2

RESOURCE LOCATION AND TRAFFIC ROUTING

To write effective network programs, you need to understand how to use human-readable names to identify nodes on the internet, how those names are translated into addresses for network devices to use, and how traffic makes its way between nodes on the internet, even if they're on opposite sides of the planet. This chapter covers those topics and more.

We'll first have a look at how IP addresses identify hosts on a network. Then we'll discuss *routing*, or sending traffic between network hosts that aren't directly connected, and cover some common routing protocols. Finally, we'll discuss *domain name resolution* (the process of translating human-readable names to IP addresses), potential privacy implications of DNS, and the solutions to overcome those privacy concerns.

You'll need to understand these topics to provide comprehensive network services and locate the resources used by your services, such as third-party application programming interfaces (APIs). This information should

also help you debug inevitable network outages or performance issues your code may encounter. For example, say you provide a service that integrates the Google Maps API to provide interactive maps and navigation. Your service would need to properly locate the API endpoint and route traffic to it. Or your service may need to store archives in an Amazon Simple Storage Service (S3) bucket via the Amazon S3 API. In each example, name resolution and routing play an integral role.

The Internet Protocol

The *Internet Protocol (IP)* is a set of rules that dictate the format of data sent over a network—specifically, the internet. *IP addresses* identify nodes on a network at the internet layer of the TCP/IP stack, and you use them to facilitate communication between nodes.

IP addresses function in the same way as postal addresses; nodes send packets to other nodes by addressing packets to the destination node's IP address. Just as it's customary to include a return address on postal mail, packet headers include the IP address of the origin node as well. Some protocols require an acknowledgment of successful delivery, and the destination node can use the origin node's IP address to send the delivery confirmation.

Two versions of IP addresses are in public use: IPv4 and IPv6. This chapter covers both.

IPv4 Addressing

IPv4 is the fourth version of the Internet Protocol. It was the first IP version in use on the internet's precursor, ARPANET, in 1983, and the predominant version in use today. IPv4 addresses are 32-bit numbers arranged in four groups of 8 bits (called *octets*) separated by decimal points.

NOTE *RFCs use the term* octet *as a disambiguation of the term* byte*, because a byte's storage size has historically been platform dependent.*

The total range of 32-bit numbers limits us to just over four billion possible IPv4 addresses. Figure 2-1 shows the binary and decimal representation of an IPv4 address.

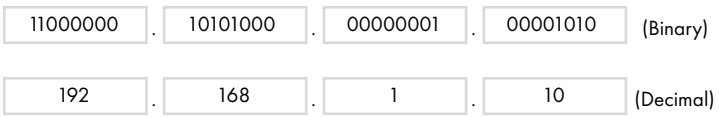

Figure 2-1: Four 8-bit octets representing an IPv4 address in both binary and decimal formats

The first line of Figure 2-1 illustrates an IPv4 address in binary form. The second line is the IPv4 address's decimal equivalent. We usually write IPv4 addresses in the more readable decimal format when displaying them or when using them in code. We will use their binary representation when we're discussing network addressing later in this section.

Network and Host IDs

The 32 bits that compose an IPv4 address represent two components: a network ID and a host ID. The *network ID* informs the network devices responsible for shuttling packets toward their destination about the next appropriate hop in the transmission. These devices are called *routers*. Routers are like the mail carrier of a network, in that they accept data from a device, examine the network ID of the destination address, and determine where the data needs to be sent to reach its destination. You can think of the network ID as a mailing address's ZIP code.

Once the data reaches the destination network, the router uses the *host ID* to deliver the data to the specific recipient. The host ID is like your street address. In other words, a network ID identifies a group of nodes whose address is part of the same network. We'll see what network and host IDs look like later in this chapter, but Figure 2-2 shows IPv4 addresses sharing the same network ID.

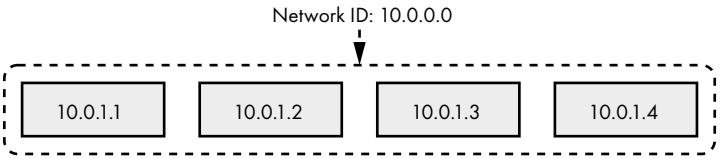

Figure 2-2: A group of nodes sharing the same network ID

Figure 2-3 shows the breakdown of common network ID and host ID sizes in a 32-bit IPv4 address.

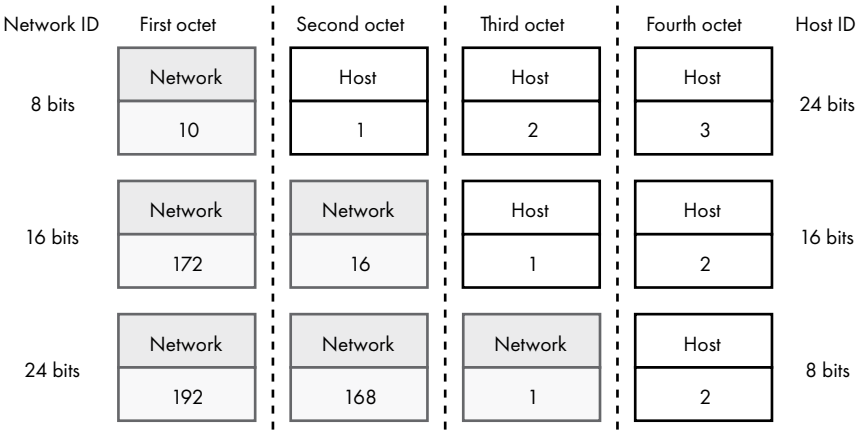

Figure 2-3: Common network ID and host ID sizes

The network ID portion of an IPv4 address always starts with the leftmost bit, and its size is determined by the size of the network it belongs to. The remaining bits designate the host ID. For example, the first 8 bits of the IPv4 address represent the network ID in an 8-bit network, and the remaining 24 bits represent the host ID.

Figure 2-4 shows the IP address 192.168.156.97 divided into its network ID and host ID. This IP address is part of a 16-bit network. This tells us that the first 16 bits form the network ID and the remaining 16 bits form the host ID.

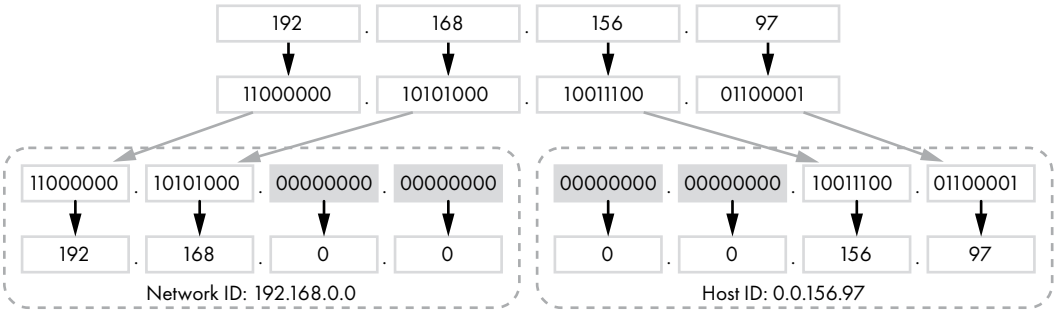

Figure 2-4: Deriving the network ID and the host ID from an IPv4 address in a 16-bit network

To derive the network ID for this example, you take the first 16 bits and append zeros for the remaining bits to produce the 32-bit network ID of 192.168.0.0. You prepend zeroed bits to the last 16 bits, resulting in the 32-bit host ID of 0.0.156.97.

Subdividing IPv4 Addresses into Subnets

IPv4's network and host IDs allow you to *subdivide*, or partition, the more than four billion IPv4 addresses into smaller groups to keep the network secure and easier to manage. All IP addresses in these smaller networks, called *subnets*, share the same network ID but have unique host IDs. The size of the network dictates the number of host IDs and, therefore, the number of individual IP addresses in the network.

Identifying individual networks allows you to control the flow of information between networks. For example, you could split your network into a subnet meant for public services and another for private services. You could then allow external traffic to reach your public services while preventing external traffic from reaching your private network. As another example, your bank provides services such as online banking, customer support, and mobile banking. These are public services that you interact with after successful authentication. But you don't have access to the bank's internal network, where its systems manage electronic transfers, balance ledgers, serve internal email, and so on. These services are restricted to the bank's employees via the private network.

Allocating Networks with CIDR

You allocate networks using a method known as *Classless Inter-Domain Routing (CIDR)*. In CIDR, you indicate the number of bits in the network ID by appending a *network prefix* to each IP address, consisting of a forward slash and an integer. Though it's appended to the end of the IP address,

you call it a *prefix* rather than a *suffix* because it indicates how many of the IP address's most significant bits, or prefixed bits, constitute the network ID. For example, you'd write the IP address 192.168.156.97 from Figure 2-4 as 192.168.156.97/16 in CIDR notation, indicating that it belongs to a 16-bit network and that the network ID is the first 16 bits of the IP address.

From there, you can derive the network IP address by applying a subnet mask. Subnet masks encode the CIDR network prefix in its decimal representation. They are applied to an IP address using a bitwise AND to derive the network ID.

Table 2-1 details the most common CIDR network prefixes, the corresponding subnet mask, the available networks for each network prefix, and the number of usable hosts in each network.

Table 2-1: CIDR Network Prefix Lengths and Their Corresponding Subnet Masks

CIDR network prefix length	Subnet mask	Available networks	Usable hosts per network
8	255.0.0.0	1	16,777,214
9	255.128.0.0	2	8,388,606
10	255.192.0.0	4	4,194,302
11	255.224.0.0	8	2,097,150
12	255.240.0.0	16	1,048,574
13	255.248.0.0	32	524,286
14	255.252.0.0	64	262,142
15	255.254.0.0	128	131,070
16	255.255.0.0	256	65,534
17	255.255.128.0	512	32,766
18	255.255.192.0	1,024	16,382
19	255.255.224.0	2,048	8,190
20	255.255.240.0	4,096	4,094
21	255.255.248.0	8,192	2,046
22	255.255.252.0	16,384	1,022
23	255.255.254.0	32,768	510
24	255.255.255.0	65,536	254
25	255.255.255.128	131,072	126
26	255.255.255.192	262,144	62
27	255.255.255.224	524,288	30
28	255.255.255.240	1,048,576	14
29	255.255.255.248	2,097,152	6
30	255.255.255.252	4,194,304	2

You may have noticed that the number of usable hosts per network is two less than expected in each row because each network has two special addresses. The first IP address in the network is the network address, and the last IP address is the broadcast address. (We'll cover broadcast addresses a bit later in this chapter.) Take 192.168.0.0/16, for example. The first IP address in the network is 192.168.0.0. This is the network address. The last IP address in the network is 192.168.255.255, which is the broadcast address. For now, understand that you do not assign the network IP address or the broadcast IP address to a host's network interface. These special IP addresses are used for routing data between networks and broadcasting, respectively.

The 31- and 32-bit network prefixes are purposefully absent from Table 2-1, largely because they are beyond the scope of this book. If you're curious about 31-bit network prefixes, RFC 3021 covers their application. A 32-bit network prefix signifies a single-host network. For example, 192.168.1.1/32 represents a subnetwork of one node with the address 192.168.1.1.

Allocating Networks That Don't Break at an Octet Boundary

Some network prefixes don't break at an octet boundary. For example, Figure 2-5 derives the network ID and host ID of 192.168.156.97 in a 19-bit network. The full IP address in CIDR notation is 192.168.156.97/19.

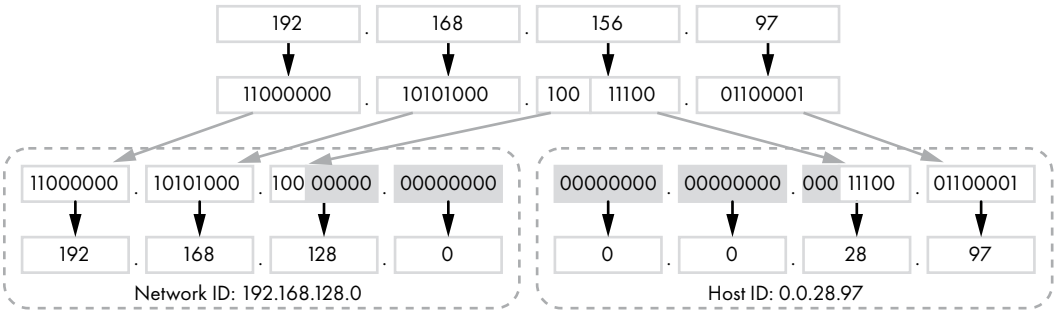

Figure 2-5: Deriving the network ID and the host ID from the IPv4 address in a 19-bit network

In this case, since the network prefix isn't a multiple of 8 bits, an octet's bits are split between the network ID and host ID. The 19-bit network example in Figure 2-5 results in the network ID of 192.168.128.0 and the host ID of 0.0.28.97, where the network ID borrows 3 bits from the third octet, leaving 13 bits for the host ID.

Appending a zeroed host ID to the network ID results in the network address. In a comparable manner, appending a host ID in which all its bits are 1 to the network ID derives the broadcast address. But the third octet's equaling 156 can be a little confusing. Let's focus on just the third octet. The third octet of the network ID is 1000 0000. The third octet of the host ID of all ones is 0001 1111 (the first 3 bits are part of the network ID, remember). If we append the network ID's third octet to the host ID's third octet, the result is 1001 1111, which is the decimal 156.

Private Address Spaces and Localhost

RFC 1918 details the private address spaces of 10.0.0.0/8, 172.16.0.0/12, and 192.168.0.0/16 for use in local networks. Universities, corporations, governments, and residential networks can use these subnets for local addressing.

In addition, each host has the 127.0.0.0/8 subnet designated as its local subnet. Addresses in this subnet are local to the host and simply called *localhost*. Even if your computer is not on a network, it should still have an address on the 127.0.0.0/8 subnet, most likely 127.0.0.1.

Ports and Socket Addresses

If your computer were able to communicate over the network with only one node at a time, that wouldn't provide a very efficient or pleasant experience. It would become annoying if your streaming music stopped every time you clicked a link in your web browser because the browser needed to interrupt the stream to retrieve the requested web page. Thankfully, TCP and UDP allow us to multiplex data transmissions by using *ports*.

The operating system uses ports to uniquely identify data transmission between nodes for the purposes of multiplexing the outgoing application data and demultiplexing the incoming data back to the proper application. The combination of an IP address and a port number is a *socket address*, typically written in the format *address:port*.

Ports are 16-bit unsigned integers. Port numbers 0 to1023 are well-known ports assigned to common services by the *Internet Assigned Numbers Authority (IANA)*. The IANA is a private US nonprofit organization that globally allocates IP addresses and port numbers. For example, HTTP uses port 80. Port 443 is the HTTPS port. SSH servers typically listen on port 22. (These well-known ports are guidelines. An HTTP server may listen to any port, not just port 80.)

Despite these ports being well-known, there is no restriction on which ports services may use. For example, an administrator who wants to obscure a service from attackers expecting it on port 22 may configure an SSH server to listen on port 22422. The IANA designates ports 1024 to 49151 as semi-reserved for lesser common services. Ports 49152 to 65535 are ephemeral ports meant for client socket addresses as recommended by the IANA. (The port range used for client socket addresses is operating-system dependent.)

A common example of port usage is the interaction between your web browser and a web server. Your web browser opens a socket with the operating system, which assigns an address to the socket. Your web browser sends a request through the socket to port 80 on the web server. The web server sends its response to the socket address corresponding to the socket your web browser is monitoring. Your operating system receives the response and passes it onto your web browser through the socket. Your web browser's socket address and the web server's socket address (server IP and port 80) uniquely identify this transaction. This allows your operating system to properly demultiplex the response and pass it along to the right application (that is, your web browser).

Network Address Translation

The four billion IPv4 addresses may seem like a lot until you consider there will be an estimated 24.6 billion Internet of Things (IoT) devices by 2025, according to the Ericsson Mobility Report of June 2020 (*https://www.ericsson.com/en/mobility-report/reports/june-2020/iot-connections-outlook/*). In fact, we've already run out of unreserved IPv4 addresses. The IANA allocated the last IPv4 address block on January 31, 2011.

One way to address the IPv4 shortage is by using *network address translation (NAT)*, a process that allows numerous nodes to share the same public IPv4 address. It requires a device, such as a firewall, load balancer, or router that can keep track of incoming and outgoing traffic and properly route incoming traffic to the correct node.

Figure 2-6 illustrates the NAT process between nodes on a private network and the internet.

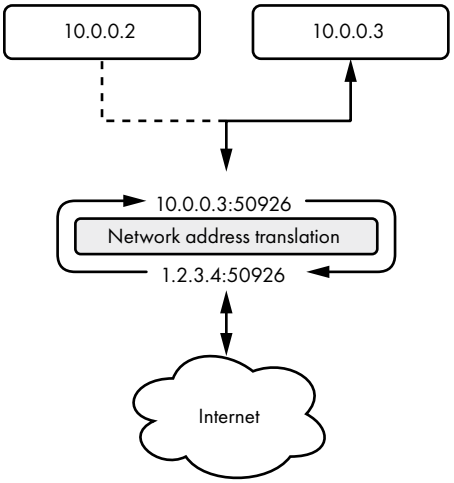

Figure 2-6: Network address translation between a private network and the internet

In Figure 2-6, a NAT-capable device receives a connection from the client socket address 10.0.0.3:50926 destined for a host on the internet. First, the NAT device opens its own connection to the destination host using its public IP 1.2.3.4, preserving the client's socket address port. Its socket address for this transaction is 1.2.3.4:50926. If a client is already using port 50926, the NAT device chooses a random port for its socket address. Then, the NAT device sends the request to the destination host and receives the response on its 1.2.3.4:50926 socket. The NAT device knows which client receives the response because it translates its socket address to the client socket address that established the connection. Finally, the client receives the destination host's response from the NAT device.

The important thing to remember with network address translation is that a node's private IPv4 address behind a NAT device is not visible or directly accessible to other nodes outside the network address–translated

network segment. If you're writing a service that needs to provide a public address for its clients, you may not be able to rely on your node's private IPv4 address if it's behind a NAT device. Hosts outside the NAT device's private network cannot establish incoming connections. Only clients in the private network may establish connections through the NAT device. Instead, your service must rely on the NAT device's properly forwarding a port from its public IP to a socket address on your node.

Unicasting, Multicasting, and Broadcasting

Sending packets from one IP address to another IP address is known as *unicast addressing*. But TCP/IP's internet layer supports IP *multicasting*, or sending a single message to a group of nodes. You can think of it as an opt-in mailing list, such as a newspaper subscription.

From a network programming perspective, multicasting is simple. Routers and switches typically replicate the message for us, as shown in Figure 2-7. We'll discuss multicasting later in this book.

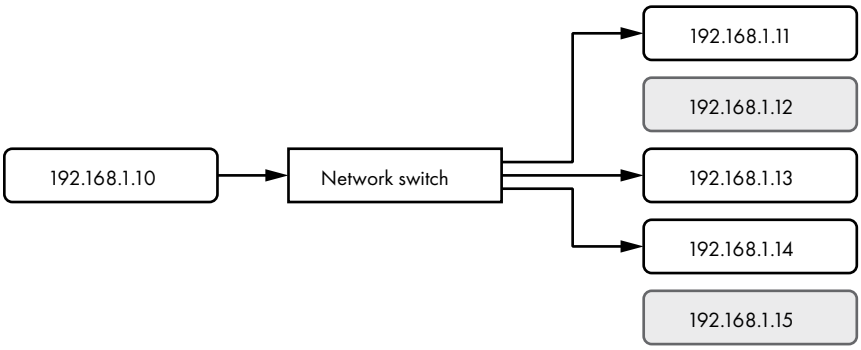

Figure 2-7: The 192.168.1.10 node sending a packet to a subset of network addresses

Broadcasting is the ability to concurrently deliver a message to all IP addresses in a network. To do this, nodes on a network send packets to the *broadcast address* of a subnet. A network switch or router then propagates the packets out to all IPv4 addresses in the subnet (Figure 2-8).

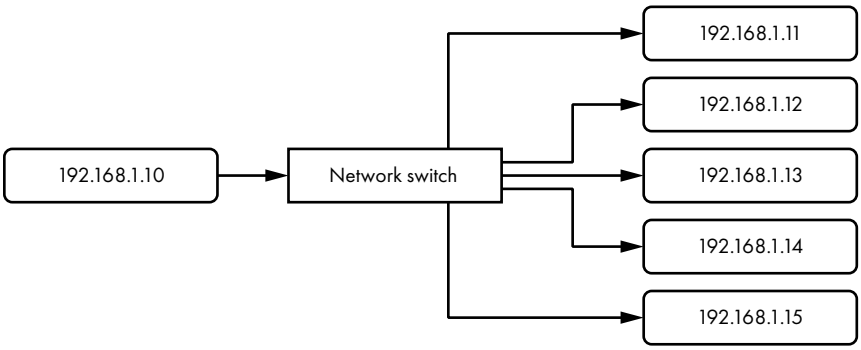

Figure 2-8: The 192.168.1.10 node sending a packet to all addresses on its subnet

Unlike multicasting, the nodes in the subnet don't first need to opt in to receiving broadcast messages. If the node at 192.168.1.10 in Figure 2-8 sends a packet to the broadcast address of its subnet, the network switch will deliver a copy of that packet to the other five IPv4 addresses in the same subnet.

Resolving the MAC Address to a Physical Network Connection

Recall from Chapter 1 that every network interface has a MAC address uniquely identifying the node's physical connection to the network. The MAC address is relevant to only the local network, so routers cannot use a MAC address to route data across network boundaries. Instead, they can route traffic across network boundaries by using an IPv4 address. Once a packet reaches the local network of a destination node, the router sends the data to the destination node's MAC address and, finally, to the destination node's physical network connection.

The *Address Resolution Protocol (ARP)*, detailed in RFC 826 (*https://tools.ietf.org/html/rfc826/*), finds the appropriate MAC address for a given IP address—a process called *resolving* the MAC address. Nodes maintain ARP tables that map an IPv4 address to a MAC address. If a node does not have an entry in its ARP table for a destination IPv4 address, the node will send a request to the local network's broadcast address asking, "Who has this IPv4 address? Please send me your MAC address. Oh, and here is my MAC address." The destination node will receive the ARP request and respond with an ARP reply to the originating node. The originating node will then send the data to the destination node's MAC address. Nodes on the network privy to this conversation will typically update their ARP tables with the values.

IPv6 Addressing

Another solution to the IPv4 shortage is to migrate to the next generation of IP addressing, IPv6. *IPv6 addresses* are 128-bit numbers arranged in eight colon-separated groups of 16 bits, or *hextets*. There are more than 340 undecillion (2^{128}) IPv6 addresses.

Writing IPv6 Addresses

In binary form, IPv6 addresses are a bit ridiculous to write. In the interest of legibility and compactness, we write IPv6 addresses with lowercase hexadecimal values instead.

NOTE *IPv6 hexadecimal values are case-insensitive. However, the Internet Engineering Task Force (IETF) recommends using lowercase values.*

A hexadecimal (hex) digit represents 4 bits, or a *nibble*, of an IPv6 address. For example, we'd represent the two nibbles 1111 1111 in their hexadecimal equivalent of `ff`. Figure 2-9 illustrates the same IPv6 address in binary and hex.

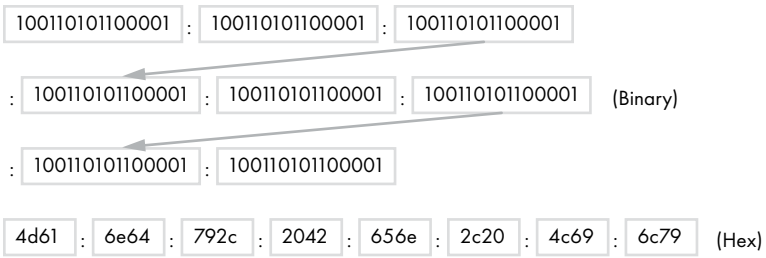

Figure 2-9: Binary and hex representations of the same IPv6 address

Even though hexadecimal IPv6 addresses are a bit more succinct than their binary equivalent, we still have some techniques available to us to simplify them a bit more.

Simplifying IPv6 Addresses

An IPv6 address looks something like this: fd00:4700:0010:0000:0000:0000:6814:d103. That's quite a bit harder to remember than an IPv4 address. Thankfully, you can improve the IPv6 address's presentation to make it more readable by following a few rules.

First, you can remove all leading zeros in each hextet. This simplifies your address without changing its value. It now looks like this: fd00:4700:10:0:0:0:6814:d103. Better, but still long.

Second, you can replace the leftmost group of consecutive, zero-value hextets with double colons, producing the shorter fd00:4700:10::6814:d103. If your address has more than one group of consecutive zero-value hextets, you can remove only the leftmost group. Otherwise, it's impossible for routers to accurately determine the number of hextets to insert when repopulating the full address from its compressed representation. For example, fd00:4700:0000:0000:ef81:0000:6814:d103 rewrites to fd00:4700::ef81:0:6814:d103. The best you could do with the sixth hextet is to remove the leading zeros.

IPv6 Network and Host Addresses

Like IPv4 addresses, IPv6 addresses have a network address and a host address. IPv6's host address is commonly known as the *interface ID*. The network and host addresses are both 64 bits, as shown in Figure 2-10. The first 48 bits of the network address are known as the *global routing prefix (GRP)*, and the last 16 bits of the network address are called the *subnet ID*. The 48-bit GRP is used for globally subdividing the IPv6 address space and routing traffic between these groups. The subnet ID is used to further subdivide each GRP-unique network into site-specific networks. If you run a large ISP, you are assigned one or more GRP-unique blocks of IPv6 addresses. You can then use the subnet ID in each network to further subdivide your allocated IPv6 addresses to your customers.

The GRP gets determined for you when you request a block of IPv6 addresses from your ISP. IANA assigns the first hextet of the GRP to a regional internet registry (an organization that handles the allocation of addresses for a global region). The regional internet registry then assigns the second GRP hextet to an ISP. The ISP finally assigns the third GRP hextet before assigning a 48-bit subnet of IPv6 addresses to you.

NOTE *For more information on the allocation of IPv6 addresses, see IANA's "IPv6 Global Unicast Address Assignments" document at* https://www.iana.org/assignments/ipv6-unicast-address-assignments/ipv6-unicast-address-assignments.xml.

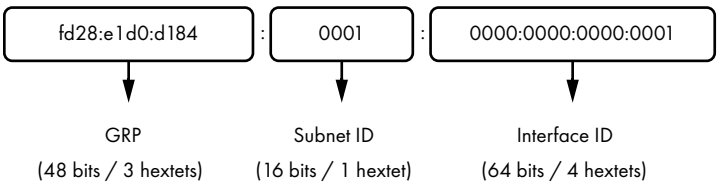

Figure 2-10: IPv6 global routing prefix, subnet ID, and interface ID

The first hextet of an IPv6 address gives you a clue to its use. Addresses beginning with the prefix 2000::/3 are meant for global use, meaning every node on the internet will have an IPv6 address starting with 2 or 3 in the first hex. The prefix fc00::/7 designates a unique local address like the 127.0.0.0/8 subnet in IPv4.

NOTE *IANA's "Internet Protocol Version 6 Address Space" document at* https://www.iana.org/assignments/ipv6-address-space/ipv6-address-space.xhtml *provides more details.*

Let's assume your ISP assigned the 2600:fe56:7891::/48 netblock to you. Your 16-bit subnet ID allows you to further subdivide your netblock into a maximum of 65,536 (2^{16}) subnets. Each of those subnets supports over 18 quintillion (2^{64}) hosts. If you assign 1 to the subnet as shown in Figure 2-10, you'd write the full network address as 2600:fe56:7891:1::/64 after removing leading zeros and compressing zero value hextets. Further subnetting your netblock may look like this: 2600:fe56:7891:**2**::/64, 2600:fe56:7891:**3**::/64, 2600:fe56:7891:**4**::/64.

IPv6 Address Categories

IPv6 addresses are divided into three categories: anycast, multicast, and unicast. Notice there is no broadcast type, as in IPv4. As you'll see, anycast and multicast addresses fulfill that role in IPv6.

Unicast Addresses

A *unicast* IPv6 address uniquely identifies a node. If an originating node sends a message to a unicast address, only the node with that address will receive the message, as shown in Figure 2-11.

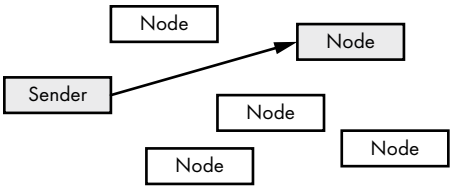

Figure 2-11: Sending to a unicast address

Multicast Addresses

Multicast addresses represent a group of nodes. Whereas IPv4 broadcast addresses will propagate a message out to all addresses on the network, multicast addresses will simultaneously deliver a message to a subset of network addresses, not necessarily all of them, as shown in Figure 2-12.

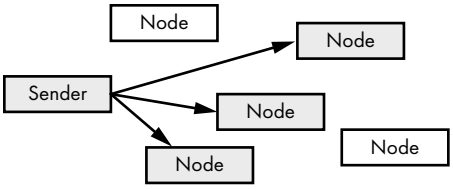

Figure 2-12: Sending to a multicast address

Multicast addresses use the prefix ff00::/8.

Anycast Addresses

Remember that IPv4 addresses must be unique per network segment, or network communication issues can occur. But IPv6 includes support for multiple nodes using the same network address. An *anycast* address represents a group of nodes listening to the same address. A message sent to an anycast address goes to the nearest node listening to the address. Figure 2-13 represents a group of nodes listening to the same address, where the nearest node to the sender receives the message. The sender could transmit to any of the nodes represented by the dotted lines, but sends to the nearest node (solid line).

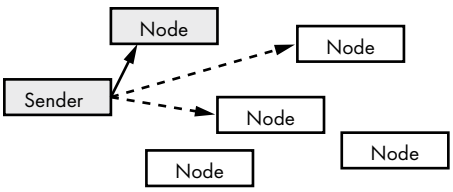

Figure 2-13: Sending to an anycast address

The nearest node isn't always the most physically close node. It is up to the router to determine which node receives the message, usually the node

with the least latency between the origin and the destination. Aside from reducing latency, anycast addressing increases redundancy and can geolocate services.

Sending traffic around the world takes a noticeable amount of time, to the point that the closer you are to a service provider's servers, the better performance you'll experience. Geolocating services across the internet is a common method of placing servers geographically close to users to make sure performance is optimal for all users across the globe. It's unlikely you access servers across an ocean when streaming Netflix. Instead, Netflix geolocates servers close to you so that your experience is ideal.

Advantages of IPv6 Over IPv4

Aside from the ridiculously large address space, IPv6 has inherent advantages over IPv4, particularly with regard to efficiency, autoconfiguration, and security.

Simplified Header Format for More Efficient Routing

The IPv6 header is an improvement over the IPv4 header. The IPv4 header contains mandatory yet rarely used fields. IPv6 makes these fields optional. The IPv6 header is extensible, in that functionality can be added without breaking backward compatibility. In addition, the IPv6 header is designed for improved efficiency and reduced complexity over the IPv4 header.

IPv6 also lessens the loads on routers and other hops by ensuring that headers require minimal processing, eliminating the need for checksum calculation at every hop.

Stateless Address Autoconfiguration

Administrators manually assign IPv4 addresses to each node on a network or rely on a service to dynamically assign addresses. Nodes using IPv6 can automatically configure or derive their IPv6 addresses through *stateless address autoconfiguration (SLAAC)* to reduce administrative overhead.

When connected to an IPv6 network, a node can solicit the router for its network address parameters using the *Neighbor Discovery Protocol (NDP)*. NDP leverages the Internet Control Message Protocol, discussed later in this chapter, for router solicitation. It performs the same duties as IPv4's ARP. Once the node receives a reply from the router with the 64-bit network address, the node can derive the 64-bit host portion of its IPv6 address on its own using the 48-bit MAC address assigned to its network interface. The node appends the 16-bit hex FFFE to the first three octets of the MAC address known as the *originally unique identifier*. To this, the node appends the remaining three octets of the MAC address, the network interface controller (NIC) identifier. The result is a unique 64-bit interface ID, as shown in Figure 2-14. SLAAC works only in the presence of a router that can respond with router advertisement packets. *Router advertisement packets* contain information clients need to automatically configure their IPv6 address, including the 64-bit network address.

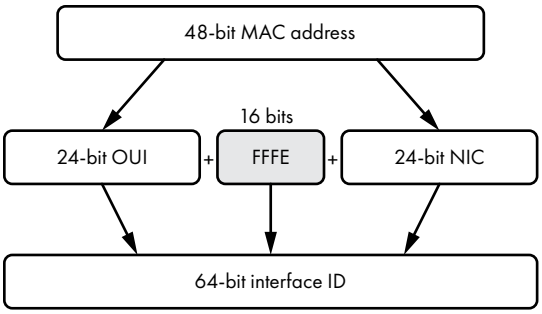

Figure 2-14: Deriving the interface ID from the MAC address

If you value your privacy, the method SLAAC uses to derive a unique interface ID should concern you. No matter which network your device is on, SLAAC will make sure the host portion of your IPv6 address contains your NIC's MAC address. The MAC address is a unique fingerprint that betrays the hardware you use and allows anyone to track your online activity. Thankfully, many people raised these concerns, and SLAAC gained privacy extensions (*https://tools.ietf.org/html/rfc4941/*), which randomize the interface ID. Because of this randomization, it's possible for more than one node on a network to generate the same interface ID. Thankfully, the NDP will automatically detect and fix any duplicate interface ID for you.

Native IPsec Support

IPv6 has native support for *IPsec*, a technology that allows multiple nodes to dynamically create secure connections between each other, ensuring that traffic is encrypted.

NOTE *RFC 6434 made IPsec optional for IPv6 implementations.*

The Internet Control Message Protocol

The Internet Protocol relies on the *Internet Control Message Protocol (ICMP)* to give it feedback about the local network. ICMP can inform you of network problems, unreachable nodes or networks, local network configuration, proper traffic routes, and network time-outs. Both IPv4 and IPv6 have their own ICMP implementations, designated ICMPv4 and ICMPv6, respectively.

Network events often result in ICMP response messages. For instance, if you attempt to send data to an unreachable node, a router will typically respond with an ICMP *destination unreachable* message informing you that your data couldn't reach the destination node. A node may become unreachable if it runs out of resources and can no longer respond to incoming data or if data cannot route to the node. Disconnecting a node from a network will immediately make it unreachable.

Routers use ICMP to help inform you of better routes to your destination node. If you send data to a router that isn't the appropriate or best

router to handle traffic for your destination, it may reply with an ICMP *redirect* message after forwarding your data onto the correct router. The ICMP redirect message is the router's way of telling you to send your data to the appropriate router in the future.

You can determine whether a node is online and reachable by using an ICMP *echo* request (also called a *ping*). If the destination is reachable and receives your ping, it will reply with its own ICMP *echo reply* message (also called a *pong*). If the destination isn't reachable, the router will respond with a destination unreachable message.

ICMP can also notify you when data reaches the end of its life before delivery. Every IP packet has a *time-to-live* value that dictates the maximum number of hops the packet can take before its lifetime expires. The packet's time-to-live value is a counter and decrements by one for every hop it takes. You will receive an ICMP *time exceeded* message if the packet you sent doesn't reach its destination by the time its time-to-live value reaches zero.

IPv6's NDP relies heavily on ICMP router solicitation messages to properly configure a node's NIC.

Internet Traffic Routing

Now that you know a bit about internet protocol addressing, let's explore how packets make their way across the internet from one node to another using those addresses. In Chapter 1, we discussed how data travels down the network stack of the originating node, across a physical medium, and up the stack of the destination node. But in most cases, nodes won't have a direct connection, so they'll have to make use of intermediate nodes to transfer data. Figure 2-15 shows that process.

The intermediate nodes (Nodes 1 and 2 in Figure 2-15) are typically routers or firewalls that control the path data takes from one node to the other. *Firewalls* control the flow of traffic in and out of a network, primarily to secure networks behind the firewall.

No matter what type of node they are, intermediate nodes have a network stack associated with each network interface. In Figure 2-15, Node 1 receives data on its incoming network interface. The data climbs the stack to Layer 3, where it's handed off to the outgoing network interface's stack. The data then makes its way to Node 2's incoming network interface before ultimately being routed to the server.

The incoming and outgoing network interfaces in Node 1 and Node 2 may send data over different media types using IPv4, so they must use encapsulation to isolate the implementation details of each media type from the data being sent. Let's assume Node 1 receives data from the client over a wireless network and it sends data to Node 2 over an Ethernet connection. Node 1's incoming Layer 1 knows how to convert the radio signals from the wireless network into bits. Layer 1 sends the bits up to Layer 2. Layer 2 converts the bits to a frame and extracts the packet and sends it up to Layer 3.

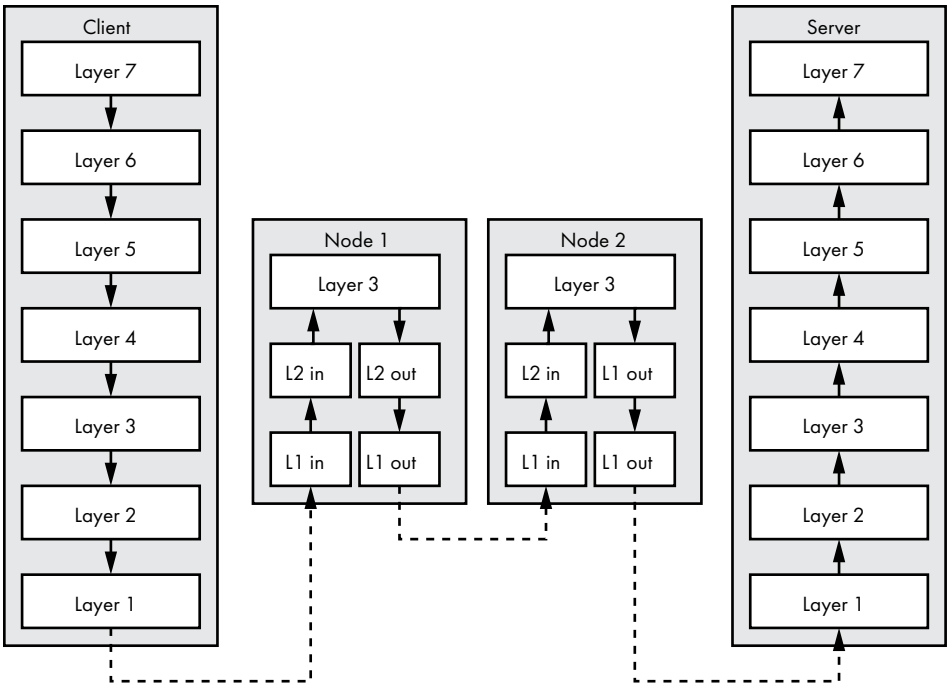

Figure 2-15: Routing packets through two hops

Layer 3 on both the incoming and outgoing NICs speak IPv4, which routes the packet between the two interface network stacks. The outgoing NIC's Layer 2 receives the packet from its Layer 3 and encapsulates it before sending the frame onto its Layer 1 as bits. The outgoing Layer 1 converts the bits into electric signals suitable for transmission over Ethernet. The data in transport from the client's Layer 7 never changed despite the data's traversing multiple nodes over different media on its way to the destination server.

Routing Protocols

The routing overview in Figure 2-15 makes the process look easy, but the routing process relies on a symphony of protocols to make sure each packet reaches its destination no matter the physical medium traversed or network outages along the way. Routing protocols have their own criteria for determining the best path between nodes. Some protocols determine a route's efficiency based on hop count. Some may use bandwidth. Others may use more complicated means to determine which route is the most efficient.

Routing protocols are either internal or external depending on whether they route packets within an autonomous system or outside of one. An *autonomous system* is an organization that manages one or more networks. An ISP is an example of an autonomous system. Each autonomous system is assigned an autonomous system number (ASN), as outlined in RFC 1930 (*https://tools.ietf.org/html/rfc1930/*). This ASN is used to broadcast an ISP's network

information to other autonomous systems using an external routing protocol. An *external routing protocol* routes data between autonomous systems. The only routing protocol we'll cover is BGP since it is the glue of the internet, binding all ASN-assigned ISPs together. You don't need to understand BGP in depth, but being familiar with it can help you better debug network issues related to your code and improve your code's resiliency.

The Border Gateway Protocol

The *Border Gateway Protocol (BGP)* allows ASN-assigned ISPs to exchange routing information. BGP relies on trust between ISPs. That is, if an ISP says it manages a specific network and all traffic destined for that network should be sent to it, the other ISPs trust this claim and send traffic accordingly. As a result, BGP misconfigurations, or *route leaks*, often result in very public network outages.

In 2008, Pakistan Telecommunications Company effectively took down YouTube worldwide after the Pakistani Ministry of Communications demanded the country block *youtube.com* in protest of a YouTube video. Pakistan Telecom used BGP to send all requests destined for YouTube to a null route, a route that drops all data without notification to the sender. But Pakistan Telecom accidentally leaked its BGP route to the world instead of restricting it to the country. Other ISPs trusted the update and null routed YouTube requests from their clients, making *youtube.com* inaccessible for two hours all over the world.

In 2012, Google's services were rerouted through Indonesia for 27 minutes when the ISP Moratel shared a BGP route directing all Google traffic to Moratel's network as if Moratel was now hosting Google's network infrastructure. There was speculation at the time that the route leakage was malicious, but Moratel blamed a hardware failure.

BGP usually makes news only when something goes wrong. Other times, it plays the silent hero, serving a significant role in mitigating distributed denial-of-service (DDOS) attacks. In a *DDOS attack*, a malicious actor directs traffic from thousands of compromised nodes to a victim node with the aim of overwhelming the victim and consuming all its bandwidth, effectively denying service to legitimate clients. Companies that specialize in mitigating DDOS attacks use BGP to reroute all traffic destined for the victim node to their AS networks, filter out the malicious traffic from the legitimate traffic, and route the sanitized traffic back to the victim, nullifying the effects of the attack.

Name and Address Resolution

The *Domain Name System (DNS)* is a way of matching IP addresses to *domain names*, which are the names we enter in an address bar when we want to visit websites. Although the internet protocol uses IP addresses to locate hosts, domain names (like *google.com*) are easier for humans to understand and remember. If I gave you the IP address 172.217.6.14 to visit, you wouldn't know who owned that IP address or what I was directing you to visit. But if

I gave you *google.com* instead, you'd know exactly where I was sending you. DNS allows you to remember a hostname instead of its IP address in the same way that your smartphone's contact list frees you from having to memorize all those phone numbers.

All domains are children of a *top-level domain*, such as *.com*, *.net*, *.org*, and so on. Take *nostarch.com*, for instance. No Starch Press registered the *nostarch* domain on the *.com* top-level domain from a registrar with the authority from IANA to register *.com* domains. No Starch Press now has the exclusive authority to manage DNS records for *nostarch.com* and publish records on its DNS server. This includes the ability for No Starch Press to publish *subdomains*—a subdivision of a domain—under its domain. For example, *maps.google.com* is a subdomain of *google.com*. A longer example is *sub3.sub2.sub1.domain.com*, where *sub3* is a subdomain under *sub2.sub1.domain.com*, *sub2* is subdomain under *sub1.domain.com*, and *sub1* is a subdomain under *domain.com*.

If you enter `https://nostarch.com` in your web browser, your computer will consult its configured *domain name resolver*, a server that knows how to retrieve the answer to your query. The resolver will start by asking one of the 13 IANA-maintained root name servers for the IP address of *nostarch.com*. The root name server will examine the top-level domain of the domain you requested and give your resolver the address of the *.com* name server. Your resolver will then ask the *.com* name server for *nostarch.com*'s IP address, which will examine the domain portion and direct your resolver to ask No Starch Press's name server. Finally, your resolver will ask No Starch Press's name server and receive the IP address that corresponds to *nostarch.com*. Your web browser will establish a connection to this IP address, retrieve the web page, and render it for you. This hierarchical journey of domain resolution allows you to zero in on a specific web server, and all you had to know was the domain name. No Starch Press is free to move its servers to a different ISP with new IP addresses, and yet you'll still be able to visit its website by using *nostarch.com*.

Domain Name Resource Records

Domain name servers maintain *resource records* for the domains they serve. Resource records contain domain-specific information, used to satisfy domain name queries, like IP addresses, mail server hostnames, mail-handling rules, and authentication tokens. There are many resource records, but this section focuses on only the most common ones: address records, start-of-authority records, name server records, canonical name records, mail exchange records, pointer records, and text records.

NOTE *For more details on types of resource records, see Wikipedia's entry at* https://en.wikipedia.org/wiki/List_of_DNS_record_types.

Our exploration of each resource record will use a utility called *dig* to query domain name servers. This utility may be available on your operating system, but in case you don't have dig installed, you can use the G Suite Toolbox Dig utility (*https://toolbox.googleapps.com/apps/dig/*) in a web browser

and receive similar output. All domain names you'll see are *fully qualified*, which means they end in a period, displaying the domain's entire hierarchy from the root zone. The *root zone* is the top DNS namespace.

Dig's default output includes a bit of data relevant to your query but irrelevant to your study of its output. Therefore, I've elected to snip out header and footer information in dig's output in each example to follow. Also please be aware that the specific output in this book is a snapshot from when I executed each query. It may look different when you execute these commands.

The Address Record

The *Address (A) record* is the most common record you'll query. An A record will resolve to one or more IPv4 addresses. When your computer asks its resolver to retrieve the IP address for *nostarch.com*, the resolver ultimately asks the domain name server for the *nostarch.com* Address (A) resource record. Listing 2-1 shows the question and answer sections when you query for the *google.com* A record.

```
$ dig google.com. a
-- snip --
❶ ;QUESTION
❷ google.com. ❸IN ❹A
❺ ;ANSWER
❻ google.com. ❼299 IN A ❽172.217.4.46
-- snip --
```

Listing 2-1: DNS answer of the google.com *A resource record*

Each section in a DNS reply begins with a header ❶, prefixed with a semicolon to indicate that the line is a comment rather than code to be processed. Within the question section, you ask the domain name server for the domain name *google.com* ❷ with the class `IN` ❸, which indicates that this record is internet related. You also use `A` to ask for the A record ❹ specifically.

In the Answer section ❺, the domain name server resolves the *google.com* A record to six IPv4 addresses. The first field of each returned line is the domain name ❻ you queried. The second field is the TTL value ❼ for the record. The *TTL value* tells domain name resolvers how long to cache or remember the record, and it lets you know how long you have until the cached record expires. When you request a DNS record, the domain name resolver will first check its cache. If the answer is in its cache, it will simply return the cached answer instead of asking the domain name server for the answer. This improves domain name resolution performance for records that are unlikely to change frequently. In this example, the record will expire in 299 seconds. The last field is the IPv4 address ❽. Your web browser could use any one of the six IPv4 addresses to establish a connection to *google.com*.

The AAAA resource record is the IPv6 equivalent of the A record.

The Start of Authority Record

The *Start of Authority (SOA) record* contains authoritative and administrative details about the domain, as shown in Listing 2-2. All domains must have an SOA record.

```
$ dig google.com. soa
-- snip --
;QUESTION
google.com. IN SOA
;ANSWER
google.com. 59 IN SOA ❶ns1.google.com. ❷dns-admin.google.com. ❸248440550
900 900 1800 60
-- snip --
```

Listing 2-2: DNS answer of the google.com *SOA resource record*

The first four fields of an SOA record are the same as those found in an A record. The SOA record also includes the primary name server ❶, the administrator's email address ❷, and fields ❸ used by secondary name servers outside the scope of this book. Domain name servers primarily consume SOA records. However, the email address is useful if you wish to contact the domain's administrator.

NOTE *The administrator's email address is encoded as a name, with the at sign (@) replaced by a period.*

The Name Server Record

The *Name Server (NS) record* returns the authoritative name servers for the domain name. *Authoritative name servers* are the name servers able to provide answers for the domain name. NS records will include the primary name server from the SOA record and any secondary name servers answering DNS queries for the domain. Listing 2-3 is an example of the NS records for *google.com*.

```
$ dig google.com. ns
-- snip --
;QUESTION
google.com. IN NS
;ANSWER
google.com. 21599 IN NS ❶ns1.google.com.
google.com. 21599 IN NS ns2.google.com.
google.com. 21599 IN NS ns3.google.com.
google.com. 21599 IN NS ns4.google.com.
-- snip --
```

Listing 2-3: DNS answer of the google.com *NS resource records*

Like the CNAME record, discussed next, the NS record will return a fully qualified domain name ❶, not an IP address.

The Canonical Name Record

The *Canonical Name (CNAME) record* points one domain at another. Listing 2-4 shows a CNAME record response. CNAME records can make administration a bit easier. For example, you can create a CNAME record named *mail.yourdomain.com* and direct it to Gmail's login page. This not only is easier for your users to remember but also gives you the flexibility of pointing the CNAME at another email provider in the future without having to inform your users.

```
$ dig mail.google.com. a
-- snip --
;QUESTION
mail.google.com. IN A
;ANSWER
❶ mail.google.com. 21599 IN CNAME ❷googlemail.l.google.com.
googlemail.l.google.com. 299 IN A 172.217.3.229
-- snip --
```

Listing 2-4: DNS answer of the mail.google.com *CNAME resource record*

Notice that you ask the domain name server for the A record of the subdomain *mail.google.com*. But in this case, you receive a CNAME instead. This tells you that *googlemail.l.google.com* ❷ is the canonical name for *mail.google.com* ❶. Thankfully, you receive the A record for *googlemail.l.google.com* with the response, alleviating you from having to make a second query. You now know your destination IP address is 172.217.3.229. Google's domain name server was able to return both the CNAME answer and the corresponding Address answer in the same reply because it is an authority for the CNAME answer's domain name as well. Otherwise, you would expect only the CNAME answer and would then need to make a second query to resolve the CNAME answer's IP address.

The Mail Exchange Record

The *Mail Exchange (MX) record* specifies the mail server hostnames that should be contacted when sending email to recipients at the domain. Remote mail servers will query the MX records for the domain portion of a recipient's email address to determine which servers should receive mail for the recipient. Listing 2-5 shows the response a mail server will receive.

```
$ dig google.com. mx
-- snip --
;QUESTION
google.com. IN MX
;ANSWER
google.com. 599 IN MX ❶10 aspmx.l.google.com.
google.com. 599 IN MX 50 alt4.aspmx.l.google.com.
google.com. 599 IN MX 30 alt2.aspmx.l.google.com.
google.com. 599 IN MX 20 alt1.aspmx.l.google.com.
google.com. 599 IN MX 40 alt3.aspmx.l.google.com.
-- snip --
```

Listing 2-5: DNS answer of the google.com *MX resource records*

In addition to the domain name, TLL value, and record type, MX records contain the *priority field* ❶, which rates the priority of each mail server. The lower the number, the higher the priority of the mail server. Mail servers attempt to deliver emails to the mail server with the highest priority, then resort to the mail servers with the next highest priority if necessary. If more than one mail server shares the same priority, the mail server will pick one at random.

The Pointer Record

The *Pointer (PTR) record* allows you to perform a reverse lookup by accepting an IP address and returning its corresponding domain name. Listing 2-6 shows the reverse lookup for 8.8.4.4.

```
$ dig 4.4.8.8.in-addr.arpa. ptr
-- snip --
;QUESTION
❶ 4.4.8.8.in-addr.arpa. IN PTR
;ANSWER
4.4.8.8.in-addr.arpa. 21599 IN PTR ❷google-public-dns-b.google.com.
-- snip --
```

Listing 2-6: DNS answer of the 8.8.4.4 PTR resource record

To perform the query, you ask the domain name server for the IPv4 address in reverse order ❶ with the special domain *in-addr.arpa* appended because the reverse DNS records are all under the *.arpa* top-level domain. For example, querying the pointer record for the IP 1.2.3.4 means you need to ask for *4.3.2.1.in-addr.arpa*. The query in Listing 2-6 tells you that the IPv4 address 8.8.4.4 reverses to the domain name *google-public-dns-b.google.com* ❷. If you were performing a reverse lookup of an IPv6 address, you'd append the special domain *ip6.arpa* to the reversed IPv6 address as you did for the IPv4 address.

NOTE *See Wikipedia for more information on reverse DNS lookup:* https://en.wikipedia.org/wiki/Reverse_DNS_lookup.

The Text Record

The *Text (TXT) record* allows the domain owner to return arbitrary text. These records can contain values that prove domain ownership, values that remote mail servers can use to authorize email, and entries to specify which IP addresses may send mail on behalf of the domain, among other uses. Listing 2-7 shows the text records associated with *google.com*.

```
$ dig google.com. txt
-- snip --
;QUESTION
google.com. IN TXT
;ANSWER
```

```
google.com. 299 IN TXT
  ❶"facebook-domain-verification=22rm551cu4k0ab0bxsw536tlds4h95"
google.com. 299 IN TXT "docusign=05958488-4752-4ef2-95eb-aa7ba8a3bd0e"
google.com. 299 IN TXT "v=spf1 include:_spf.google.com ~all"
google.com. 299 IN TXT
    "globalsign-smime-dv=CDYX+XFHUw2wml6/Gb8+59BsH31KzUr6c1l2BPvqKX8="
-- snip --
```

Listing 2-7: DNS answer of the google.com TXT resource records

The domain queries and answers should start to look familiar by now. The last field in a TXT record is a string of the TXT record value ❶. In this example, the field has a Facebook verification key, which proves to Facebook that Google's corporate Facebook account is who they say they are and has the authority to make changes to Google's content on Facebook. It also contains *Sender Policy Framework* rules, which inform remote mail servers which IP addresses may deliver email on Google's behalf.

NOTE *The Facebook for Developers site has more information about domain verification at* https://developers.facebook.com/docs/sharing/domain-verification/.

Multicast DNS

Multicast DNS (mDNS) is a protocol that facilitates name resolution over a local area network (LAN) in the absence of a DNS server. When a node wants to resolve a domain name to an IP address, it will send a request to an IP multicast group. Nodes listening to the group receive the query, and the node with the requested domain name responds to the IP multicast group with its IP address. You may have used mDNS the last time you searched for and configured a network printer on your computer.

Privacy and Security Considerations of DNS Queries

DNS traffic is typically unencrypted when it traverses the internet. A potential exception occurs if you're connected to a virtual private network (VPN) and are careful to make sure all DNS traffic passes through its encrypted tunnel. Because of DNS's unencrypted transport, unscrupulous ISPs or intermediate providers may glean sensitive information in your DNS queries and share those details with third parties. You can make a point of visiting HTTPS-only websites, but your DNS queries may betray your otherwise secure browsing habits and allow the DNS server's administrators to glean the sites you visit.

Security is also a concern with plaintext DNS traffic. An attacker could convince your web browser to visit a malicious website by inserting a response to your DNS query. Considering the difficulty of pulling off such an attack, it's not an attack you're likely to experience, but it's concerning nonetheless. Since DNS servers often cache responses, this attack usually takes place between your device and the DNS server it's configured to use. RFC 7626 (*https://tools.ietf.org/html/rfc7626/*) covers these topics in more detail.

Domain Name System Security Extensions

Generally, you can ensure the authenticity of data sent over a network in two ways: authenticating the content and authenticating the channel. *Domain Name System Security Extensions (DNSSEC)* is a method to prevent the covert modification of DNS responses in transit by using digital signatures to authenticate the response. DNSSEC ensures the authenticity of data by authenticating the content. DNS servers cryptographically sign the resource records they serve and make those signatures available to you. You can then validate the responses from authoritative DNS servers against the signatures to make sure the responses aren't fraudulent.

DNSSEC doesn't address privacy concerns. DNSSEC queries still traverse the network unencrypted, allowing for passive observation.

DNS over TLS

DNS over TLS (DoT), detailed in RFC 7858 (*https://tools.ietf.org/html/rfc7858/*), addresses both security and privacy concerns by using *Transport Layer Security (TLS)* to establish an encrypted connection between the client and its DNS server. TLS is a common protocol used to provide cryptographically secure communication between nodes on a network. Using TLS, DNS requests and responses are fully encrypted in transit, making it impossible for an attacker to eavesdrop on or manipulate responses. DoT ensures the authenticity of data by authenticating the channel. It does not need to rely on cryptographic signatures like DNSSEC because the entire conversation between the DNS server and the client is encrypted.

DoT uses a different network port than does regular DNS traffic.

DNS over HTTPS

DNS over HTTPS (DoH), detailed in RFC 8484 (*https://tools.ietf.org/html/rfc8484/*) aims to address DNS security and privacy concerns while using a heavily used TCP port. Like DoT, DoH sends data over an encrypted connection, authenticating the channel. DoH uses a common port and maps DNS requests and responses to HTTP requests and responses. Queries over HTTP can take advantage of all HTTP features, such as caching, compression, proxying, and redirection.

What You've Learned

We covered a lot of ground in this chapter. You learned about IP addressing, starting with the basics of IPv4 multicasting, broadcasting, TCP and UDP ports, socket addresses, network address translation, and ARP. You then learned about IPv6, its address categories, and its advantages over IPv4.

You learned about the major network-routing protocols, ICMP and DNS. I'll again recommend the *TCP/IP Guide* by Charles M. Kozierok (No Starch Press, 2005) for its extensive coverage of the topics in this chapter.