IP Routing

IP Datagram Direct Delivery and Indirect Delivery (Routing)

The overall job of the Internet Protocol is to transmit messages from higher layer protocols over an internetwork of devices. These messages must be packaged and addressed, and if necessary fragmented, and then they must be *delivered*. The process of delivery can be either simple or complex, depending on the proximity of the source and destination devices.

Datagram Delivery Types

Conceptually, we can divide all IP datagram deliveries into two general types, shown graphically in Figure 91:

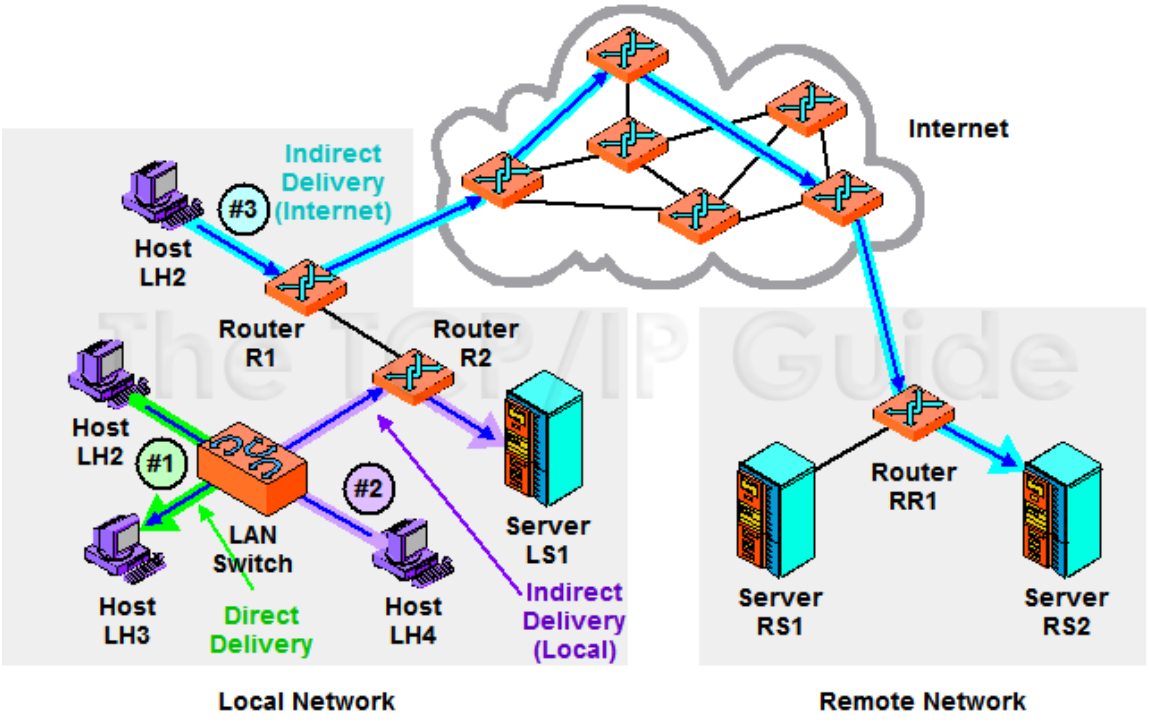

Figure 91: Direct and Indirect (Routed) Delivery of IP Datagrams

This diagram shows three examples of IP datagram delivery. The first transmission (highlighted in green) shows a direct delivery between two devices on the local network. The second (purple) shows indirect delivery within the local network, between a client and server separated by a router. The third shows a more distant indirect delivery, between a client on the local network and a server

across the Internet.

- **Direct Datagram Deliveries:** When datagrams are sent between two devices on the same physical network, it is possible for datagrams to be delivered directly from the source to the destination. Imagine that you want to deliver a letter to a neighbor on your street. You probably wouldn't bother mailing it through the post office; you'd just put the neighbor's name on the envelope and stick it right into his or her mailbox.

- **Indirect Datagram Deliveries:** When two devices are not on the same physical network, the delivery of datagrams from one to the other is *indirect*. Since the source device can't see the destination on its local network, it must send the datagram through one or more intermediate devices to deliver it. Indirect delivery is analogous to mailing a letter to a friend in a different city. You don't deliver it yourself—you put it into the postal system. The letter journeys through postal system, possibly taking several intermediate steps, and ends up in your friend's neighborhood, where a postal carrier puts it into his or her mailbox.

Comparing Direct and Indirect Delivery

Direct delivery is obviously the simpler of these. The source just sends the IP datagram down to its data link layer implementation. The data link layer encapsulates the datagram in a frame that is sent over the physical network directly to the recipient's data link layer, which passes it up to the IP layer.

Indirect delivery is much more complicated, because we can't send the data straight to the recipient. In fact, we usually will not even know where the recipient is, exactly. Sure, we have its address, but we may not know what network it is on, or where that network is relative to our own. (If I told you my address you'd know it's somewhere in Bennington, Vermont, but could you find it?) Like relying on the postal system in the envelope analogy, we must rely on the internetwork itself to indirectly deliver datagrams. And like the postal system, the power of IP is that you don't need to know how to get the letter to its recipient; you just put it into the system.

The devices that accomplish this "magic" of indirect delivery are generally known as *routers*, and indirect delivery is more commonly called *routing*. Like entrusting a letter to your local mail carrier or mailbox, a host that needs to send to a distant device generally sends datagrams to its local router. The router connects to one or more other routers, and they each maintain information about where to send datagrams so that they reach their final destination.

Indirect delivery is almost always required when communicating with distant devices, such as those on the Internet or across a WAN link. However, it may

also be needed even to send to a device in the next room of your office, if that device is not connected directly to yours at layer two.

Note: In the past, routers were often called *gateways*. Today, this term more generally can refer to a device that connects networks in a variety of ways. You will still sometimes hear routers called gateways—especially in the context of terms like "default gateway"—but since it is ambiguous, the term *router* is preferred.

The Relationship Between Datagram Routing and Addressing

Obviously, each time a datagram must be sent, it is necessary that we determine first of all whether we can deliver it directly or if routing is required. Remember all those pages and pages of details about IP addressing? Well, this is where the payoff is. The same thing that makes IP addressing sometimes hard to understand—the division into network ID and host ID bits, as well as the subnet mask—is what allows a device to quickly determine whether or not it is on the same network as its intended recipient:

- **Conventional "Classful" Addressing:** We know the class of each address by looking at the first few bits. This tells us which bits of an address are the network ID. If the network ID of the destination is the same as our own, the recipient is on the same network; otherwise, it is not.

- **Subnetted "Classful" Addressing:** We use our subnet mask to determine our network ID and subnet ID and that of the destination address. If the network ID and subnet are the same, the recipient is on the same subnet. If only the network ID is the same, the recipient is on a different subnet of the same network. If the network ID is different, the destination is on a different network entirely.

- **Classless Addressing:** The same basic technique is used as for subnetted "classful" addressing, except that there are no subnets. We use the "slash number" to determine what part of the address is the network ID and compare the source and destination as before. There are complications here, however, that I discuss more in the topic on routing in a classless environment.

Key Concept: The delivery of IP datagrams is divided into two categories: *direct* and *indirect*. Direct delivery is possible when two devices are on the same physical network. When they are not, indirect delivery, more commonly called *routing*, is required to get the datagrams from source to destination. A device can tell which type of delivery is required by looking at the IP address of

the destination, in conjunction with supplemental information such as the subnet mask that tells the device what network or subnet it is on.

The determination of what type of delivery is required is the first step in the source deciding where to send a datagram. If it realizes the destination is on the same local network it will address the datagram to the recipient directly at the data link layer. Otherwise, it will send the datagram to the data link layer address of one of the routers to which it is connected. The IP address of the datagram will still be that of the ultimate destination. Mapping between IP addresses and data link layer addresses is accomplished using the TCP/IP Address Resolution Protocol (ARP).

I should also clarify one thing regarding the differentiation between direct and indirect delivery. Routing is done in the latter case to get the datagram to the local network of the recipient. After the datagram has been routed to the recipient's physical network, it is sent to the recipient by the recipient's local router. So, you could say that indirect delivery includes direct delivery as its final step.

The next topic discusses IP routing processes and concepts in more detail.

Note: Strictly speaking, any process of delivery between a source and destination device can be considered routing, even if they are on the same network. It is common, however, for the process of routing to refer more specifically to indirect delivery as explained above.

IP Routing Concepts and the Process of Next-Hop Routing

When a datagram is sent between source and destination devices that are not on the same physical network, the datagram must be delivered indirectly between the devices, a process called *routing*. It is this ability to route information between devices that may be far away that allows IP to create the equivalent of a virtual internetwork that spans potentially thousands of physical networks, and lets devices even on opposite ends of the globe communicate. The process of routing in general terms is too complex to get into in complete detail here, but I do want to take a brief look at key IP routing concepts.

Overview of IP Routing and Hops

To continue with our postal system analogy, I can send a letter from my home in the United States to someone in, say, India, and the postal systems of both countries will work to deliver the letter to its destination. However, when I drop a letter in the mailbox, it's not like someone shows up, grabs the letter, and hand-delivers it to the right address in India. The letter travels from the mailbox to my local post office. From there, it probably goes to a regional distribution center, and then from there, to a hub for international traffic. It goes to India, perhaps (likely) via an intermediate country. When it gets to India, the Indian postal system uses its own network of offices and facilities to route the letter to its destination. The envelope "hops" from one location to the next until it reaches its destination.

IP routing works in very much the same manner. Even though IP lets devices "connect" over the internetwork using indirect delivery, all of the actual communication of datagrams occurs over physical networks using *routers*. We don't know where exactly the destination device's network is, and we certainly don't have any way to connect directly to each of the thousands of networks out there. Instead, we rely on intermediate devices that are each physically connected to each other in a variety of ways to form a mesh containing millions of paths between networks. To get the datagram where it needs to go, it needs to be handed off from one router to the next, until it gets to the physical network of the destination device. The general term for this is *next-hop routing*. The process is illustrated in Figure 92.

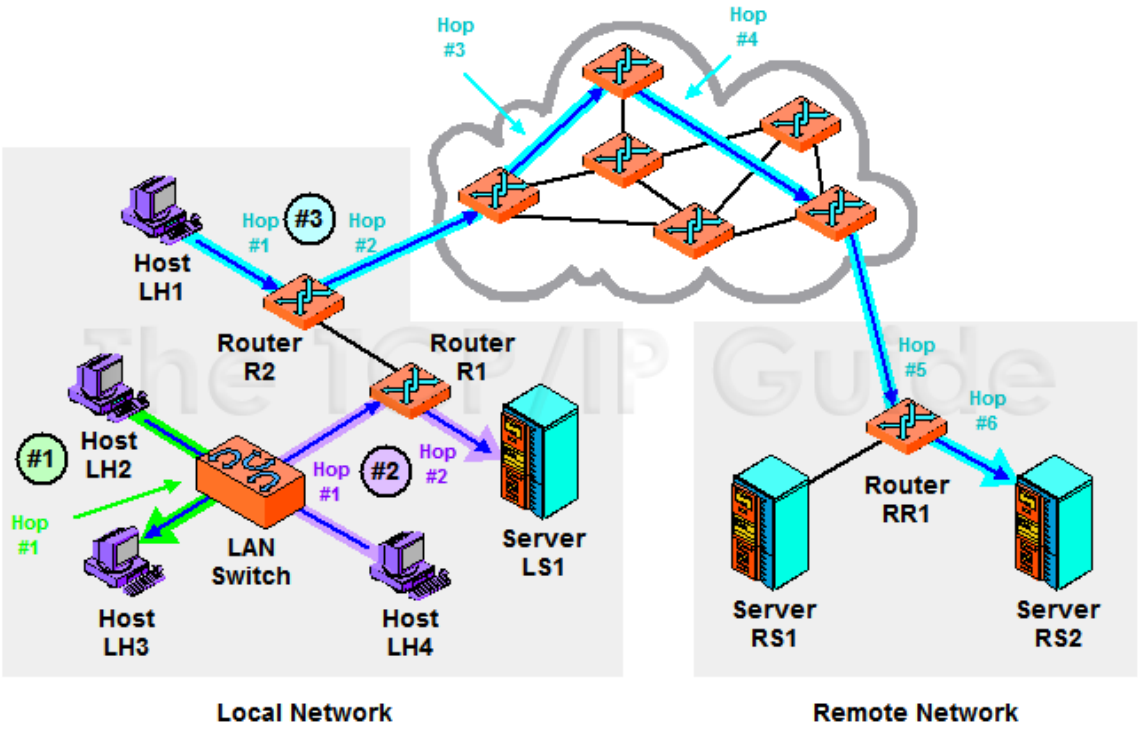

Figure 92: IP Datagram Next-Hop Routing

This is the same diagram as that shown in Figure 91, except this time I have explicitly shown the hops taken by each of the three sample transmissions. The direct delivery of the first (green) transmission has only one hop (remember that the switch doesn't count because it is invisible at layer three). The local indirect delivery passes through one router, so it has two hops. The Internet delivery in this case has six hops; actual Internet routes can be much longer.

The Benefits of Next-Hop Routing

This is a critical concept in how IP works: routing is done on a step-by-step basis, one hop at a time. When we decide to send a datagram to a device on a distant network, we don't know the exact path that the datagram will take; we only have enough information to send it to the correct router to which we are attached. That router, in turn, looks at the IP address of the destination and decides where the datagram should next "hop" to. This process continues until the datagram reaches the destination host's network, when it is delivered.

Next-hop routing may seem at first like a strange way of communicating datagrams over an internetwork. In fact, it is part of what makes IP so powerful. On each step of the journey to any other host, a router only needs to know where the next step for the datagram is. Without this concept, each device and router would need to know what path to take to every other host on the internet, which would be quite impractical.

Key Concept: Indirect delivery of IP datagrams is accomplished using a process called *next-hop routing*, where each message is handed from one router to the next until it reaches the network of the destination. The main advantage of this is that each router needs only to know which neighboring router should be the next recipient of a given datagram, rather than needing to know the exact route to every destination network.

Datagram Processing At Each Hop

As mentioned above, each "hop" in routing consists of traversal of a physical network. After a source sends a datagram to its local router, the data link layer on the router passes it up to the router's IP layer. There, the datagram's header is examined, and the router decides what the next device is to send the datagram to. It then passes it back down to the data link layer to be sent over one of the router's physical network links, typically to another router. The router will either have a record of the physical addresses of the routers to which it is connected, or it will use ARP to determine these addresses.

The Essential Role of Routers in IP Datagram Delivery

Another key concept related to the principle of next-hop routing is that routers are designed to accomplish routing, not hosts. Most hosts are connected using only one router to the rest of the internet (or Internet). It would be a maintenance nightmare to have to give each host the smarts to know how to route to every other host. Instead, hosts only decide if they are sending locally to their own network, or if they are sending to a non-local network. If the latter, they just send the datagram to their router and say “here, ***you*** take care of this”. If a host has a connection to more than one router, it only needs to know which router to use for certain sets of distant networks. How routers decide what to do with the datagrams when they receive them from hosts is the subject of the next topic.

IP Routes and Routing Tables

Routers are responsible for forwarding traffic on an IP internetwork. Each router accepts datagrams from a variety of sources, examines the IP address of the destination and decides what the next hop is that the datagram needs to take to get it that much closer to its final destination. A question then naturally arises: how does a router know where to send different datagrams?

Each router maintains a set of information that provides a mapping between different network IDs and the other routers to which it is connected. This information is contained in a data structure normally called a *routing table*. Each entry in the table, unsurprisingly called a *routing entry*, provides information about one network (or subnetwork, or host). It basically says “if the destination of this datagram is in the following network, the next hop you should take is to the following device”. Each time a datagram is received the router checks its destination IP address against the routing entries in its table to decide where to send the datagram, and then sends it on its next hop.

Obviously, the fewer the entries in this table, the faster the router can decide what to do with datagrams. (This was a big part of the motivation for classless addressing, which aggregates routes into “supernets” to reduce router table size, as we will see in the next topic.) Some routers only have connections to two other devices, so they don't have much of a decision to make. Typically, the router will simply take datagrams coming from one of its interfaces and if necessary, send them out on the other one. For example, consider a small company's router acting as the interface between a network of three hosts and the Internet. Any datagrams sent to the router from a host on this network will need to go over the router's connection to the router at the ISP.

When a router has connections to more than two devices, things become considerably more complex. Some distant networks may be more easily reachable if datagrams are sent using one of the routers than the other. The

routing table contains information not only about the networks directly connected to the router, but also information that the router has “learned” about more distant networks.

Key Concept: A router make decisions about how to route datagrams using its internal *routing table*. The table contains entries specifying to which router datagrams should be sent to reach a particular network.

Routing Tables in an Example Internetwork

Let’s consider an example (see Figure 93) with routers R1, R2 and R3 connected in a “triangle”, so that each router can send directly to the others, as well as to its own local network. Suppose R1's local network is 11.0.0.0/8, R2's is 12.0.0.0/8 and R3's is 13.0.0.0/8. (I'm just trying to keep this simple. ☺) R1 knows that any datagram it sees with 11 as the first octet is on its local network. It will also have a routing entry that says that any IP address starting with “12” should go to R2, and any starting with “13” should go to R3.

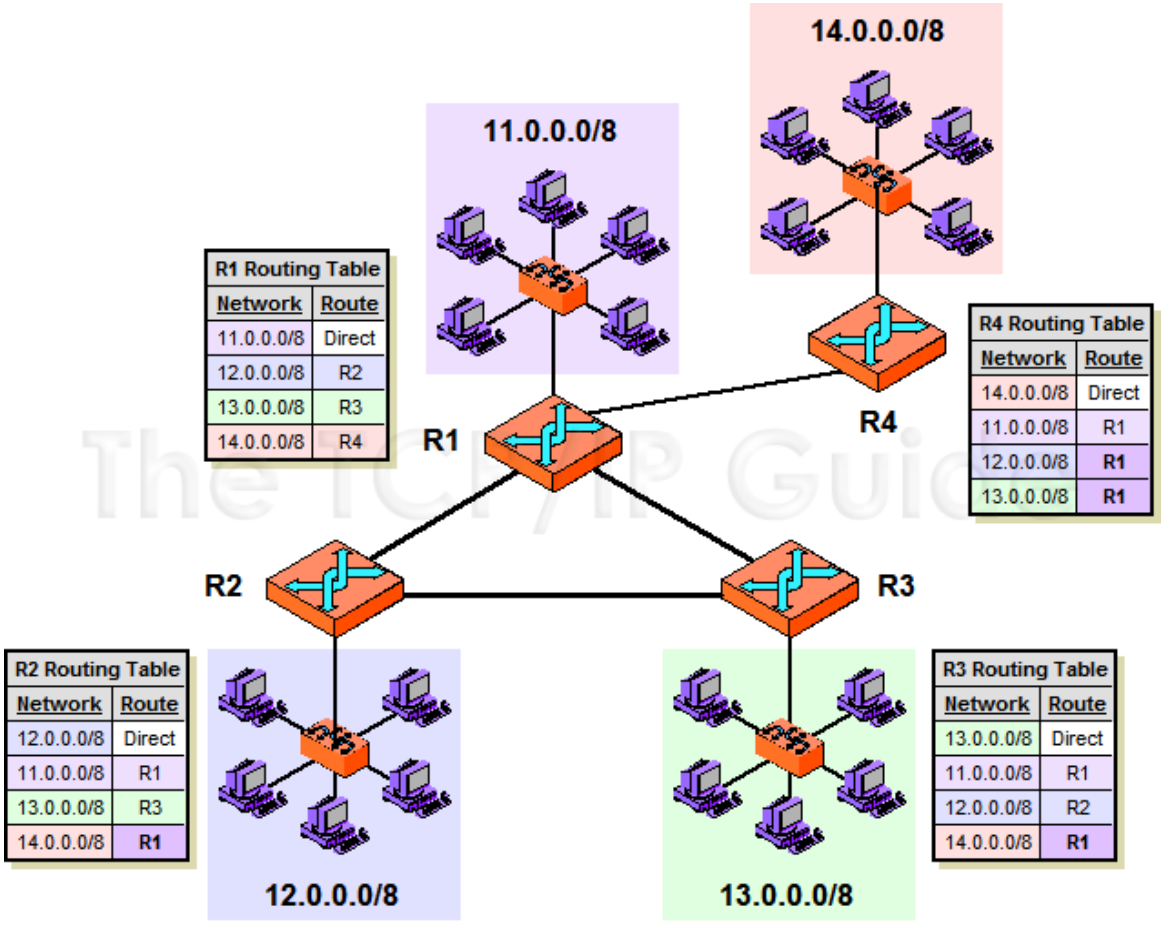

Figure 93: IP Routing and Routing Tables

This diagram shows a small, simple internetwork consisting of four LANs each served by a router. The routing table for each lists the router to which datagrams for each destination network should be sent, and is color coded to match the colors of the networks. Notice that due to the "triangle", each of R1, R2 and R3 can send to each other. However, R2 and R3 must send through R1 to deliver to R4, and R4 must use R1 to reach either of the others.

Let's suppose that R1 also connects to another router, R4, which has 14.0.0.0/8 as its local network. R1 will have an entry for this local network. However, R2 and R3 also need to know how to reach 14.0.0.0/8, even though they don't connect to it its router directly. Most likely, they will have an entry that says that any datagrams intended for 14.0.0.0/8 should be sent to R1. R1 will then forward them to R4. Similarly, R4 will send any traffic intended for 12.0.0.0/8 or 13.0.0.0/8 through R1.

Route Determination

Now, imagine that this process is expanded to handle thousands of networks and routers. Not only do routers need to know which of their local connections to use for each network, they want to know, if possible, what is the ***best*** connection to use for each network. Since routers are interconnected in a mesh there are usually multiple routes between any two devices, but we want to take the best route whenever we can. This may be the shortest route, the least congested, or the route considered optimal based on other criteria.

Determining what routes we should use for different networks turns out to be an important but very complex job. Routers must plan routes and exchange information about routes and networks, which can be done in a variety of ways. This is accomplished in IP using special IP *routing protocols*. It is through these protocols that R2 and R3 would find out that 14.0.0.0/8 exists and that it is connected to them via R1. I discuss these important "support protocols" in their own section.

Note: There is a difference between a ***routable*** protocol and a ***routing*** protocol. IP is a *routable* protocol, which means its messages (datagrams) can be routed. Examples of *routing* protocols are RIP or BGP, which are used to exchange routing information between routers.

IP Routing In A Subnet Or Classless Addressing (CIDR) Environment

There are three main categories of IP addressing: “classful”, subnetted “classful”, and classless. As we have already seen, the method used for determining whether direct or indirect delivery of a datagram is required is different for each type of addressing. The type of addressing used in the network also impacts how routers decide to forward traffic in an internet.

One of the main reasons why the traditional class-based addressing scheme was created was that it made both addressing and routing relatively simple. We must remember that IPv4 was developed in the late 1970s, when the cheap and powerful computer hardware we take for granted today was still in the realm of science fiction. For the internetwork to function properly, routers had to be able to look at an IP address and quickly decide what to do with it.

“Classful” addressing was intended to made this possible. There was only a two-level hierarchy for the entire internet: network ID and host ID. Routers could tell by looking at the first four bits which of the bits in any IP address were the network ID and which the host ID. Then they needed only consult their routing tables to find the network ID and see which router was the best route to that network.

The addition of subnetting to conventional addressing didn't really change this for the main routers on the internet, because subnetting is internal to the organization. The main routers handling large volumes of traffic on the Internet didn't look at subnets at all; the additional level of hierarchy that subnets represent existed only for the routers within each organization that chose to use subnetting. These routers, when deciding what to do with datagrams within the organization's network, had to extract not only the network ID of IP addresses, but also the subnet ID. This told them which internal physical network to send the datagram to.

Aggregated Routes and their Impact on Routing

Classless addressing is formally called *Classless Inter-Domain Routing* or *CIDR*. The name mentions routing and not addressing, and this is evidence that CIDR was introduced in large part to improve the efficiency of routing. This improvement occurs because classless networks use a multiple-level hierarchy. Each network can be broken down into subnetworks, sub-subnetworks, and so on. This means that when we are deciding how to route in a CIDR environment, we can also describe routes in a hierarchical manner. Many smaller networks can be described using a single, higher-level network description that represents them all to routers in the rest of the internet. This technique, sometimes called *route aggregation*, reduces routing table size.

Let's refer back to the detailed example I gave in the addressing section on CIDR. An ISP started with the block 71.94.0.0/15 and subdivided it multiple times

to create smaller blocks for itself and its customers. To the customers and users of this block, these smaller blocks must be differentiated; the ISP obviously needs to know how to route traffic to the correct customer. To everyone else on the Internet, however, these details are unimportant in deciding how to route datagrams to anyone in that ISP's block. For example, suppose I am using a host with IP address 211.42.113.5 and I need to send to 71.94.1.43. My local router, and the main routers on the Internet, don't know where in the 71.94.0.0/15 block that address is, and they don't need to know either. They just know that anything with the first 15 bits containing the binary equivalent of 71.94 goes to the router(s) that handle(s) 71.94.0.0/15, which is the aggregated address of the entire block. They let the ISP's routers figure out which of its constituent subnetworks contains 71.94.1.43.

Contrast this to the way it would be in a “classful” environment. Each of the customers of this ISP would probably have one or more Class C address blocks. Each of these would require a separate router entry, and these blocks would have to be known by ***all*** routers on the Internet. Thus, instead of just one 71.94.0.0/15 entry, there would be dozens or even hundreds of entries for each customer network. In the classless scheme, only one entry exists, for the “parent” ISP.

Potential Ambiguities in Classless Routes

CIDR provides benefits to routing but also increases complexity. Under CIDR, we cannot determine which bits are the network ID and which the host ID just from the IP address. To make matters worse, we can have networks, subnetworks, sub-subnetworks and so on that all have the same base address!

In our example above, 71.94.0.0/15 is the complete network, and subnetwork #0 is 71.94.0.0/16. They have a different prefix length (the number of network ID bits) but the same base address. If a router has more than one match for a network ID in this manner, it must use the match with the longest network identifier first, since it represents a more specific network description.

TCP/IP Communication Verification Utility (ping/ping6)

One of the most common classes of problems that network administrators are often called upon to solve is an inability of two hosts to communicate. For example, a user on a corporate network might not be able to retrieve one of his files from a local server, or another user might be having difficulty loading her favorite Web site. In these and many similar situations, one important step in diagnosing the problem is to verify that basic communication is possible between

the TCP/IP software stacks on the two machines. This is most often done using the *ping* utility, or *ping6* in IPv6 implementations.

Note: Some people say that "ping" is an acronym for "Packet Internet Groper", while others insist that it is actually based on the use of the term to refer to a sonar pulse sent by a submarine to check for nearby objects. I really don't know which of these is true, but I prefer the second explanation. Consider that the utility works in a way similar to a sonar "ping", and that it was originally written by a gent named Mike Muuss, who worked at the US Army Ballistics Research Laboratory. The first explanation is weaker; it's possible, but the phrase "Packet Internet Groper" isn't really grammatical, and I don't even want to ***think*** about what it is this utility is supposed to be "groping"! ☺

The *ping* diagnostic utility is one of the most commonly used, and is present in just about every TCP/IP implementation. It is usually implemented and accessed as a command-line utility, though there are also now graphical and menu-based versions of the program on some operating systems.

Operation of the ping Utility

The *ping* utility is implemented using ICMP *Echo (Request)* and *Echo Reply* messages. As explained in the topic discussing these message types, they are designed specifically for these sorts of diagnostic purposes. When Device *A* sends an ICMP *Echo* message to device *B*, device *B* responds by sending an ICMP *Echo Reply* message back to device *A*. The same functionality exists in ICMPv6, the IPv6 version of ICMP; the ICMPv6 Echo and Echo Reply messages only differ from the IPv4 ones slightly in their field structure.

This would seem to indicate that *ping* would be an extremely simple utility that would send one *Echo* message and wait to see if an *Echo Reply* was received back; if so, then this would provide that the two devices were able to communicate, and if not, this would indicate a problem somewhere on the internetwork between the two. However, almost all *ping* implementations are much more complex than this. They use multiple sets of *Echo* and *Echo Reply* messages, along with considerable internal logic, to allow an administrator to determine all of the following, and more:

- Whether or not the two devices can communicate;
- Whether congestion or other problems exist that might allow communication to succeed sometimes but cause it to fail in others, seen as packet loss—if so, how bad the loss is;

- How much time it takes to send a simple ICMP message between devices, which gives an indication of the overall latency between the hosts, and also indicates if there are certain types of problems.

Basic ping Use

The most basic use of the *ping* command is to enter it by itself with the IP address of a host. Virtually all implementations also allow a host name to be used, which will be resolved to an IP address automatically. When the utility is invoked with no additional options, default values are used for parameters such as what size message to send, how many messages to be sent, how long to wait for a reply, and so on. The utility will transmit a series of *Echo* messages to the host and report back whether or not a reply was received for each; if a reply is seen, it will also indicate how long it took for the response to be received. When the program is done, it will provide a statistical summary showing what percentage of the *Echo* messages received a reply, and the average amount of time for them to be received.

Table 284 shows an example using the *ping* command on a Windows XP computer (mine!), which by default sends four 32-byte *Echo* messages and allows four seconds before considering an *Echo* message lost. I use a satellite Internet connection that has fairly high latency and also occasionally drops packets. This isn't great for me, but it is useful for illustrating how *ping* works.

Table 284: Verifying Communication Using the *ping* Utility

```
D:\aa>ping www.pcguide.com
Pinging pcguide.com [209.68.14.80] with 32 bytes of data:

Reply from 209.68.14.80: bytes=32 time=582ms TTL=56
Reply from 209.68.14.80: bytes=32 time=601ms TTL=56
Request timed out.
Reply from 209.68.14.80: bytes=32 time=583ms TTL=56

Ping statistics for 209.68.14.80:
Packets: Sent = 4, Received = 3, Lost = 1 (25% loss),
Approximate round trip times in milli-seconds:
Minimum = 582ms, Maximum = 601ms, Average = 588ms
```

Methods of Diagnosing Connectivity Problems Using ping

Most people find that using *ping* with default settings is enough for their needs. In fact, the utility can be used in this simplest form to perform a surprising number of diagnostic checks. In many cases, the *ping* command can be used to diagnose connectivity problems by using it multiple times in sequence, often starting with

checks at or close to the transmitting device and then proceeding outwards towards the other device with which the communication problem has been observed. Some examples of how *ping* can be used in this way:

- **Internal Device TCP/IP Stack Operation:** By performing a *ping* on the device's own address, you can verify that its internal TCP/IP stack is working. This can also be done using the standard IP loopback address, 127.0.0.1.
- **Local Network Connectivity:** If the internal test succeeds, it's a good idea to do a *ping* on another device on the local network, to verify that local communication is possible.
- **Local Router Operation:** If there is no problem on the local network, it makes sense to *ping* whatever local router the device is using to make sure it is operating and reachable.
- **Domain Name Resolution Functionality:** If a *ping* performed on a DNS domain name fails, you should try it with the device's IP address instead. If that works, this implies either a problem with domain name configuration or resolution.
- **Remote Host Operation:** If all the preceding checks succeed, you can try *pinging* a remote host to see if it responds. If it does not, you can try a different remote host; if that one works, it is possible that the problem is actually with the first remote device itself and not with your local device.

Note: While the inability to get a response from a device to a *ping* has traditionally been interpreted as a problem in communication, this is not always necessarily the case. In the current era of increased security consciousness, some networks are set up to not respond to *Echo* messages, to protect against attacks that use floods of such messages. In this case a *ping* will fail, even though the host may be quite reachable.

Key Concept: The TCP/IP *ping* utility is used to verify the ability of two devices on a TCP/IP internetwork to communicate. It operates by having one device send ICMP *Echo (Request)* messages to another, which responds with *Echo Reply* messages. The program can be helpful in diagnosing a number of connectivity issues, especially if it is used to test the ability to communicate with other devices in different locations. It also allows the average round-trip delay to exchange messages with another device to be estimated.

ing Options and Parameters

All *ping* implementations include a number of options and parameters that allow an administrator to fine-tune how it works. They allow *ping* to be used for more extensive or specific types of testing. For example, *ping* can be set in a mode where it sends *Echo* messages continually, to check for an intermittent problem over a long period of time. You can also increase the size of the messages sent or the frequency with which they are transmitted, to test the ability of the local network to handle large amounts of traffic.

As always, the exact features of the *ping* program are implementation-dependent; even though UNIX and Windows systems often include many of the same options, they usually use completely different option codes. Table 285 shows some of the more important options that are often defined for the utility on many UNIX systems, and where appropriate, the parameters supplied with the option. Table 286 shows a comparable table for a typical Windows system.

Table 285: Common UNIX *ping* Utility Options and Parameters

Option / Parameters	Description
-c <count>	Specifies the number of *Echo* messages that should be sent.
-f	Flood mode; sends *Echo* packets at high speed to stress-test a network. This can cause serious problems if not used carefully!
-i <wait-interval>	Tells the utility how long to wait between transmissions.
-m <ttl-value>	Overrides the default Time To Live (TTL) value for outgoing Echo messages.
-n	Numeric output only; suppresses lookups of DNS host names to save time.
-p <pattern>	Allows a byte pattern to be specified for inclusion in the transmitted *Echo* messages. This can be useful for diagnosing certain odd problems that may only occur with certain types of transmissions.
-q	Quiet output; only summary lines are displayed at the start and end of the program's execution, while the lines for each individual message are suppressed.
-R	Tells the utility to include the *Record Route* IP option, so the route taken by the ICMP *Echo* message can be displayed. This option is not supported by all implementations; the traceroute utility is usually a better idea.

-s <packet-size>	Specifies the size of outgoing message to use.
-S <src-addr>	On devices that have multiple IP interfaces (addresses), allows a ping sent from one interface to use an address from one of the others.
-t <timeout>	Specifies a timeout period, in seconds, after which the *ping* utility will terminate, regardless of how many requests or replies have been sent or received.

Table 286: Common Windows *ping* Utility Options and Parameters

Option / Parameters	Description
-a	If the target device is specified as an IP address, force the address to be resolved to a DNS host name and displayed.
-f	Sets the Don't Fragment bit in the outgoing datagram.
-i <ttl-value>	Specifies the TTL value to be used for outgoing Echo messages.
-j <host-list>	Sends the outgoing messages using the specified loose source route.
-k <host-list>	Sends the outgoing messages using the indicated strict source route.
-l <buffer-size>	Specifies the size of the data field in the transmitted *Echo* messages.
-n <count>	Tells the utility how many *Echo* messages to send.
-r <count>	Specifies the use of the *Record Route* IP option and the number of hops to be recorded. As with the corresponding UNIX "-R" option, the traceroute utility is usually preferable.
-s <count>	Specifies the use of the IP *Timestamp* option to record the arrival time of the *Echo* and *Echo Reply* messages.
-t	Sends *Echo* messages continuously until the program is interrupted.
-w <timeout>	Specifies how long the program should wait for each *Echo Reply* before giving up, in milliseconds (default is 4000, for 4 seconds).

The ping6 Utility

The IPv6 version of *ping*, sometimes called *ping6*, works in very much the same way as IPv4 *ping*. The main differences between the two utilities are that *ping6*'s options and parameters reflect the changes made in addressing and routing in IPv6.

TCP/IP Route Tracing Utility (traceroute/tracert/traceroute6)

The *ping* utility described in the preceding topic is extremely helpful for checking whether or not two devices are able to talk to each other. However, it provides very little information regarding what is going on between those two devices. In the event that ping shows either a total inability to communicate or intermittent connectivity with high loss of transmitted data, we need to know more about what is happening to IP datagrams as they are carried across the internetwork. This is especially important when the two devices are far from each other, especially if we are trying to reach a server on the public Internet.

I described in my overview of IP datagram delivery that when two devices are not on the same network, data sent between them must be delivered from one network to the next until it reaches its destination. This means that any time data is sent from device *A* on one network to device *B* on another, it follows a *route*, which may not be the same for each transmission.

When communication problems arise, it is very useful to be able to check the specific route taken by data between two devices. A special route tracing utility is provided for this function, called *traceroute* (abbreviated *tracert* in Windows systems, a legacy of the old eight-character limit for DOS program names). The IPv6 equivalent of this program is called *traceroute6*.

Operation of the traceroute Utility

Like the *ping* utility, *traceroute* is implemented using ICMP messages. However, unlike *ping*, *traceroute* was originally not designed to use a special ICMP message type intended exclusively for route tracing. Instead, it makes clever use of the IP and ICMP features that are designed to prevent routing problems.

Recall that the IP datagram format includes a *Time To Live (TTL)* field. This field is set to the maximum number of times that a datagram may be forwarded before it must be discarded; it exists to prevent datagrams from circling an internetwork endlessly. If a datagram must be discarded due to expiration of the *TTL* field, the device that discards it is supposed to send back to the device that send that

datagram an ICMP *Time Exceeded* message. This is explained in much more detail in the topic that describes that message.

Under normal circumstances, this only occurs when there is a problem, such as a router loop or other misconfiguration issue. What *traceroute* does is to force each router in a route to report back to it by intentionally setting the *TTL* value in test datagrams to a value too low to allow them to reach their destination.

Suppose we have device *A* and device *B*, which are separated by routers *R1* and *R2*—three hops total. If you do a *traceroute* from device *A* to device *B*, here's what happens:

1. The *traceroute* utility sends a dummy UDP message (sometimes called a *probe*) to a port number that is intentionally selected to be invalid. The *TTL* field of the IP datagram is set to 1. When *R1* receives the message, it decrements the field, which will make its value 0. That router discards the probe and sends an ICMP *Time Exceeded* message back to device *A*.

2. Device *A* then sends a second UDP message with the *TTL* field set to 2. This time, *R1* reduces the *TTL* value to 1 and sends it to *R2*, which reduces the *TTL* field to 0 and sends a *Time Exceeded* message back to *A*.

3. Device *A* sends a third UDP message, with the *TTL* field set to 3. This time, the message will pass through both routers and be received by device *B*. However, since the port number was invalid, the message is rejected by device *B*, which sends back a *Destination Unreachable* message to device *A*.

This process is illustrated in Figure 321. Amusingly, we see that *A* sends out three messages to *B*, and gets back three error messages, and is happy about it! ☺ The route to device *B* is thus indicated by the identities of the devices sending back the error messages, in sequence. By keeping track of the time between when it sent each UDP message and received back the corresponding error message, the *traceroute* utility can also display how long it took to communicate with each device. In practice, usually three dummy messages are sent with each *TTL* value, so their transit times can be averaged by the user if desired.

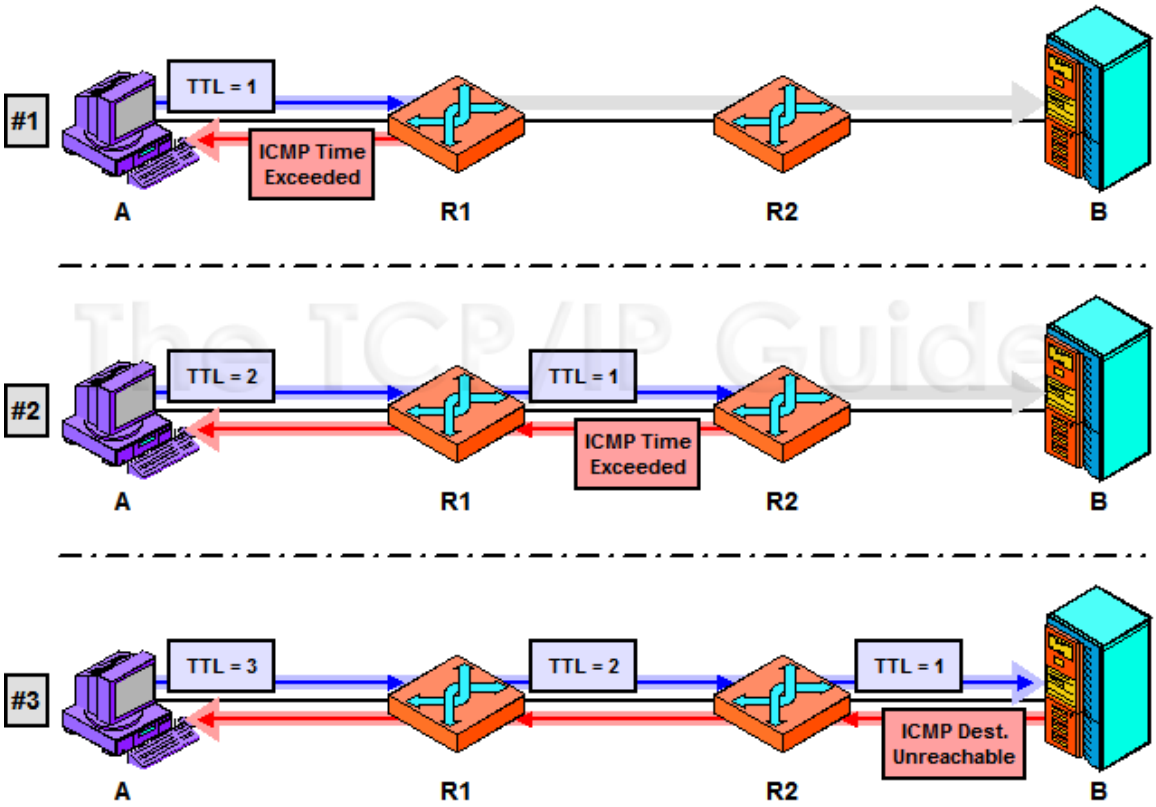

Figure 321: Operation of the traceroute/tracert Utility

The traceroute utility identifies the devices in a route by forcing them to report back failures to route datagrams with parameters intentionally set to invalid values. The first message sent by device *A* here has a *Time To Live (TTL)* value of 1, which will cause *R1* it to drop it and send an ICMP *Time Exceeded* message back to *A*. The second one has a *TTL* value of 2, so it will be dropped and reported by *R2*. The third will pass both routers and get to the destination host, *B*, but since the message is deliberately chosen with a bogus port number, this will cause an ICMP *Destination Unreachable* to be returned. These error messages identify the sequence of devices in the route between devices *A* and *B*.

Key Concept: The *traceroute* utility takes the idea behind *ping* one step further, allowing not only communication between two devices to be checked, but also letting an administrator see a list of all the intermediate devices between the pair. It works by having the initiating host send a series of test datagrams with TTL values that cause each to expire sequentially at each device on the route. The traceroute program also shows how much time it takes to communicate with each device between the sending host and a destination device.

Basic Use of the traceroute Utility

Table 287 shows an example of a *traceroute* sent between two of the UNIX computers I use on a regular basis. I added the "-q2" parameter to change the default of three dummy messages per hop to two, so the output would fit better in its display table. In this case, the servers are separated by 14 hops. Notice how the elapsed time generally increases as the distance from the transmitting device increases, but it is not consistent because of random elements in the delay between any two devices (see the incongruously-large value in hop #10, for example). Also notice the asterisk ("*") in the seventh hop, which means that no response was received before the timeout period for the second transmission with a *TTL* value of 7. Finally, there is no report at all for hop #13; this machine may have been configured not to send *Time Exceeded* messages.

Table 287: Route Tracing Using the *traceroute* Utility

```
traceroute -q2 www.pcguide.com
traceroute to www.pcguide.com (209.68.14.80), 40 hops max,
40 byte packets
1 cisco0fe0-0-1.bf.sover.net (209.198.87.10) 1.223 ms 1.143
ms
2 cisco1fe0.bf.sover.net (209.198.87.12) 1.265 ms 1.117 ms
3 cisco0a5-0-102.wnskvtao.sover.net (216.114.153.170) 8.004
ms 7.270 ms
4 207.136.212.234 (207.136.212.234) 7.163 ms 7.601 ms
5 sl-gw18-nyc-2-0.sprintlink.net (144.232.228.145) 15.948
ms 20.931 ms
6 sl-bb21-nyc-12-1.sprintlink.net (144.232.13.162) 21.578
ms 16.324 ms
7 sl-bb27-pen-12-0.sprintlink.net (144.232.20.97) 18.296 ms
*
8 sl-bb24-pen-15-0.sprintlink.net (144.232.16.81) 18.041 ms
18.338 ms
9 sl-bb26-rly-0-0.sprintlink.net (144.232.20.111) 20.259 ms
21.648 ms
10 sl-bb20-rly-12-0.sprintlink.net (144.232.7.249) 132.302
ms 37.825 ms
11 sl-gw9-rly-8-0.sprintlink.net (144.232.14.22) 23.085 ms
20.082 ms
12 sl-exped4-1-0.sprintlink.net (144.232.248.126) 43.374 ms
42.274 ms
13 * *
14 pcguide.com (209.68.14.80) 41.310 ms 49.455 ms
```

Additional “unusual” results may be displayed under certain circumstances. For example, the traceroute program may display a code such a “!H”, “!N” or “!P” to indicate receipt of an unexpected *Destination Unreachable* message for a host, network, or protocol, respectively. Other error messages may also exist, depending on the implementation.

Note: Not all *traceroute* utility implementations use the technique described above. Microsoft’s *tracert* works not by sending UDP packets but rather ICMP *Echo* messages with increasing TTL values. It knows it has reached the final host when it gets back an *Echo Reply* message. A special ICMP Traceroute message was also developed in 1993, which was intended to improve the efficiency of *traceroute* by eliminating the need to send many UDP messages for each route tracing. Despite its technical advantages, since it was introduced long after TCP/IP was widely deployed, it never became a formal Internet standard and is not seen as often as the traditional method.

traceroute Options and Parameters

As is the case with *ping*, *traceroute* can be used with an IP address or host name. If no parameters are supplied, default values will be used for key parameters; on the system I used, the defaults are three “probes” for each *TTL* value, a maximum of 64 hops tested, and packets 40 bytes in size. However, a number of options and parameters are also supported to give an administrator more control over how the utility functions (such as the “-q” parameter I used in Table 287). Some of the typical ones in UNIX systems are described in Table 288, while a smaller number of options exist in Windows, shown in Table 289.

Table 288: Common UNIX *traceroute* Utility Options and Parameters

Option / Parameters	Description
-g <host-list>	Specifies a source route to be used for the trace.
-M <initial-ttl-value>	Overrides the default value of 1 for the initial *TTL* value of the first outgoing probe message.
-m <max-ttl-value>	Sets the maximum *TTL* value to be used; this limits how long a route the utility will attempt to trace.
-n	Displays the route using numeric addresses only, rather than showing both IP addresses and host names. This speeds up the display by saving the utility from having to perform reverse DNS lookups on all the devices in the route (ICMP messages use IP

	addresses, not domain names.)
-p <port-number>	Specifies the port number to be used as the destination of the probe messages.
-q <queries>	Tells the utility how many probes to send to each device in the route (default is 3).
-r	Tells the program to bypass the normal routing tables and send directly to a host on an attached network.
-s <src-addr>	On devices that have multiple IP interfaces (addresses), allows the device to use an address from one interface on a traceroute using another interface.
-S	Instructs the program to display a summary of how many probes did not receive a reply.
-v	Sets verbose output mode, which informs the user of all ICMP messages received during the trace.
-w <wait-time>	Specifies how long the utility should wait for a reply to each probe, in seconds (typical default is 3 to 5).

Table 289: Common Windows *tracert* Utility Options and Parameters

Option / Parameters	Description
-d	Displays the route using numeric addresses only rather than showing both IP addresses and host names, for faster display. This is the same as the “-n” option on UNIX systems.
-h <maximum-hops>	Specifies the maximum number of hops to use for tracing; default is 30.
-j <host-list>	Sends the outgoing probes using the specified loose source route.
-w <wait-time>	Specifies how long to wait for a reply to each probe, in milliseconds (default is 4000, for 4 seconds).

The traceroute6 Utility

The *traceroute6* utility is the IPv6 version of *traceroute* and functions in a very similar manner to its IPv4 predecessor. It obviously uses IPv6 datagrams instead of IPv4 ones, and responses from traced devices are in the form of ICMPv6 Time

Exceeded and Destination Unreachable messages rather than their ICMPv4 counterparts.