CHAPTER

Forwarding IP Packets

9

What You Will Learn

In this chapter, you will learn how routers forward IP packets. We'll start with the logical steps a router follows to forward ("route") a packet out the next-hop interface. Then we'll look at router architectures to see how specialized devices (there are "software-only" routers) accomplish routing and forwarding.

Finally, you will learn about how IPv4 routers transition to handling IPv6 routing and various methods to *tunnel* IPv6 packets through links connected by IPv4-only routers. Tunnels were introduced in Chapters 3 and 4 and occur when the normal encapsulation sequence of packet–inside frame is violated in some fashion.

This chapter is really a continued investigation into many of the concepts introduced in the previous chapter. Figure 9.1 highlights the network components we'll be working with in this chapter.

The routers on our network are Juniper Networks routers. These routers have a different "look and feel" compared to other routers, most of which use a more "Cisco-like" interface and display. For example, the routing tables seem very long and detailed compared to Cisco routers' default displays.

```
admin@CE6> show route 10.10/16

inet.0: 34 destinations, 35 routes (34 active, 0 holddown, 0 hidden)
+ = Active Route, - = Last Active, * = Both

10.10.11.0/24      *[OSPF/10] 1w5d 18:25:05, metric 6
                    > via ge-0/0/3.0
10.10.12.0/24      *[Direct/0] 2w2d 00:15:44
                    > via fe-1/3/0.0
10.10.12.1/32      *[Local/0] 2w2d 00:15:44
                       Local via fe-1/3/0.0
```

We'll talk about the routing table entry marked Open Shortest Path First (OSPF) in Chapter 14. This route was learned by a routing protocol running between the routers on our network, and we'll see how OSPF is configured in a later chapter. Note that

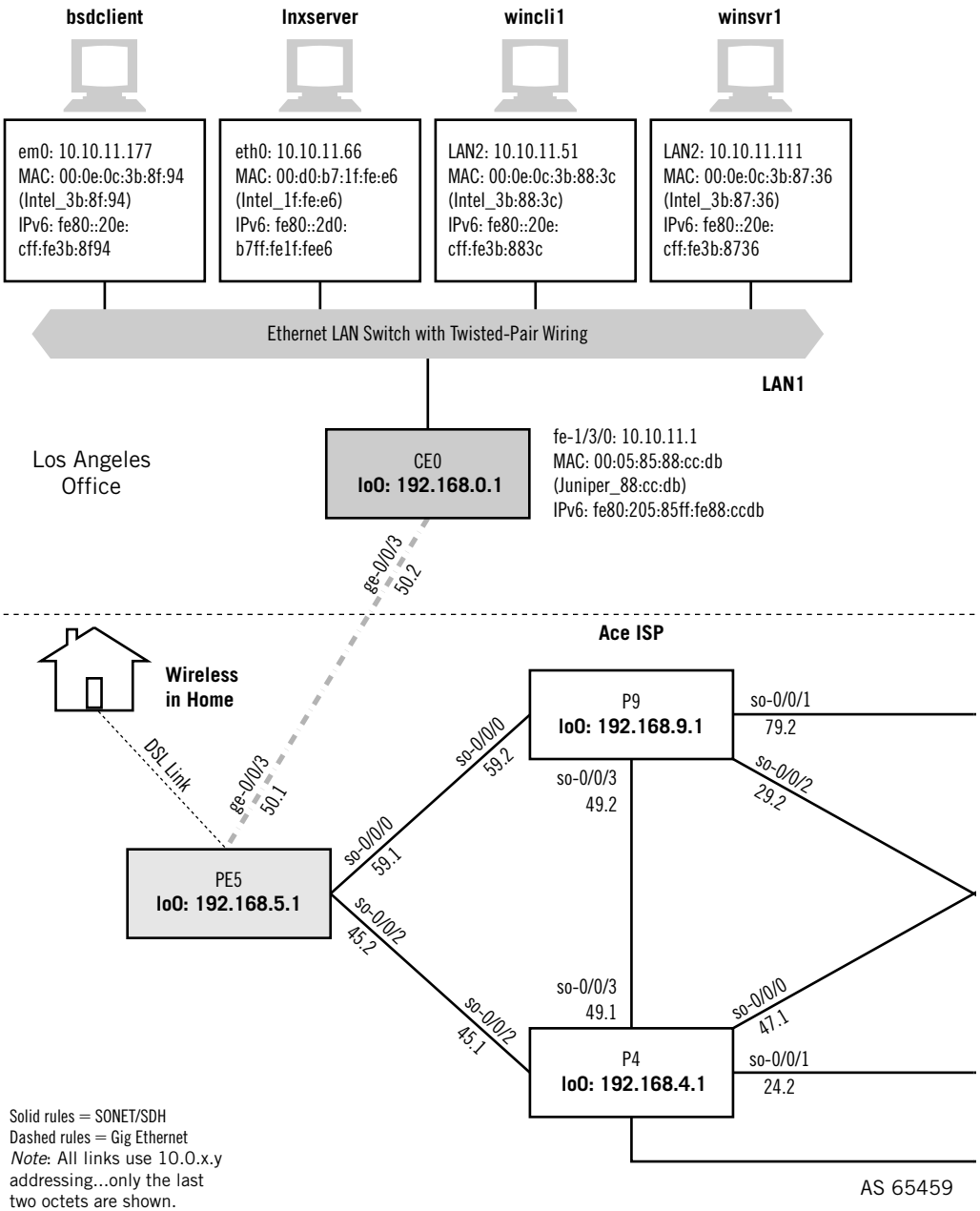
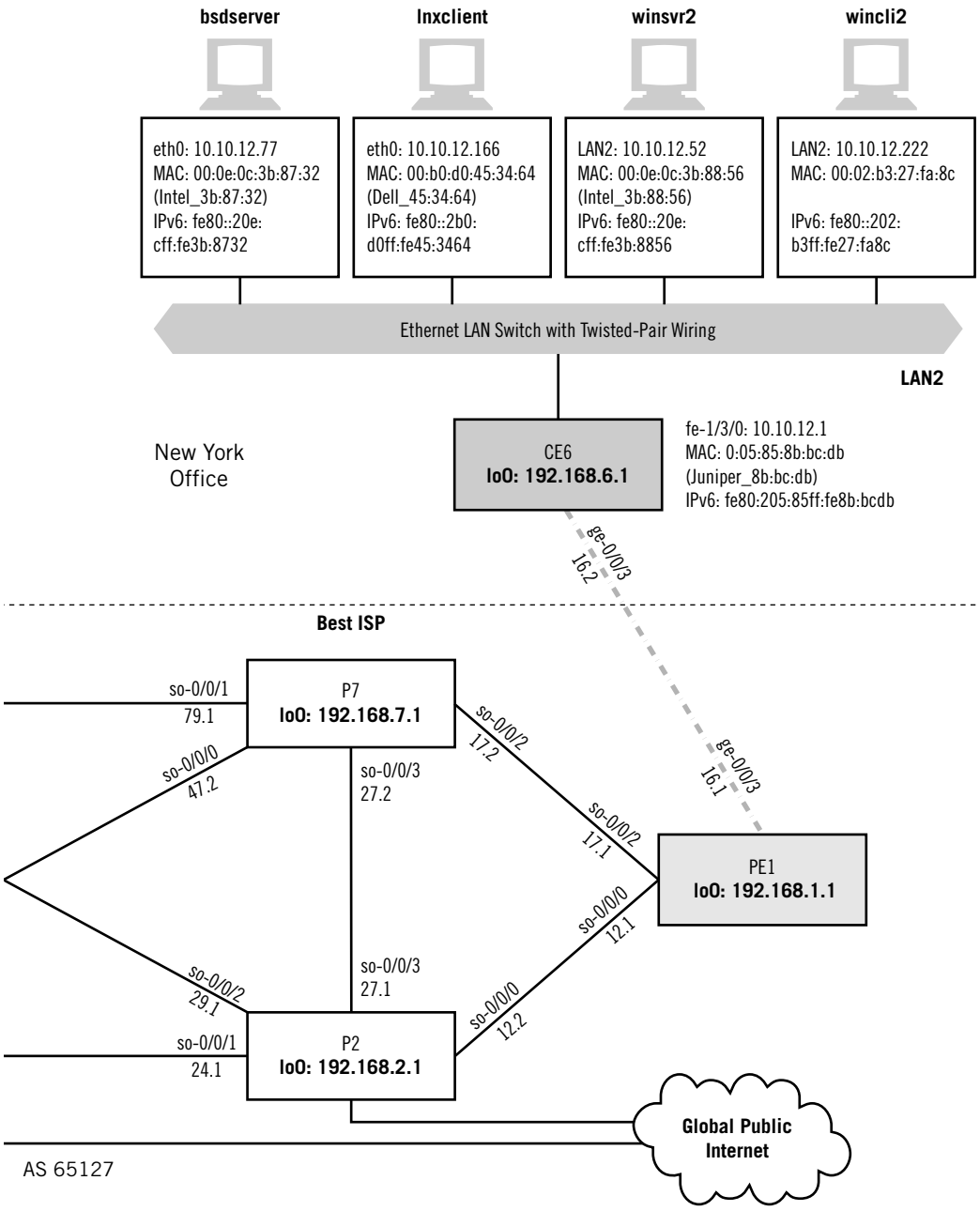


FIGURE 9.1

Forwarding packets across the network. Note that we'll be using the customer-edge routers `CE0` and `CE6` in this chapter.


bsdserver
lnxclient
winsvr2
wincli2
eth0: 10.10.12.77
MAC: 00:0e:0c:3b:87:32
(Intel_3b:87:32)
IPv6: fe80::20e:
cff:fe3b:8732
eth0: 10.10.12.166
MAC: 00:b0:d0:45:34:64
(Dell_45:34:64)
IPv6: fe80::2b0:
d0ff:fe45:3464
LAN2: 10.10.12.52
MAC: 00:0e:0c:3b:88:56
(Intel_3b:88:56)
IPv6: fe80::20e:
cff:fe3b:8856
LAN2: 10.10.12.222
MAC: 00:02:b3:27:fa:8c
IPv6: fe80::202:
b3ff:fe27:fa8c
Ethernet LAN Switch with Twisted-Pair Wiring
LAN2
New York
Office
CE6
lo0: 192.168.6.1
fe-1/3/0: 10.10.12.1
MAC: 0:05:85:8b:bc:db
(Juniper_8b:bc:db)
IPv6: fe80:205:85ff:fe8b:bcdb
ge-0/0/3
16.2
Best ISP
so-0/0/1
79.1
P7
lo0: 192.168.7.1
so-0/0/2
17.2
so-0/0/0
47.2
so-0/0/3
27.2
ge-0/0/3
16.1
so-0/0/2
17.1
PE1
lo0: 192.168.1.1
so-0/0/0
12.1
so-0/0/2
29.1
so-0/0/3
27.1
so-0/0/0
12.2
so-0/0/1
24.1
P2
lo0: 192.168.2.1
Global Public
Internet
AS 65127

the entry has a *preference* of 10 (which makes it more "costly" to use than direct/local interface routes [0] or static routes [5]). Traffic to destinations on LAN1 is sent to `PE1` over the `ge-0/0/3` interface. A preference is distinct from the metric or cost of a route itself; preference applies to routes learned in different ways.

We can make the routing table display more Cisco-like by using the `terse` option:

```
admin@CE6> show route 10.10/16 terse

inet.0: 34 destinations, 35 routes (34 active, 0 holddown, 0 hidden)
+ = Active Route, - = Last Active, * = Both

A Destination       P Prf    Metric 1     Metric 2     Next hop        AS path
* 10.10.11.0/24     O  10    6                         >ge-0/0/3.0
* 10.10.12.0/24     D   0                              >fe-1/3/0.0
* 10.10.12.1/32     L   0                              Local
```

The asterisk (*) means the route is active (used for forwarding), and the P field is for protocol. One metric is used (two are allowed), the next-hops are the same (thankfully!), and we'll talk about what an AS path is in the chapter on the BGP routing protocol.

Let's use `traceroute` to see which routers `CE6` uses to reach LAN1, attached to router `CE0` at interface `10.10.11.1`.

```
admin@CE6> traceroute 10.10.11.1

traceroute to  10.10.11.1  (10.10.11.1), 30 hops max, 40 byte packets
 1  10.0.16.1  (10.0.16.1)   0.743 ms  0.681 ms  0.573 ms
 2  10.0.12.2  (10.0.12.2)   0.646 ms  0.647 ms  0.620 ms
 3  10.0.24.2  (10.0.24.2)   0.656 ms  0.664 ms  0.632 ms
 4  10.0.45.2  (10.0.45.2)   0.690 ms  0.677 ms  0.695 ms
 5  10.10.11.1  (10.10.11.1)   0.846 ms  0.819 ms  0.775 ms
```

Each router handles the three-packet set generated by the source (`CE6`) in one of three ways:

1. If the packet is not for this router (the device does not have `10.10.11.1` configured locally), and the TTL is 1 or 0, then the router creates an ICMP Time-Exceeded message, sets the source address to the router's receiving interface address, sets the destination address to the source's, and sends the ICMP packet out the interface listed as the route back to the source in the forwarding table. This does not have to be the same as the receiving interface, but it usually is.
2. If the packet is not for this router and the TTL is not 1 or 0, then the router decrements the TTL field and forwards the packet out the interface leading to the next hop on the way to the destination address.
3. If the packet is for this router or device, then it sends back an ICMP Port Unreachable message.

Why a TTL of 1 or 0? Some routers decrement the TTL immediately and others only as part of the forwarding process, right before output queuing. This way both types of router handle the packet consistently.

When the source receives a Time-Exceeded message, it records the results of the round-trip time for the three packets, checks to see if it has a DNS entry for the IP address, and prints a line of output with a "hop" number and the rest of the statistics. When it receives a Port Unreachable message, the traceroute utility prints the final results and exits.

Because we don't yet have DNS running, all the IPv4 addresses are repeated twice. From the network diagram, we can see that the packets flowed from `CE6` to `PE1` (not surprisingly) at `10.0.16.1` and then through `P2` (`10.0.12.2`), `P4` (`10.0.24.2`), `PE5` (`10.0.45.2`) and on to `CE0` (`10.10.11.1`, the local interface target, is used instead of `10.0.50.2`). (We'll see what happens when one of the P routers or links between them fails in a later chapter.)

We have IPv6 running on the LANs and routers `CE0` and `CE6`. Let's see what happens on `CE6` when we ping the LAN1 interface address four times using the LAN2 interface IPv6 source address. Recall that the private ULA IPv6 addresses on LAN1 start with `fc00:ffb3:d5:a`.

```
admin@CE6> ping count 4 inet6 source fc00:fe67:d4:b:205:85ff:fe8b:bcdb
fc00:ffb3:d5:a:205:85ff:fe88:ccdb
PING6(56=40+8+8 bytes) fc00:fe67:d4:b:205:85ff:fe8b:bcdb –> fc00:ffb3:d5:
a:205:85ff:fe88:ccdb
–- fc00:ffb3:d5:a:205:85ff:fe88:ccdb ping6 statistics –-
4 packets transmitted, 0 packets received, 100% packet loss
```

What happened? Well, for one thing, we have no routes to any IPv6 addresses on LAN1 in the IPv6 routing table. And if they're not in the routing table, they won't be in the forwarding table.

```
admin@CE6> show route table inet6 fc00:ffb3:d5:a::/64

admin@CE6>
```

What can we do about this? Well, we could add some static routes to the IPv6 tables on each router, or we could run an IPv6 routing protocol between the routers to share the routing information (we'll do this in a later chapter). Or, we can configure an IPv6 over IPv4 tunnel between routers `CE6` and `CE0` (and back). We know we have connectivity with IPv4 between the edge routers, as shown with traceroute.

Here's how to configure an IPv6-over-IPv4 tunnel on routers `CE0` and `CE6`. It basically tells the router to take any traffic for LAN1 or LAN2 IPv6 addresses, put them inside IPv4 packets with the LAN IPv4 interface addresses, and send them out as if they were IPv4 packets. We'll apply the tunnels on a logical interface known as the Generic Routing Encapsulation (GRE) interfaces, abbreviated `gr-` on Juniper Networks routers. Only the final configuration statements are shown.

```
[edit interfaces gr-1/0/0]
admin@CE6# set interfaces gr-1/0/0
admin@CE6# set unit 0 tunnel source 10.10.12.1;
    /*source address on LAN2 interface*/
```

```
admin@CE6# set unit 0 tunnel destination 10.10.11.1;
    /*destination address on LAN1 interface*/
admin@CE6# set unit 0 family inet6 address fc00:ffb3::/32
    /*LAN1 addresses*/

[edit interfaces gr-1/0/0]
admin@CE0# set interfaces gr-1/0/0
admin@CE0# set unit 0 tunnel source 10.10.11.1;
    /*source address on LAN1 interface*/
admin@CE0# set unit 0 tunnel destination 10.10.12.1;
    /*destination address on LAN2 interface*/
admin@CE0# set unit 0 family inet6 address fc00:ffb3::/32
    /*LAN2 addresses*/
```

Now we should be able to ping and traceroute an IPv6 address on LAN1 (in this case, `fc00:ffb3:d5:a:20e:cff:fe3b:8f95` for `bsdclient`) from the customer-edge router on LAN2. And we can. Note that, because of the tunnel, the destination seems to be only two hops away.

```
admin@CE6> ping inet6 count 4 source fc00:fe67:d4:b:205:85ff:fe8b:bcdb
fc00:ffb3:d5:a:20e:cff:fe3b:8f95
PING6(56=40+8+8 bytes) fc00:fe67:d4:b:205:85ff:fe8b:bcdb –>
fc00:ffb3:d5:a:20e:cff:fe3b:8f95
16 bytes from fc00:fe67:d4:b:205:85ff:fe8b:bcdb, icmp_seq=0 hlim=64
time=0.900 ms
16 bytes from fc00:fe67:d4:b:205:85ff:fe8b:bcdb, icmp_seq=1 hlim=64
time=0.728 ms
16 bytes from fc00:fe67:d4:b:205:85ff:fe8b:bcdb, icmp_seq=2 hlim=64
time=0.856 ms
16 bytes from fc00:fe67:d4:b:205:85ff:fe8b:bcdb, icmp_seq=3 hlim=64
time=0.838 ms

admin@CE6> traceroute inet6 source fc00:fe67:d4:b:205:85ff:fe8b:bcdb
fc00:ffb3:d5:a:20e:cff:fe3b:8f95
traceroute6 to fc00:ffb3:d5:a:20e:cff:fe3b:8f95 (fc00:ffb3:d5:a:205:85ff:
fe88:ccdb) from fc00:fe67:d4:b:205:85ff:fe8b:bcdb, 30 hops max, 12 byte
packets
 1  fc00:ffb3:d4:b:205:85ff:fe88:ccdb (fc00:ffb3:d4:b:205:85ff:fe88:ccdb)
1.059 ms  0.979 ms  0.819 ms
 2  fc00:ffb3:d5:a:20e:cff:fe3b:8f95 (fc00:ffb3:d5:a:20e:cff:fe3b:8f95)
0.832 ms 0.887 ms  0.823 ms
```

Let's take a look at the some basic types of router architectures that can be used to implement these packet-forwarding strategies.

ROUTER ARCHITECTURES

There are three main steps that a router must follow to process and forward a packet to the next hop. Processing a packet means to check an incoming packet for errors and other parameters, looking up the destination address in a forwarding table to

determine the proper output port for the packet, and then sending the packet out on that port.

But how are the input ports connected to the output ports? In smaller routers, which can even be implemented on PC or laptop computers with two or more interfaces, software simply examines the packet headers and forwards the packets where they need to go. Windows PCs can do this, and often do on home networks. In Linux, there is a command to allow the "host" to forward packets without processing the content of the packet more fully.

```
[root@lnxserver admin]# echo "1" > /proc/sys/net/ipv4/ip_forward
```

Linux IP Forwarding

If you enter the `ip_forward` command from the shell command prompt, the setting is not "remembered" after a reboot. If the host is to function as a gateway as well as host, place the command in an initialization script.

Small routers, such as those for DSL or small-edge LANs, can allow the incoming packet to sit in a memory buffer somewhere and adjust header fields, perform tunnel encapsulation, and so on, and then queue the packet for output. Larger routers, such as those used by ISPs or on the Internet backbones, must route as fast as they can, usually at *wire speeds* (this means that the device processes data without reducing overall transmission speed, so even if the packets arrive as fast as the input line allows, under maximum load, there is minimal delay through the router).

Instead of *software-based* forwarding architectures, these larger routers use *hardware-based* forwarding fabric architectures. The differences are important, so we'll take a look at them in more detail.

Basic Router Architectures

When it comes to architecture, routers look very much like a PC. This was one of the reasons for the initial success of routers: Routers could be fabricated out of simple, off-the-shelf parts and did not require extensive or customized chipsets or hardware. So these routers have a CPU, memory, interfaces, peripheral ports—in short, usually everything but a hard drive. Small routers do not even have floppy drives or other forms of external storage. This makes sense: Routers don't need to store much of anything. A forwarding table needs to be in memory at all times, because it's much too slow to try and fetch a piece of the table off a hard drive when needed. A lot of routers boot themselves from special servers, and have nonvolatile random access memory (NVRAM) that keeps whatever information they need to remember whenever their power is cut or turned off. Volatile memory like normal RAM is always erased when power is lost, but NVRAM is like a disk.

The chief distinction is that at the heart of such routers is a general-purpose computer. The architecture for large modern routers does not have a "center."

Routers do not have to worry about adding cards for video, graphics, or other tasks either. The slots in the chassis just handle various types of networking interfaces such as Ethernet, ATM, SONET/SDH (Synchronous Optical Network/Synchronous Digital Hierarchy), or other types of point-to-point WAN links. Most interface modules have multiple ports, depending on the type of interface that they support. In a lot of high-end router models, the interface cards are complex devices all by themselves and often called *blades*. Interfaces usually can be added as needed for the networking environment—one or more LAN cards for the routers that handle customers and one or more WAN cards for connection to other routers. Backbone routers often have only WAN cards and no customers at all.

Another difference between a software-based router and a common PC is that PCs almost always have only a single CPU. Because of the central role of these chips in running all of the hardware and software on the computer, single-CPU architectures require very powerful CPU chips.

Some routers use a variety of CPU chips, and because the tasks are shared among the processors, these CPU chips do not have to be tremendously powerful either. Each CPU set is chosen to fit the mission of the router. They have enough horsepower for the home and small office, and these chips are stable, plentiful, and inexpensive.

Some routers use different types of memory. Figure 9.2 shows the general layout of the motherboard of a generic software-based router. Many router motherboards have four types of memory intended for specific purposes. Each type of memory and its location on the motherboard is shown in the figure. This architecture is also very similar to the network processor engine (NPE) for larger Cisco router architectures. A lot of architectures forgo packet memory because of the bandwidth available in their shared

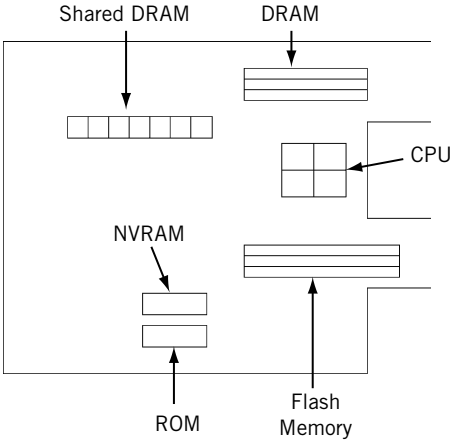


FIGURE 9.2

Software-based architecture for small routers, showing the various types of memory used.

memory architecture or because the CPU itself contains a dedicated packet handling architecture.

Every router ships with at least the factory default minimum of DRAM (dynamic random access memory) and flash memory, but more can be added in the factory or in the field. Generally, the DRAM can be doubled or increased fourfold, depending on model, and flash memory can be doubled.

RAM/DRAM is sometimes called *working storage* because in the days before hard drives and other types of external storage, memory was all that computers had for storing information outside of the immediate CPU. In a router, the RAM/DRAM performs the same functions for the router's CPU as the memory in a PC does for its CPU. So when the router is up and running, the RAM/DRAM contains an image of the operating system software, the running configuration (called *running-config* in routers using the Cisco configuration conventions) file, the routing table and associated tables built after startup, and the packet buffer. If this seems like a lot of work for one type of memory, this just shows the flexibility of function in a general-purpose architecture router.

The RAM acronym often used by router vendors is somewhat misleading. Almost all RAM in a router today is DRAM, since static memory—regular RAM—became obsolete some time ago. But people are used to the old RAM acronym, and it's included in a lot of literature just for familiarity.

In addition to the DRAM near the CPU, these types of routers include shared DRAM or shared memory. Also known as *packet memory*, the shared DRAM handles the packet buffers in the router. Splitting the packet buffers from the other DRAM improves I/O performance, because the shared DRAM is physically closer to the interfaces that handle the packets.

Nonvolatile RAM (NVRAM) is memory that retains information even when power is cut off to the router. Routers use NVRAM to store a copy of the router configuration file. Without NVRAM, the router would never be able to remember its proper configuration when it was restarted. NVRAM is where the startup configuration (called *startup-config* on routers using the Cisco configuration conventions) is stored.

Flash memory is another form of nonvolatile memory. But although flash memory is different from NVRAM, flash memory can also be erased and reprogrammed as needed. In many routers, flash memory is used to hold one or more copies of the router's operating system: In the case of Cisco, this is called the Internetwork Operating System, or IOS.

ROM is read-only memory and is therefore nonvolatile, but, as might be expected, ROM cannot be changed. Routers use ROM to hold what is called the *bootstrap* program. Normally, flash memory and NVRAM hold all of the information that the router needs to come up again properly with the current configuration after a shutdown or other power loss. But if there is a catastrophe, the bootstrap program in ROM can be used to boot the router into a minimum configuration. ROM used for this purpose is also called ROMMON (ROM monitor) and usually has a distinctive `rommon>>` prompt taken from early Unix systems. ROMMON at least gets the router to the point where simple commands can be typed in through a system console terminal (monitor). In smaller routers, ROM holds only a minimal subset of the router's operating system software. In larger routers, the ROM often holds a full copy of the router's operating system software.

Another Router Architecture

In contrast to the basic router architecture just explored, no one would accuse a large Internet backbone router of looking or acting like a PC. Routers based on a central CPU just about run out of gas once link speeds move into the multigigabit ranges with OC-48 (2.4 Gbps) and OC-192 (10 Gbps). And with 10 Gigabit Ethernet and OC-768 (40 Gbps), a change to the basic architecture of the router for the Internet backbone is necessary. Many Internet backbone routers share the same basic architecture, whether they come from Cisco or Juniper Networks or someone else. However, the terminology used for the components varies considerably from vendor to vendor. Because the Illustrated Network uses Juniper Networks routers as its network nodes, we'll use the Juniper Networks architecture and terminology in this section, but only as an example, not necessarily as an endorsement.

Larger network routers, oddly enough, do have hard drives. In fact, many Internet backbone routers have a complete PC built right in (some even have two PCs). But wait a minute. Isn't the PC architecture much too slow for heavy duty, "wire-speed" routing? And isn't a hard drive useless when it comes to routing because the forwarding table has to be in memory? Right on both counts. The PC in the backbone router, called the routing engine (RE) in Juniper Networks routers, does not forward packets at all. Packets are routed and forwarded by the packet-forwarding engine (PFE), which is where all the specialized ASICs are located. The RE controls the router, handles the routing protocols, and performs all of the other tasks that can be handled more leisurely than wire-speed packet transit traffic. Packets are forwarded from input to output port using the forwarding table (FT) in the hardware fabric.

The fundamental principle in large router design is the idea that the functions of a router can be split into two distinct parts: one portion for handling routing and control operations and another for forwarding packets. By separating these two operations, the router hardware can be designed and optimized to perform each function well.

This division of labor makes perfect sense. It has already been pointed out several times that no one really sends traffic to a router. The vast majority of packets just pass through the router. So transit packets never leave the hardware-based *fabric* linking input and output ports and control packets, such as those for the routing protocols, which only come along every few seconds or so, and can be handled as required by the RE.

Just like other routers, large backbone routers can handle various types of networking interfaces. But these routers are normally intended for mainly customer traffic aggregation or for an ISP backbone, although many corporations are attracted to edge-oriented routers with this architecture as well. And anywhere in an enterprise where there is a requirement for sustained 2-Gbps operation, routing is probably not being done in software.

The overall concept of the division between routing engine (routing protocol control and management) and packet-forwarding engine (line-rate routing transit traffic) with a hardware-based "switching" fabric is shown in Figure 9.3.

The section of the router that is designed to handle the general routing operations (and control-plane management tasks) is the RE. The RE is designed to handle all the routing protocols, user interaction, system management, and OAM&P (operations,

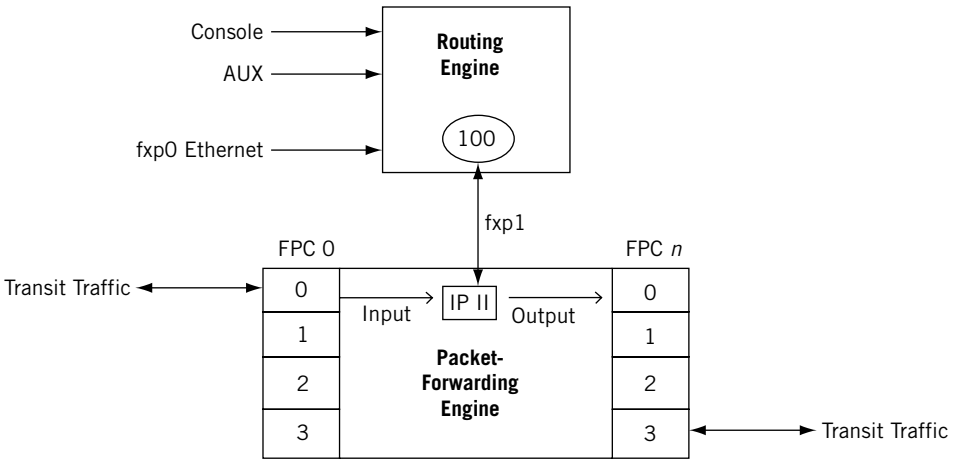


FIGURE 9.3

A hardware-based router with a switching fabric architecture. Note that the figure uses the architecture and terminology of Juniper Networks routers, which are used on the Illustrated Network.

administration, maintenance, and provisioning), and so on. The second section in Juniper Networks routers is the PFE, and is specifically designed to handle the forwarding of packets across the router from input to output interface. Transit packets never enter the routing engine at all.

The communications channel between the routing engine and the PFE is a standard 100-Mbps Fast Ethernet. This might seem somewhat surprising at first, because the interfaces on a Juniper Networks router can be many gigabits per second. But only control information needs to enter the routing engine. The vast majority of packets only transits the PFE at wire speeds. There are many advantages to using a standard interface, even internally. A standard interface is easier to implement than creating a new proprietary interface, and standard chipsets are readily available, inexpensive, and so on.

The routing engine of a Juniper Networks router contains the router's operating system, the JUNOS Internet software, the command line interface (CLI) for configuration and control, and the routing table (RT) itself. The routing table in a Juniper Networks router contains all of the routing information gathered from all routing protocols running on the router, as well as miscellaneous information such as interface addresses, static routes, and so forth.

It might not seem that the RE would have to be very powerful, or have a large hard drive, but it usually does. This is because of the increasing expense of converging a growing routing table.

The PFE is where the forwarding table resides. The forwarding table contains all the active route information that is actually used to determine the packet's next hop without needing to send the packet to the routing engine.

ROUTER ACCESS

Users don't generally communicate directly *with* routers, but rather *through* routers. The situation is different for network administrators and managers, however, who must communicate directly with the individual routers in order to install, configure, and manage the routers.

Routers are key devices on the Internet and almost any type of network. Many backbone routers handle packets for hundreds or thousands of users, and some handle packets for even more. So when a router goes down, or even slows down due to congestion or a problem, the users go wild and the network managers react immediately. For this reason, network managers need multiple and foolproof ways to access the routers they are responsible for in order to manage them.

Larger routers, and many smaller ones, do not normally come with a keyboard, mouse, and monitor. Nevertheless, there are usually three ways that a network administrator can communicate with a router.

The Console Port

This port is for a serial terminal that is at the same location as the router and attached by a short cable from the serial port on the terminal to the console port on the router. The terminal is usually a PC or Unix workstation running a terminal emulation program. There are several physical connector types used for this port on Cisco routers. Network administrators sometimes have to carry around several different connector types so they can be sure to have the proper connector for the router they need to manage. (Usually, after initial installation, the console ports are connected to a terminal server on a management network so that access does not have to be right where the router is.)

The Auxiliary Port

This port is for a serial terminal that is at a remote location. Connection is made through a pair of modems, one connected to the router and the other connected to the terminal. There is little difference, if any, between the auxiliary (AUX) and console ports in terms of characteristics. They are separate because routers might require simultaneous local and remote access that would be impossible if there were only one serial port on the router.

The Network

The router can always be managed over the same network on which it is routing packets. This is often called "in-band management" in contrast to the console and AUX ports, which are "out-of-band." This just means that the network access method shares the link to the router "in the same bandwidth" as user packets transiting the router. There are often three ways to access a router over the network: through Telnet

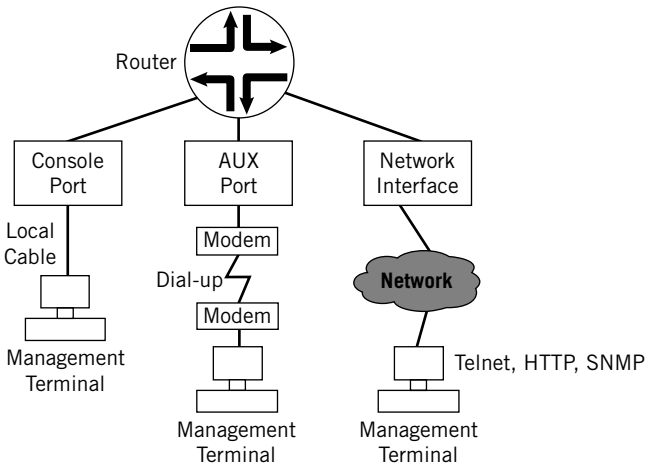


FIGURE 9.4

The three router access methods. Note that the console port requires access to the router, while the others allow remote access.

(called VTY lines on a Cisco router), with a more secure remote access program called secure shell (SSH), using a Web browser (HTTP is the protocol), or with SNMP (Simple Network Management Protocol), a protocol invented expressly for remote router management.

These arrangements are shown in Figure 9.4. Small routers usually only have a console port. With the proper cables, these console ports can be hooked up to a modem for remote access, but obviously cannot be used simultaneously for local access. On some routers, the console ports are labeled "Admin" or "Management." It is tempting to try and access a console or AUX ports using the normal graphical interface provided by Windows, a Mac, or Unix X-Windows. But the console and AUX ports only understand a simple, character-based serial protocol. On Windows PCs, for example, only HyperTerminal (or another serial terminal emulation program) can communicate with a router through the console or AUX ports.

FORWARDING TABLE LOOKUPS

In the connectionless, best-effort world of IP, every packet is forwarded independently, hop by hop, toward the destination. Each router determines the next hop for the destination address in the packet header based on information gathered into the routing table and distilled into the forwarding table. The essential operation of a router is the looking up of the packet's destination IP address in this table to determine the next hop.

It's unusual that a packet address is an exact match for a table entry. Otherwise, routing and forwarding tables would need an entry for every host in the world—all 32 bits for IPv4 and 128 bits for IPv6! So in the current classless (prefix) world of IP addressing, the host-hop destination is chosen by the *longest match* rule. Figure 9.5 shows how the next-hop address and interface information are used with the ARP process (cache or query) to forward the packet in a frame toward the destination.

Consider a packet sent to `10.10.11.77` (`bsdclient`) from LAN2. Remember, the network is `10.10.11.0/24`. Suppose the Best ISP edge router, `PE1`, has the entries shown in Table 9.1 about `10.10/16` networks in its tables; the longest match determines the correct interface that should forward the packet.

Which interface is the "best" next hop toward the destination? It would be easy if we had an entry like `10.10.11/24` to work with, but routers closer to the backbone use *aggregate* addresses in their tables. In most cases, Internet backbone routers will accept prefixes of `/24` or shorter. (It would be nice to accept only `/19` or shorter, but not many could get away with that.)

So where should the router send a packet for network `10.10.11.0/24`? Which next hop should it use? All three table entries are "close" to the destination address, but which one is "best"?

According to the longest-match rule, the router will send the packet for `10.10.11.77` to `10.10.17.2` on interface `so-0/0/2`. But how exactly does it work?

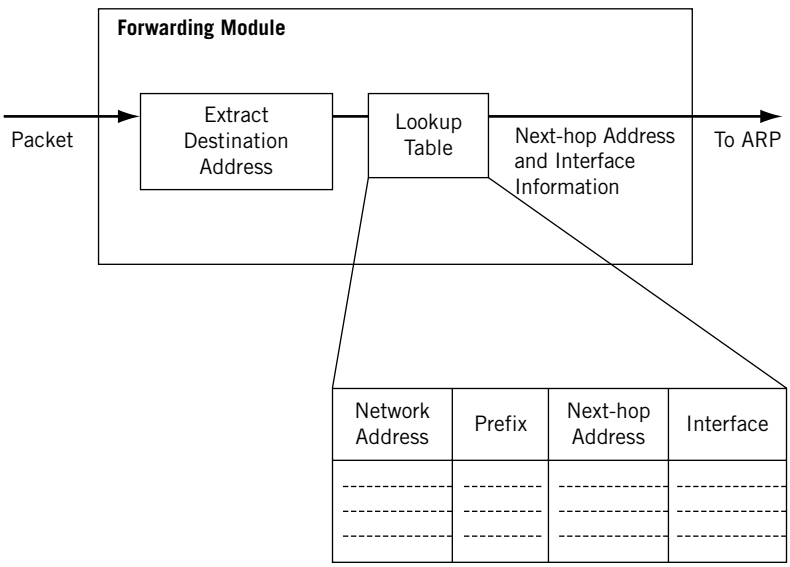


FIGURE 9.5

How the longest match rule applies to a forwarding table lookup. More specific (longer) routes are preferred to less specific (shorter) routes.

Table 9.1 Tables for Router PE1

Network (Network Bits in Bold)	Prefix	Next-Hop Address	Interface
10.10.0 (00001010 00001010 0000xxxx xxxx)	/20	10.0.12.2	so-0/0/0
10.10.8 (00001010 00001010 00001xxx xxxx)	/21	10.0.19.2	so-0/0/1
10.10.8 (00001010 00001010 000010xx xxxx)	/22	10.0.17.2	so-0/0/2

Routers today can "mix and match" prefixes of differing lengths in a routing or forwarding table and still send packets to the correct next hop. In the table, `10.10.8/21` and `10.10.8/22` are different routes, as would be `10.10.8/23` and `10.10.8/24`.

Now, the 32-bit destination address, `10.10.11.77`, in bits is `00001010 00001010 00001011 01001101`. There is, of course, no subnet mask associated with a host address. Looking at the table, the first 20 bits are exactly the same in all three entries, as well as the destination address. But which is the *longest* match? The router will keep comparing the addresses in the table to the destination address bit by bit until the table runs out of entries. The last match is the longest match, no matter if it's all 32 bits, or none (the default `0/0` entry matches everything).

The 21st bit is a 1 bit in the table entry for `10.10.8/21`, and so is the 21st bit in the destination address. The 22nd bit is a 0 bit in the table entry for `10.10.8/22`, and so is the 22nd bit in the destination address. There is no longer entry. This makes the `/22` entry the longest match for the destination address, and the packet is forwarded to `10.10.17.2`. The rest of the bits are used for local delivery of the packet on LAN2.

The longest match is also often called the *best match* or the *more specific route* for a given destination IP address. But whatever it is called, the point is the same: The longest-match next hop is always used in favor of a potential, but shorter match, next hop.

What if there were other entries such as `10.10.8/23` or `10.10.8/24`? It doesn't matter. The 1 bit in the 23rd position will not match these entries, which all have 0s at the end of the entry. The same longest match rules apply at each router.

DUAL STACKS, TUNNELING, AND IPv6

So far, we've seen how routers forward packets, what the routers look like internally, and how the longest match determines the output port. But most of this chapter dealt with IPv4. But what about IPv6 packets? It's one thing to say that some routers can handle both IPv4 and IPv6, but what about older or smaller routers and hosts that don't integrate IPv6 support and handle IPv4 only? This chapter ends with a consideration of the role of the router in a world that is slowly making its way toward IPv6.

The transition to IPv6 will be a long one for most networks. There might be networks where it will be necessary to mix hosts and routers that run IPv4 only, IPv6 only, and a combination of the two. Why would a host need to run both IPv4 and IPv6? Well, a Web site that only ran IPv6 would be forever unreachable by IPv4 browsers. Routers, of course, can be used to build separate IPv4 and IPv6 router networks. For example,

LAN1 and LAN2 could have two routers each—one for IPv4 and one for IPv6 traffic. But a lot of newer routers should be able to handle both IPv4 and IPv6 packets, and many do.

There are two main strategies that have emerged for dealing with mixed IPv4 and IPv6 environments. These are *dual protocol stacks* and *tunneling*.

Dual Protocol Stacks

All of the hosts on the Illustrated Network, as we have seen, are capable of assigning both an IPv6 and IPv4 address to their network interfaces. This is possible because they all implement a sort of "split" IP network layer. For example, if the Ethernet Type field is set to `0x0800` the packet is handed off to the IPv4 process, and if the Type field is set to `0x86DD`, then the packet is handed off to the IPv6 process. This is shown conceptually in Figure 9.6.

The dual protocol stack must provide error messages that are IPv6 "aware," and routing protocols have to adapt to IPv6 addresses as well (as we'll see). And in spite of the figure, which is a very common representation, the TCP/UDP layer is also dual.

Dual protocols stacks are not new with IPv6. This method was frequently used whenever two or more protocol stacks had to share a single host interface. In fact, very complex arrangements were not unknown, with IBM's (and Microsoft's) NetBios sharing the network with Novell's NetWare and IP itself (for Internet access).

Tunneling

Tunneling is a much misunderstood topic in general. This section talks about IPv6 tunnels, but networks also feature IPSec tunnels, VPN tunnels, and possibly even more. But they all employ tunnels. *Tunneling occurs whenever the normal sequence of encapsulation headers is violated.* That's all.

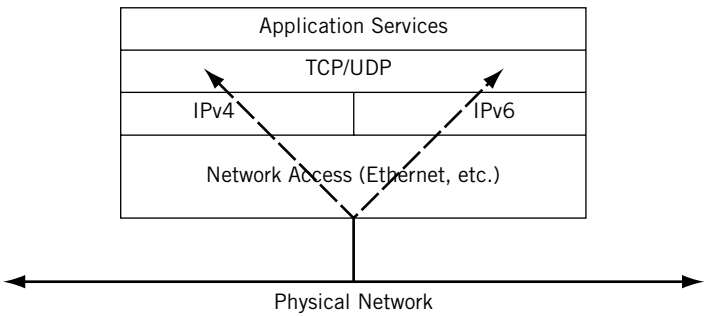


FIGURE 9.6

Dual protocol stacks for IPv4 and IPv6 sharing a single network connection. Technically, TCP and UDP have to be adjusted for an IPv6 environment.

Normally, a message is broken up into segments, which are put inside packets placed inside frames that are sent as a sequence of bits to an adjacent system. The receiver usually expects that the frame contains a packet, and so on, but what if it doesn't? Then the device is using tunneling.

We've already seen a form of tunneling in action. When we put PPP frames inside Ethernet frames, we put a frame inside a frame and violated the normal OSI-RM sequence of headers. That's okay, *as long as the receiver knows the sequence of headers the sender is generating*.

Not all devices need to know the exact sequence of encapsulations used by the sender and receiver. Only the *endpoints* (usually hosts, but not always) need to know how to encapsulate the data at one end and process the headers correctly at the destination. In between, inside the tunnel, all other devices can treat the data units as usual.

Tunneling in a mixed IPv4 and IPv6 network is used to transport IPv6 packets over a series of IPv4 routers or to an IPv4 host. There is a lot of variation in tunnels to support IPv4/IPv6 operation. For example, a native IPv6 backbone might tunnel IPv4 to reduce address consumption in the network core. For the sake of simplicity, let's consider four types of tunnels and two major scenarios for their use:

1. *Host to router*—Hosts with dual-stack capabilities can tunnel IPv6 packets to a dual-stack router that is only reachable over a series IPv4-only device.
2. *Router to router*—Routers with dual-stack capabilities can tunnel IPv6 packets over an IPv4 infrastructure to other routers.
3. *Router to host*—Routers with dual-stack capabilities can tunnel IPv6 packets over an IPv4 infrastructure to a dual-stack destination host.
4. *Host to host*—Hosts with dual-stack capabilities can tunnel IPv6 packets over an IPv4 infrastructure to other dual-stack IP hosts without an intervening router.

The four types of tunnels are shown in Figure 9.7. When the IPv6 packet is sent to a router (the first two tunneling methods), the endpoint of the tunnel is not the same as the destination, so the destination address of the IPv6 packet does not indicate the same device as the IPv4 tunnel endpoint address that carries the IPv6 packet. The source host or router must have the tunnel endpoint's IPv4 address configured. This is called *configured tunneling*.

In contrast, the last two methods send the encapsulated IPv6 packet directly to the destination host, so the IPv4 and IPv6 addresses used correspond to the same host. This lets the IPv6 destinations use IPv4-compatible addresses that are derived automatically by the devices. This is called *automatic tunneling* because it does not require explicit configuration.

Automatic tunneling uses a special form of the IPv6 address. The 32-bit IPv4 address is simply prepended with 96 zero bits in the form `0:0:0:0:0:0:<IPv4 address>`. This format is abbreviated as `::<IPv4 address>`.

All dual-stack IP hosts recognize this format and encapsulate the IPv6 packet inside an IPv4 packet using the embedded IPv4 address, creating an end-to-end tunnel. The

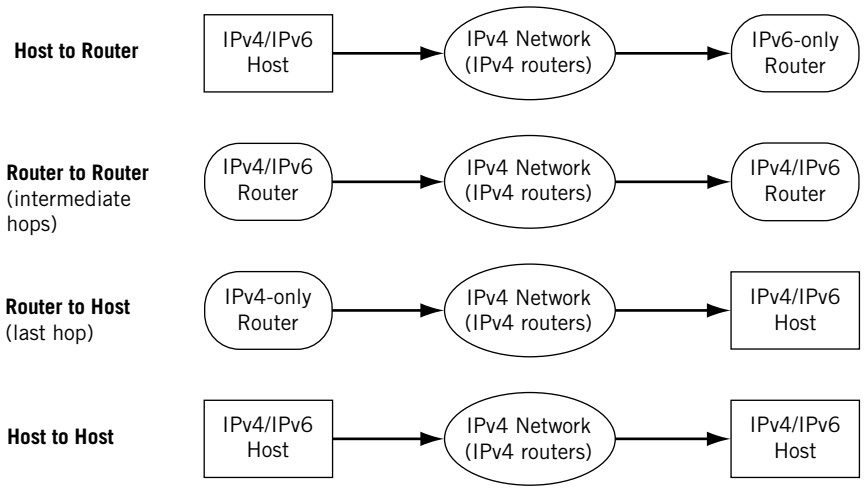


FIGURE 9.7

The various types of IPv6 tunnels, showing host and router situations that can be used to connect.

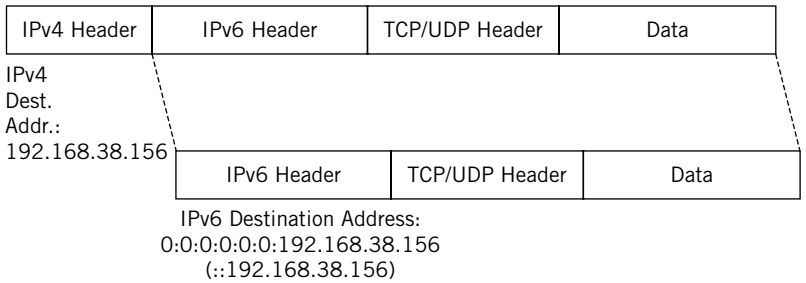


FIGURE 9.8

The special IPv6 tunnel-addressing format for dual-stack routers.

receiver simply strips off the IPv4 header and processes the IPv6 header and packet inside.

Hosts that only run IPv6 can use dual-stack routers to communicate using this special form of IPv6 address also. Dual-stack routers recognize the IPv6 traffic and use the last 32 bits to create the IPv4 address for the IPv4 "wrapper." Figure 9.8 shows how this special addressing format works. Naturally, this requires IPv6-only hosts to have valid and routable IPv4 addresses, which clearly marks the format as a transitional method. If the IPv6 address is not in this special address form, then a configured tunnel must be used, or, if every device on the path from source to destination uses dual protocol stacks, or IPv6 only, well-formed IPv6 addresses can be used.

TUNNELING MECHANISMS

The theory of tunneling IPv6 packets through a collection of IPv4 routers is one thing. Exactly how to do it is another. There are several tunnel *mechanisms* that embody the concepts discussed previously.

Manually configured tunnels—These are defined in RFC 2893, and both endpoints of the tunnel must have both IPv4 and IPv6 addresses. These tunnels are usually used between dual-stack edge routers.

Generic Routing Encapsulation (GRE) tunnels—GRE tunnels were designed to transport non-IP protocols over an IP network. But GRE is also a good way to carry IPv6 across the IPv4 routers. We used a GRE tunnel earlier in this chapter.

IPv4-compatible (6over4) tunnels—Also defined in RFC 2893, these are the automatic tunnels based on IPv4-compatible IPv6 addresses using the `::<IPv4 address>` form of IPv6 address.

6to4 tunnels—Another form of automatic tunnel defined in RFC 3065. They use an IPv4 address embedded in the IPv6 address to identify the tunnel endpoint.

Intra-site Automatic Tunnel Addressing Protocol (ISATAP) tunnels—ISATAP tunnels are a mechanism much like 6to4 tunneling, but for local site (campus) networks. An ISATAP address uses a special prefix and the IPv4 address to identify the endpoint.

The differences between the 6to4 tunnel and the ISATAP tunnel address are shown in Figure 9.9.

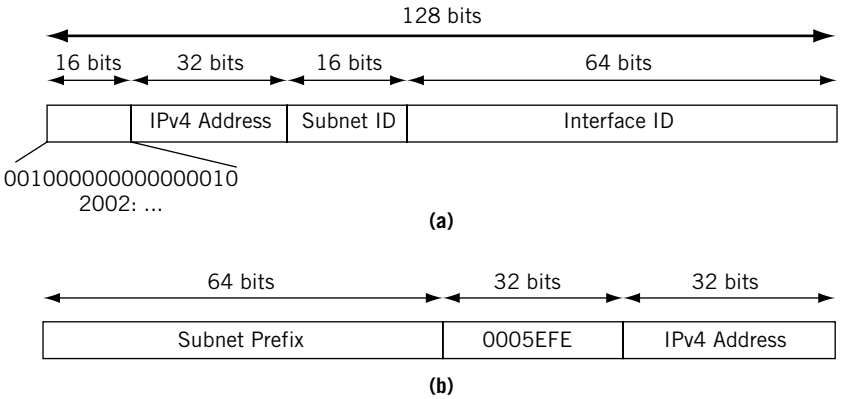


FIGURE 9.9

The differences between 6to4 and ISATAP tunnel addressing, showing how the 128 bits of the IPv6 address are structured in each case. (a) 6to4 tunneling address format (b) ISATAP tunneling address format

TRANSITION CONSIDERATIONS

Routers occupy a key position during the transition period between IPv4 and IPv6. There are still a lot of routers, mostly older ones, that do not handle IPv6 or understand only the `::<IPv4 address>` form of IPv6 address. How will IPv4 and IPv6 routers and hosts interoperate?

A transition plan has been put in place and contains some distinct terminology that is new. The IPv4 to IPv6 transition plan defines the following terms for nodes:

- *IPv4-only Node*—A host or router that implements only IPv4.
- *IPv6/IPv4 (dual) Node*—A host or router that implements both IPv4 and IPv6.
- *IPv6-only Node*—A host or router that implements only IPv6.
- *IPv6 Node*—A host or router that implements IPv6. Both IPv4/IPv6 dual nodes and IPv6-only nodes are included in this category.
- *IPv4 Node*—A host or router that implements IPv4. Both IPv4/IPv6 dual nodes and IPv4-only nodes are included in this category.

In addition, the plan defines three types of addresses:

1. *IPv4-compatible IPv6 address*—An address assigned to an IPv6 node that can be used in both IPv6 and IPv4 packets. The `::<IPv4 address>` format is used for this type of IP address. For example, an address such as `::10.10.11.66` is used when there is no IPv6 router available.
2. *IPv4-mapped IPv6 address*—An address assigned to an IPv4-only node represented as an IPv6 address. These addresses always identify IPv4-only nodes, never IPv4/IPv6 or IPv6-only nodes. These are provided when an IPv6 application requests the host name for a node with an IPv4 address only. For example, `::FFFF:10.10.12.166` is an IPv4-mapped IPv6 address.
3. *IPv6-only address*—An address globally assigned to any IPv4/IPv6 or IPv6-only node. These addresses never identify IPv4-only nodes.

These terms can be somewhat confusing, but all they mean is that hosts and routers can be classified either as IPv4 devices, IPv6 devices, or both IPv4 and IPv6 devices. The IPv4/IPv6 devices are capable of understanding and using both IPv4 and IPv6. However, the IPv6-only address (an address that has no relationship to an IPv4 address) can be used in an IPv6/IPv4 device.

QUESTIONS FOR READERS

Figure 9.10 shows some of the concepts discussed in this chapter and can be used to help you answer the following questions.

```
admin@router0> show route
inet.0: 2 destinations, 2 routes (2 active...
10.10.0.0/16          >via interface #1
10.10.64.0/18         >via interface #2
10.10.128.0/18        >via interface #3
```

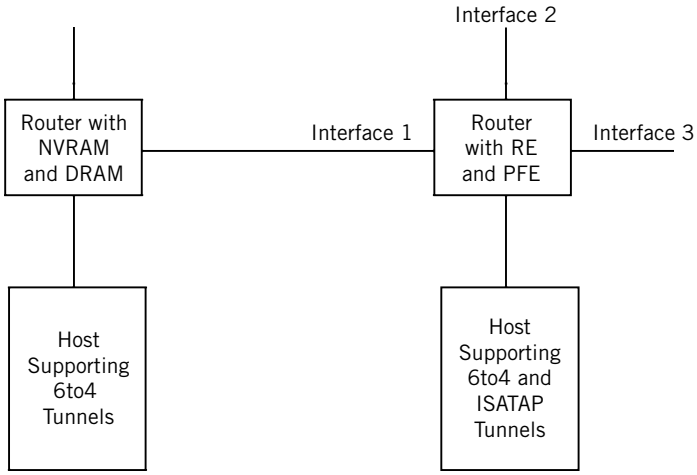


FIGURE 9.10

A simple network of routers and hosts, showing architecture, a routing table, and tunnel support.

1. Which router, based on the architecture in the figure, is probably a small site router? Which is probably a large Internet backbone router?
2. Which output interface, based on the routing table shown in the figure, will packets arriving from the directly attached host for IPv4 address `10.10.11.1` use for forwarding? Assume longest match is used.
3. Which output interface will packets for `10.10.192.10` use? Assume the longest match is used.
4. Which IPv6 tunneling protocol can be used between the two hosts? How many bits will be used for the subnet identifier?
5. Do the routers require IPv6 support to deliver packets between the two hosts?