `scp` command:

```
[root@server1 ~]# scp /root/sample root@appserver:/var
root@appserver's password: LINUXrocks!
[root@server1 ~]#_
```

Many Linux utilities have built-in support for SSH, including the **rsync command**, which can also be used to copy files to other computers. For example, to copy the /root/sample file

using rsync with SSH encryption to the /var directory on appserver, you could run the following rsync command:

```
[root@server1 ~]# rsync -e ssh /root/sample root@appserver:/var
root@appserver's password: LINUXrocks!
[root@server1 ~]#_
```

Alternatively, you could use SSH encryption to synchronize the contents of the /data directory with the /data directory on appserver using the following rsync command:

```
[root@server1 ~]# rsync -a -e ssh /data root@appserver:/data
root@appserver's password: LINUXrocks!
[root@server1 ~]#_
```

Although SSH is primarily used to perform command-line administration of remote systems, the -X option to the ssh command can be used to tunnel X Windows information through the SSH connection if you are using the ssh command within a desktop environment. For example, if you start a command-line terminal within a GNOME desktop and run the command ssh -X root@appserver, you will receive a command prompt where you can type commands as you would in any command-line SSH session. However, if you type a graphical command (e.g., firefox), you will execute the Firefox web browser on appserver across the SSH tunnel. All graphics, keystrokes, and mouse movement will be passed between X Windows on appserver and X Windows on the local system.

Understanding SSH Host and User Keys

SSH uses symmetric encryption to encrypt the data that is sent over the network but requires asymmetric encryption to securely communicate the symmetric encryption key between the two computers at the beginning of the SSH connection. This asymmetric encryption requires that each computer running sshd have a public key and private key for the various asymmetric encryption algorithms supported by SSH (DSA, RSA, ECDSA, and ED25519); these keys are called the **SSH host keys** and are stored in the /etc/ssh directory as shown in the following output:

```
[root@server1 ~]# ls /etc/ssh | grep key
ssh_host_dsa_key
ssh_host_dsa_key.pub
ssh_host_ecdsa_key
ssh_host_ecdsa_key.pub
ssh_host_ed25519_key
ssh_host_ed25519_key.pub
ssh_host_rsa_key
ssh_host_rsa_key.pub
[root@server1 ~]#_
```

In the previous output, the ssh_host_ecdsa_key file contains the ECDSA private key for the host, whereas the ssh_host_ecdsa_key.pub file contains the ECDSA public key for the host. At the beginning of the SSH connection, the two computers negotiate the strongest asymmetric encryption algorithm that is supported by both computers.

When you connect to a new computer for the first time using SSH, you will be prompted to accept the SSH public key for the target computer, which is stored in ~/.ssh/known_hosts for subsequent connections. If the target computer's encryption keys are regenerated, you will need to remove the old key from the ~/.ssh/known_hosts file before you connect again.

Note 42

You can regenerate the host keys used by sshd using the **ssh-keygen command**. For example, the ssh-keygen -f /etc/ssh/ssh_host_ecdsa_key -t ecdsa command regenerates the private and public keys stored in /etc/ssh/ssh_host_ecdsa_key and /etc/ssh/ssh_host_ecdsa_key.pub, respectively.

To make authenticating to remote SSH hosts easier, you can generate a public key and private key for your user account for use with SSH using the `ssh-keygen` command; these keys are called SSH user keys and are stored in the ~/.ssh directory. For example, the ~/.ssh/id_rsa file contains the RSA private key for your user, and the ~/.ssh/id_rsa.pub file contains the RSA public key for your user. SSH user keys can be used in place of a password when connecting to trusted computers. To do this, you use the **`ssh-copy-id` command** to copy your public key to the user account on each computer that you would like to connect to without supplying a password. The target computer will store your public key in the ~/.ssh/authorized_keys file for each user account you specified with the `ssh-copy-id` command. The following example generates SSH user keys for the root user on server1, copies them to the root user account on server2, and then connects to server2 as the root user without specifying a password:

```
[root@server1 ~]# ssh-keygen
Generating public/private rsa key pair.
Enter file in which to save the key (/root/.ssh/id_rsa):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /root/.ssh/id_rsa.
Your public key has been saved in /root/.ssh/id_rsa.pub.
The key fingerprint is:
SHA256:lZu0U5EsQWcM1dFWNHvYK07YB5D8uQhgp/XuRACP5PQ root@server1
The key's randomart image is:
+---[RSA 2048]----+
|       +..=B*o.=+|
|      +o++.*=..o=|
|      .o=E*.o.ooo|
|       . + Boo. o|
|        S O..+.o |
|           =o.o  |
|          o  .   |
|           .     |
|                 |
+----[SHA256]-----+
[root@server1 ~]# ls .ssh
id_rsa  id_rsa.pub  known_hosts
[root@server1 ~]# ssh-copy-id root@server2
/usr/bin/ssh-copy-id: INFO: Source of key(s) to be installed:
"/root/.ssh/id_rsa.pub"
/usr/bin/ssh-copy-id: INFO: attempting to log in with the new key(s), to filter out
any that are already installed
/usr/bin/ssh-copy-id: INFO: 1 key(s) remain to be installed -- if you are prompted now
it is to install the new keys
root@server2's password: **********

Number of key(s) added: 1

Now try logging into the machine, with:   "ssh 'root@server2'"
and check to make sure that only the key(s) you wanted were added.

[root@server1 ~]# ssh root@server2
Last login: Sun Oct 14 20:17:41 2023 from server1
[root@server2 ~]# ls .ssh
authorized_keys
[root@server2 ~]#_
```

Note from the `ssh-keygen` command in the previous output that no passphrase was supplied to protect the private key. If you supply a passphrase, then you will need to supply that passphrase each time you use the private key, including each time you connect to another computer using your SSH user keys. To prevent this, you can use the **`ssh-agent` command** to start the SSH agent process on your computer and run the **`ssh-add` command** to add your private key to this process (supplying your passphrase when prompted). The SSH agent process will then automatically supply your passphrase when your SSH user keys are used to connect to other computers.

Configuring SSH

You can configure the functionality of sshd by editing the /etc/ssh/sshd_config file. Most of this file is commented and should only be edited to change the default settings that sshd uses when servicing SSH clients. The most commonly changed options in this file are those that deal with authentication and encryption. For example, to allow the root user to log into SSH, you can modify the value of the `PermitRootLogin` line to `yes` within /etc/ssh/sshd_config and restart sshd.

Note 43

On most Linux distributions, root access via SSH is denied by default. This is considered good security practice because malicious software on the Internet often attempts to log in as the root user. In this case, you can access SSH as a regular user and then use the `su` command to switch to the root user to perform administrative tasks.

Recall that the data communicated across the network with SSH is encrypted using a symmetric encryption algorithm that is negotiated at the beginning of the SSH connection, after the SSH host keys have been exchanged. Each symmetric encryption algorithm differs in its method of encryption and the cryptography key lengths used to encrypt data; the longer the key length, the more difficult it is for malicious users to decode the data. The main types of symmetric encryption supported by sshd are as follows:

- Triple Data Encryption Standard (3DES), which encrypts blocks of data in three stages using a 168-bit key length
- Advanced Encryption Standard (AES), an improvement on 3DES encryption and is available in 128-, 192-, and 256-bit key lengths
- Blowfish, an encryption algorithm that is much faster than 3DES and can use keys up to 448 bits in length
- Carlisle Adams Stafford Tavares (CAST), a general-purpose encryption similar to 3DES that is commonly available using a 128-bit key length
- ARCfour, a fast encryption algorithm that operates on streams of data instead of blocks of data and uses variable-length keys up to 2048 bits in length

In addition, all of the aforementioned types of encryption except ARCfour typically use Cipher Block Chaining (CBC), which can be used to encrypt larger amounts of data.

Client computers can use a /etc/ssh/ssh_config or ~/ssh/ssh_config file to set SSH options for use with the `ssh` command, including the symmetric encryption types that can be used, because the SSH client is responsible for randomly generating the symmetric encryption key at the beginning of the SSH connection. However, the encryption types on the client computer must match those supported by the SSH server for a connection to be successful.

Virtual Network Computing (VNC)

Like the `-X` option of `ssh`, **Virtual Network Computing (VNC)** is another graphical option for administrating a Linux system remotely. After installing a VNC server daemon on a computer, other computers that run a VNC client can connect to the VNC server daemon across a network to obtain a full desktop environment. VNC uses a special platform-independent protocol called Remote FrameBuffer (RFB) to transfer graphics, mouse movements, and keystrokes across the network.

Note 44

VNC server software and client software exist for Linux, macOS, UNIX, and Windows systems. This allows you to use a single technology to obtain the desktop of all of the Linux, macOS, UNIX, and Windows systems on your network.

Note 45

Because X Windows is not installed on Ubuntu Server by default, we'll focus on the configuration of VNC on Fedora in this section. However, the steps are similar on other Linux distributions.

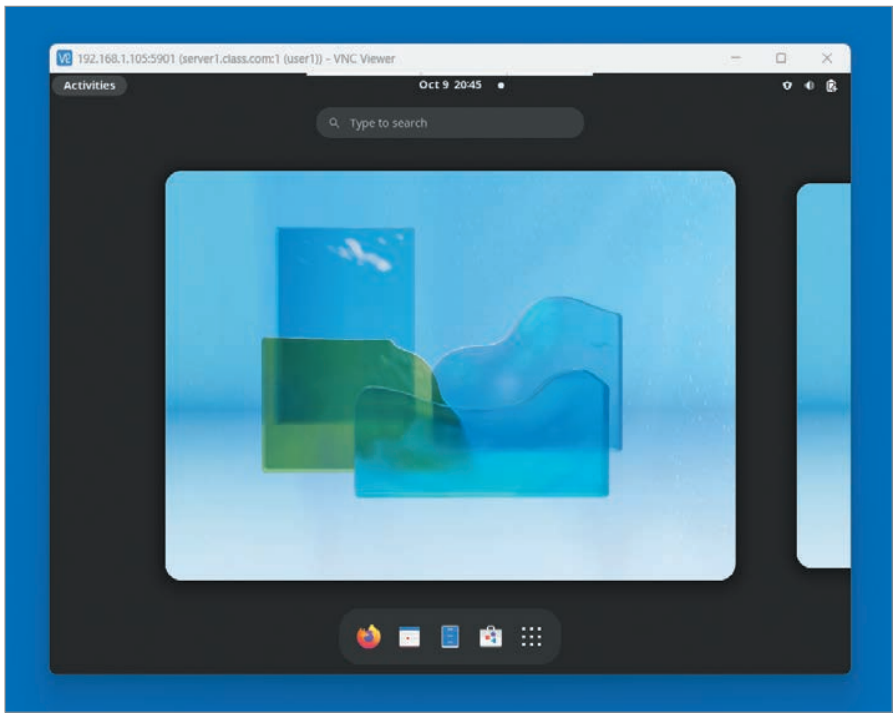
On Fedora, you can install a VNC server by running the `dnf install tigervnc-server` command. Next, you can set a VNC connection password for a user using the **`vncpasswd` command**. The VNC password is stored in the ~/.vnc/passwd file; for user1, the VNC password will be stored in the /home/user1/.vnc/passwd file. Finally, you can run the **`vncserver` command** as the user you specified the VNC password for; this will start a VNC server process with the next available display number, starting from 1.

Note 46

By default, the VNC server process listens on port 5900 + display number. For display number 1, the VNC server will listen on port 5901, for display number 2, the VNC server will listen on port 5902, and so on.

Other computers can then connect to the VNC server using a **VNC viewer** program, such as RealVNC. When using a VNC viewer program to connect to a remote VNC server, you can specify the server name or IP address, port, and display number using the syntax *`server:port:display`*. For example, to connect to the VNC server that uses display number 1 on the computer server1.class.com, you could use the syntax `server1.class.com:5901:1`. You will then obtain a desktop session on the remote computer, as shown in Figure 12-9.

Figure 12-9 A remote VNC session



Note 47

Many VNC viewers allow you to specify the server name or IP address and either the port or display number. For example, you could specify `server1.class.com:5901` or `server1.class.com:1` within a VNC viewer to connect to the server shown in Figure 12-9.

Note 48

You can use the remote or local port forwarding feature of SSH to encrypt the traffic sent by other services, such as VNC. For example, if you run the command `ssh -R 5901:server1.class.com:5901 bob@client1.class.com`, bob on the client1.class.com computer can start a VNC viewer and connect to client1.class.com:5901 in order to access the VNC server running on port 5901 on server1.class.com via SSH. Alternatively, you can run the `ssh -L 5901:localhost:5901 bob@server1.class.com` command on your computer to start an SSH session that forwards any local traffic on port 5901 to server1.class.com on port 5901 as bob. Following this, you can start a VNC viewer and connect to localhost:5901 to access the VNC server running on port 5901 on server1.class.com via SSH.

To configure a VNC server process to run persistently, you must edit the /etc/tigervnc/vncserver.users file and add a line to associate a user account to each X Windows display you created a service unit for. For example, the line `:1=bob` would associate the user bob to the first X windows display. Next, you must ensure that the user specified within the /etc/tigervnc/vncserver.users file has a VNC connection password set using the `vncpasswd` command. Finally, you must create and start a VNC Systemd service unit that specifies the display number. To create and start a VNC Systemd service unit that listens on the first X Windows display, you can use the following command:

```
[root@server1 ~]# systemctl enable vncserver@:1.service
Create symlink /etc/systemd/system/multi-user.target.wants/vncserver@:
1.service -> /usr/lib/systemd/system/vncserver@.service
[root@server1 ~]# systemctl start vncserver@:1.service
[root@server1 ~]#_
```

Note 49

You can also specify user-specific VNC configuration by uncommenting and modifying the appropriate lines within the ~/.vnc/config file.

Note 50

On systems that do not use the Systemd system initialization system, the /etc/sysconfig/vncservers configuration file is used in place of /lib/systemd/system/vncserver@:1.service.

Note 51

There are many alternatives to VNC that provide for remote access to Linux desktops, including Xrdp, which uses the Microsoft Remote Desktop Protocol, and NoMachine, which uses the proprietary NX protocol.

Summary

- A network is a collection of connected computers that share information.
- A protocol is a set of rules that define the format of information that is transmitted across a network. TCP/IP is the standard protocol used by the Internet and most networks.
- Each computer on an IP network must have a valid IPv4 or IPv6 address.
- The IPv4 configuration of a network interface can be specified manually, obtained automatically from a DHCP or BOOTP server, or autoconfigured by the system.
- The IPv6 configuration of a network interface can be specified manually, obtained automatically from a DHCP server or router using ICMPv6, or autoconfigured by the system.
- The /etc/NetworkManager/system-connections directory on a Fedora Workstation system and the /etc/netplan directory on an Ubuntu Server system contain the configuration files for network interfaces.
- Host names are computer names that, unlike IP addresses, are easy for humans to remember. Host names that are generated by the hierarchical Domain Name Space are called FQDNs.
- Host names must be resolved to an IP address before network communication can take place.
- Routers are devices that forward IP packets from one network to another. Each computer and router has a route table that it uses to determine how IP packets are forwarded.
- Network services listen for requests on a certain port number. They are normally started by a stand-alone daemon, or on demand using inetd, xinetd, or a Systemd socket unit.
- There are many ways to remotely administer a Linux system. You can perform command-line administration remotely via the `telnet` and `ssh` commands. For graphical remote administration, you can use the `ssh -X` command or VNC.

Key Terms

aggregation
ANDing
Automatic Private IP Addressing (APIPA)
bonding
bridging
broadcast
classless interdomain routing (CIDR) notation
default gateway
`dhclient` command
`dig` command
Ethernet
`ethtool` command
Extended Internet Super Daemon (xinetd)
`host` command
host ID
host name
`hostname` command
`hostnamectl` command
`ifconfig` (interface configuration) command
Internet Control Message Protocol (ICMP)
Internet Control Message Protocol version 6 (ICMPv6)
Internet Super Daemon (inetd)
`ip` command
IP forwarding
IP version 4 (IPv4)
IP version 6 (IPv6)
local area network (LAN)
Media Access Control (MAC) address
media access method
Modem Manager utility
`mtr` command
multicast
multihomed hosts
`ncat` (net cat) command
`netplan` command
`netstat` command
network
Network Connections utility
network ID
network renderer
network service
`networkctl` command
NetworkManager
`nm-connection-editor` command
`nmcli` command
`nslookup` command
octet
packet
`ping` (Packet Internet Groper) command
`ping6` command
Point-to-Point Protocol (PPP)
port
PPP over Ethernet (PPPoE)
`pppoeconf` command
protocol
Putty
Remote Direct Memory Access (RDMA)
`resolvectl` command
`route` command
route table
routing