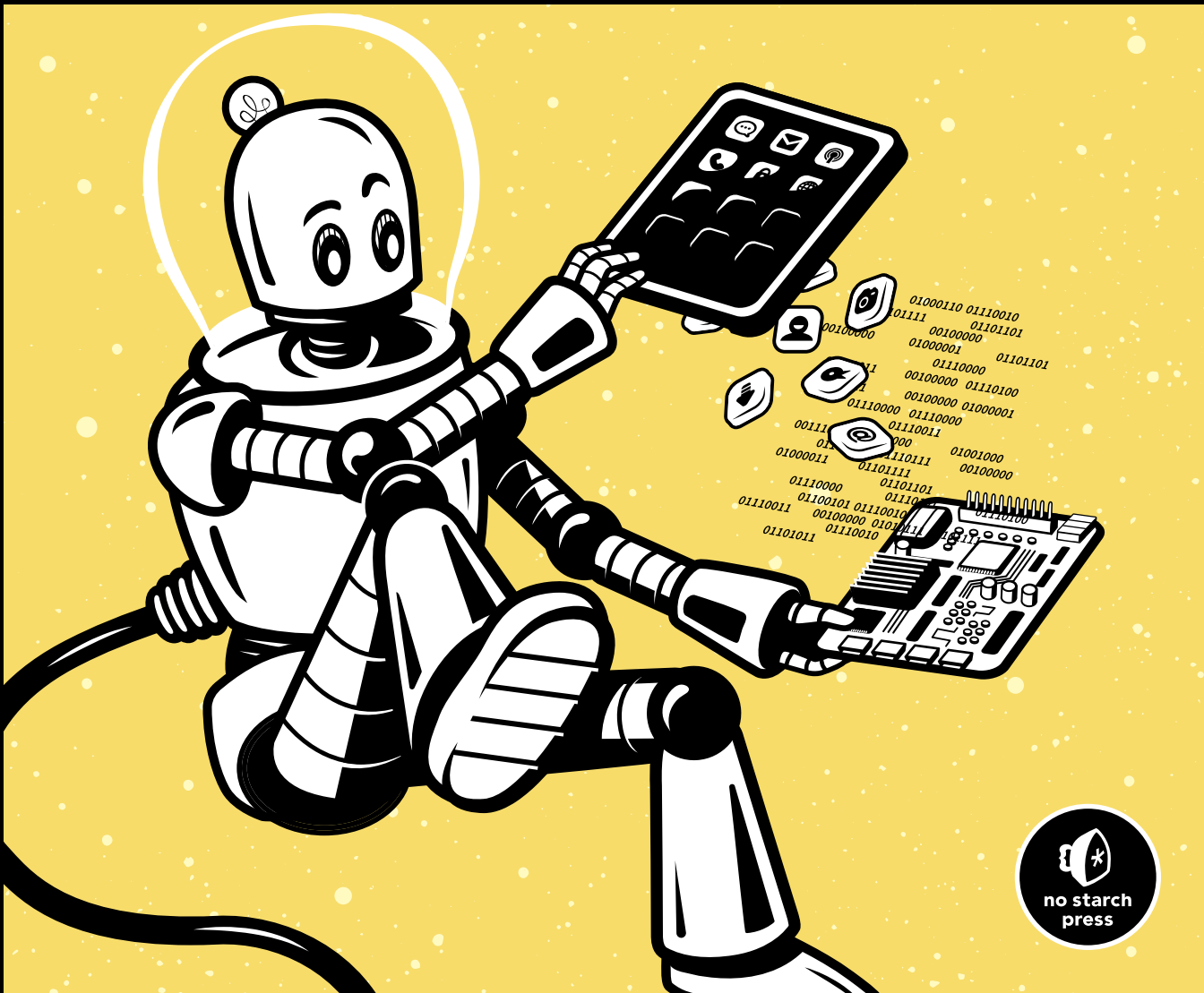
# HOW COMPUTERS REALLY WORK

## A HANDS-ON GUIDE TO THE INNER WORKINGS OF THE MACHINE

MATTHEW JUSTICE



no starch press

# HOW COMPUTERS REALLY WORK

# HOW COMPUTERS REALLY WORK

## A Hands-On Guide to the Inner Workings of the Machine

by Matthew Justice

no starch press

San Francisco

ISBN-13: 978-1-71850-066-2 (print)
ISBN-13: 978-1-71850-067-9 (ebook)

Publisher: William Pollock
Execuitve Editor: Barbara Yien
Production Editor: Katrina Taylor
Developmental Editor: Alex Freed
Project Editor: Dapinder Dosanjh
Cover Design: Gina Redman
Interior Design: Octopod Studios
Technical Reviewers: William Young, John Hewes, and Bryan Wilhem
Copyeditor: Happenstance Type-O-Rama
Compositor: Happenstance Type-O-Rama
Proofreader: Happenstance Type-O-Rama

The following images are reproduced with permission:

Figures 7-7, 7-9, 7-10, and 7-11 ALU symbol was altered from the image created by Eadthem and is licensed under the Creative Commons Attribution-Share Alike 3.0 Unported license (*https://commons.wikimedia.org/wiki/File:ALU_symbol-2.svg*). Figures 11-3, 11-5, 11-14, 11-15, 11-16, 11-17, 12-4, 12-8, 12-9, 12-10, 13-5, 13-9 server icon is courtesy of Vecteezy.com.

For information on distribution, translations, or bulk sales, please contact No Starch Press, Inc. directly:
No Starch Press, Inc.
245 8th Street, San Francisco, CA 94103
phone: 1.415.863.9900; info@nostarch.com
www.nostarch.com

*Library of Congress Cataloging-in-Publication Data:*

```
Names: Justice, Matthew, author.
Title: How Computers Really Work : a hands-on guide to the inner workings of the machine / Matthew Justice.
Description: San Francisco : No Starch Press, Inc., [2020] | Includes
   index.
Identifiers: LCCN 2020024168 (print) | LCCN 2020024169 (ebook) | ISBN
   9781718500662 (paperback) | ISBN 1718500661 (paperback) | ISBN
   9781718500679 (ebook)
Subjects: LCSH: Electronic digital computers--Popular works.
Classification: LCC QA76.5 .J87 2020  (print) | LCC QA76.5  (ebook) | DDC
   004--dc23
LC record available at https://lccn.loc.gov/2020024168
LC ebook record available at https://lccn.loc.gov/2020024169
```

## About the Author

Matthew Justice is a software engineer. He spent 17 years at Microsoft where he took on various roles, including debugging the Windows kernel, developing automated fixes, and leading a team of engineers responsible for building diagnostic tools and services. He has worked on low-level software (the operating system) and on software far removed from the underlying hardware (such as web applications). Matthew has a degree in electrical engineering. When he's not writing code or building circuits, Matthew enjoys spending time with his family, hiking, reading, arranging music, and playing old video games.

## About the Tech Reviewers

Dr. Bill Young is Associate Professor of Instruction in the Deptartment of Computer Science at the University of Texas at Austin. Prior to joining the UT faculty in 2001, he had 20 years of experience in the industry. He specializes in formal methods and computer security, but often teaches computer architecture, among other courses.

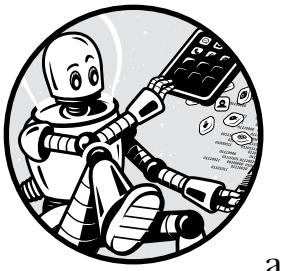
Bryan Wilhelm is a software engineer. He has degrees in mathematics and computer science and has been working at Microsoft for 20 years in roles ranging from debugging the Windows kernel to developing business applications. He enjoys reading, science-fiction movies, and classical music.

John Hewes began connecting electrical circuits at an early age, moving on to electronics projects as a teenager. He later earned a physics degree and continued to develop his interest in electronics, helping school students with their projects while working as a science technician. John has taught electronics and physics up to an advanced level in the United Kingdom and ran a school electronics club for children aged 11 to 18 years, setting up the website *http://www.electronicsclub.info/* to support the club. He believes that everyone can enjoy building electronics projects, regardless of their age or ability.

# BRIEF CONTENTS

# 11

# THE INTERNET



So far, we've focused on computing that occurs on a single device. In this chapter and the next, we look at computing that spans multiple devices. We're going to examine two significant innovations in computing, the internet and the world wide web, which are not the same thing! This chapter focuses on the internet, and we begin by defining key terms. Then we look at a layered model of networks and dig into some of the foundational protocols used on the internet.

## Networking Terms Defined

To discuss the internet and networks in general, you first need to become familiar with some concepts and terms, which we cover here. A *computer*

*network* is a system that allows computing devices to communicate with each other, as illustrated in Figure 11-1. Networks can be connected wirelessly, using technologies like Wi-Fi, which transmits data using radio waves. Networks can also be connected with cables, such as copper wiring or fiber optics. Computing devices on a network must use a common *communications protocol*, a set of rules that describe how information is to be exchanged.

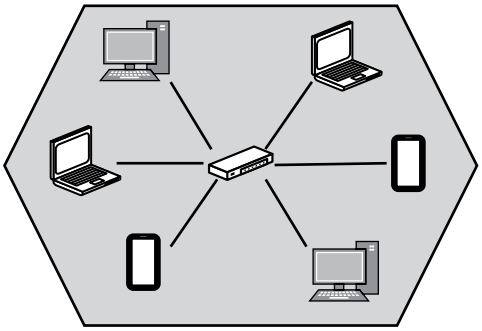


*Figure 11-1: A computer network*

The *internet* is a globally connected set of computer networks that all use a suite of common protocols. The internet is a *network of networks*, connecting networks from various organizations all around the world, as shown in Figure 11-2.

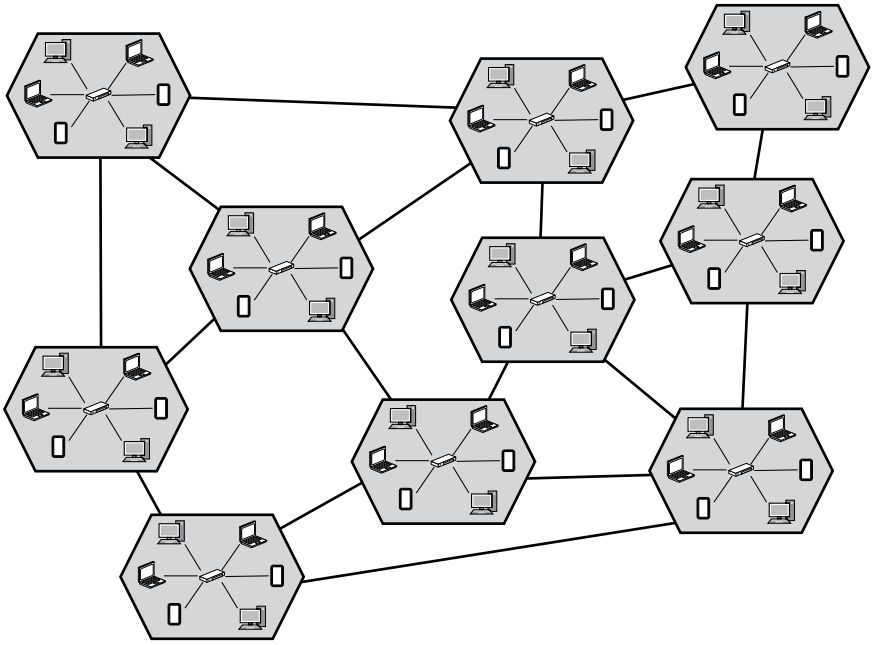


*Figure 11-2: The internet: a network of networks*

A *host* or *node* is a single computing device attached to a network. A host can act as a server or a client on the network, or sometimes both. A network *server* is a host that listens for inbound network connections and provides services to other hosts. Examples are a web server and an email server. A network *client* is a host that makes outbound connections and requests services from network servers. Example clients are smartphones or laptops running web browsers or email apps. A client makes a *request* to a server, and the server replies with a *response*, as illustrated in Figure 11-3.

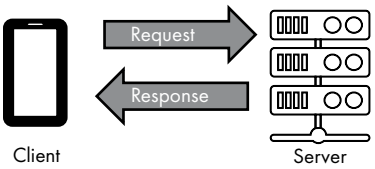



*Figure 11-3: A client makes a request to a server, and the server responds*

The term *server*, as just used, refers to any device that accepts inbound requests and provides services to clients. However, *server* can also refer to a class of computer hardware that's specifically intended to act as a network server. These specialized computers are physically designed to be mounted into racks of computers in a datacenter and often include hardware redundancy and management capabilities not found in a typical PC. However, any device with the right software can act as a server on a network.

## The Internet Protocol Suite

Physically connecting the networks of the world isn't enough to allow the devices on those networks to communicate with each other. All participating computers need to communicate in the same way. The *internet protocol suite* standardizes the method of communication on the internet, ensuring that all devices on the network speak the same language. The two foundational protocols in the internet protocol suite are *Transmission Control Protocol (TCP)* and *Internet Protocol (IP)*, collectively known as *TCP/IP*.

Network protocols operate in a layered model, and an implementation of such a model is referred to as a *network stack* (not to be confused with a stack in memory, as covered in Chapter 9). The protocols at the lowest layer interact with the underlying networking hardware, whereas applications interact with protocols in the upper layers. Protocols in the intermediate layers provide services such as addressing and reliable delivery of data. A protocol at a certain layer doesn't have to concern itself with the entire networking stack, only the layers with which it interfaces, simplifying the overall design. This is another example of encapsulation.

The internet protocol suite is designed around a four-layer model. This is sometimes called the *TCP/IP model*. The four layers of the TCP/IP model, starting from the bottom up, are the link layer, the internet layer, the transport layer, and the application layer, as shown in Figure 11-4.

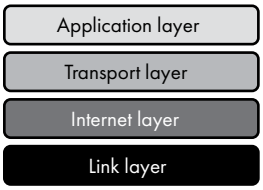



*Figure 11-4: The internet protocol suite model of networking*

**OSI—ANOTHER NETWORK MODEL**

Another commonly used model for network protocols is the *Open Systems Interconnection (OSI) model*. The OSI model divides protocols into seven layers rather than four. This model is often referenced in technical literature, but the internet is based on the internet protocol suite, so this book focuses on the TCP/IP model.

These networking layers represent an abstraction, a model for us to use when discussing the operation of the internet. In practice, each layer is realized with specific networking protocols. Each network layer represents a scope of responsibilities, and protocols must fulfill the responsibilities of their assigned layer. Table 11-1 provides a description of each layer.

**Table 11-1:** Description of the Four Layers of the Internet Protocol Suite

| Layer | Description | Example protocols |
|---|---|---|
| Application | Protocols that operate at the application layer provide application-specific functionality, such as sending an email or retrieving a web page. These protocols accomplish tasks that end users (or backend services) wish to complete. Application layer protocols structure the data used in process-to-process communication across a network. All the lower layer protocols exist as "plumbing" to support the application layer. | HTTP, SSH |

| Layer | Description | Example protocols |
|---|---|---|
| Transport | Transport layer protocols provide a communications channel for applications to send and receive data between hosts. An application structures data according to an application layer protocol and then hands off that data to a transport layer protocol for delivery to a remote host. | TCP, UDP |
| Internet | Internet layer protocols provide a mechanism for communicating across networks. This layer is responsible for identifying hosts with addresses and enabling the routing of data from network to network across the internet. The transport layer relies on the internet layer for addressing and routing. | IP |
| Link | Link layer protocols provide a way to communicate on a local network. Protocols at this layer are closely associated with the type of networking hardware on a local network, such as Wi-Fi. Protocols at the internet layer rely on link layer protocols to communicate on a local network. | Wi-Fi, Ethernet |

Protocols at each layer communicate with the protocols in adjacent layers. An outgoing transmission from a host travels down through the network layers, from an application layer protocol, to a transport layer protocol, to an internet layer protocol, and finally to a link layer protocol. An incoming transmission to a host travels up through the network layers, reversing the order just described.

Although network hosts (such as a client or server) make use of protocols from all four layers, other types of networking hardware (such as switches and routers) only use protocols associated with lower layers. Such devices can perform their jobs without bothering to examine the higher layer protocol data contained in a network transmission.

An outgoing request from a client to a server, and its relationship to the networking layers, is illustrated in Figure 11-5.

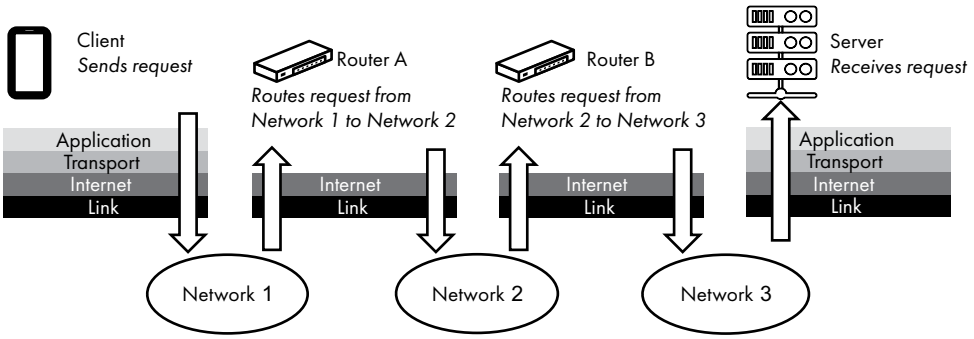



*Figure 11-5: A network request travels through various network layers*

Let's walk through the flow of Figure 11-5. An application on the client device forms a request using an application layer protocol. That request is handed off to a transport layer protocol, then to an internet layer protocol, and finally to a link layer protocol. All of this happens on the client device. At this point the request is transmitted onto the local network, labeled Network 1 in the diagram. The request makes its way across the internet, going from network to network. In this example, Router A routes the request from Network 1 to Network 2, and Router B routes the request from Network 2 to Network 3. Once the request reaches the destination server, it works its way up through the networking protocols, starting with a link layer protocol, and ending at an application layer protocol. A process running on the server receives the request, which is formatted according to the application layer protocol originally used by the client. The server process interprets the request and responds in an appropriate manner.

Let's now take a look at each layer, starting from the bottom.

### Link Layer

The lowest level of the internet protocol suite is the *link layer*. The physical and logical connections between hosts are known as network *links*. Link layer protocols are used by devices on the same network to communicate with each other. Each device on a link has a network address that uniquely identifies it. For many link layer protocols, this address is known as a *media access control address* (or *MAC address*). Link layer data is divided into small units known as *frames*, each including a header describing the frame, a payload of data, and finally, a frame footer used to detect errors. This is illustrated in Figure 11-6.


*Figure 11-6: A link layer frame*

The frame header contains source and destination MAC addresses. The header also includes a descriptor of the type of data carried in the frame data section.

If your home has a Wi-Fi network, Wi-Fi is the link between the hosts on your network. The Wi-Fi protocol, defined by the IEEE 802.11 specifications, doesn't know or care what type of data is being sent over the wireless network; it simply enables communication between devices. Each device connected to the Wi-Fi network has a MAC address and receives frames sent to its address. MAC addresses are only useable on a local network; a computer on a remote network cannot directly send data to a MAC address on your local network.

Another notable link layer technology is *Ethernet*, used for wired physical connections. Ethernet is defined by the IEEE 802.3 standards. Ethernet typically uses a cable with pairs of copper wires inside that ends in a connector commonly known as *RJ45*, shown in Figure 11-7.

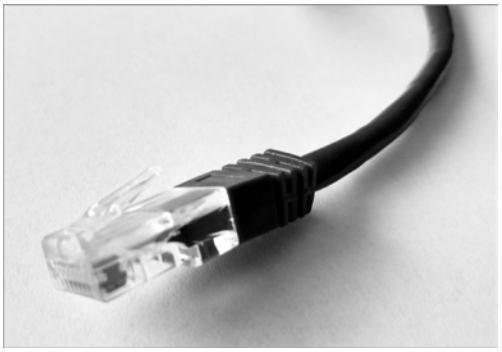


*Figure 11-7: The cable commonly used for Ethernet*

All devices connected to the internet participate in the link layer. This is required, since it's the link layer that provides connectivity (either wired or wireless) to a local network. A host, like a laptop or smartphone, participates in all layers, but certain networking devices operate at the link layer only. The most basic example of this is a hub. A *network hub* is a networking device that connects multiple devices on a local network without any intelligence regarding the frames being sent. A simple hub might provide multiple Ethernet ports for connecting devices. The hub simply retransmits every frame it receives on one physical port to all its other ports. A more intelligent link layer device is a *network switch*, which examines the MAC addresses in the frames it receives and sends those frames to the physical port where the device with the destination MAC address is connected.

**NOTE** *Please see Project #29 on page 254, where you can look at link layer devices and MAC addresses.*

### Internet Layer

The *internet layer* allows data to travel beyond the local network. The primary protocol used in this layer is simply called Internet Protocol (IP). It enables *routing*, the process of determining a path for data that's transmitted between networks. Every host on the internet is assigned an *IP address*, a number that uniquely identifies the host on the global internet. It's also possible to have private IP addresses that aren't directly exposed on the internet. IP addresses are usually assigned by a server on the local network, and a device's IP address typically changes when it connects to a new network. We'll cover more on address assignment and private IP addresses later.

Data sent over the internet layer is called a *packet*, which is enclosed in a link layer frame. Figure 11-8 illustrates the idea that a packet fits within a frame's data section.

The IP packet header contains a source IP address and a destination IP address. The header also includes information that describes the packet, such as the IP version in use and the header length. The data section of the IP packet contains the payload that the IP layer is carrying.

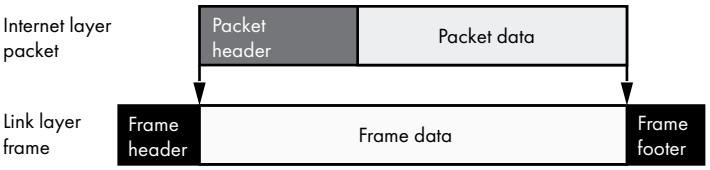



*Figure 11-8: A packet is contained in the data section of a frame*

Two versions of Internet Protocol are in use on the internet today. *Internet Protocol Version 4 (IPv4)* is the dominant version in use, and the other active version is *Internet Protocol Version 6 (IPv6)*. You may wonder what happened to IPv5. No such protocol ever existed, but an experimental protocol called Internet Stream Protocol identified its IP version as 5, and so IPv5 was skipped when the successor to IPv4 was developed. A significant difference between IPv4 and IPv6 is the size of an IP address. An IPv4 address is 32 bits in length, whereas an IPv6 address is 128 bits. This difference allows for a vastly larger number of addresses with IPv6. This change in address size is meant to help deal with the relatively short supply of IPv4 addresses. In this book, we focus on IPv4 addresses (and just refer to them as *IP addresses*), as they are still the primary means of addressing on the internet today.

A 32-bit IP address is typically displayed in dotted decimal notation, meaning the 32 bits are separated into four groups of 8 bits each, the 8-bit numbers are displayed in decimal (rather than hexadecimal or binary), and the four decimal numbers are separated by periods (dots). An example IP address, displayed in dotted decimal notation, is 192.168.1.23. Each 8-bit decimal number can be referred to as an *octet*.

Computers connected to the same local network have IP addresses that begin with the same leading bits and are said to be on the same *subnet*. Computers that are on the same subnet are able to communicate directly with each other at the link layer because they are operating on the same physical network. Computers that are on different subnets must send their traffic through a *router*, a device that connects subnets and operates at the internet layer.

Subnetting divides the IP address into two parts: the *network prefix*, which all devices on the same subnet share, and the *host identifier*, which is unique to a host on that subnet. The number of bits included in the network prefix varies based on the network configuration.

Let's look at an example. Assume a subnet uses a 24-bit network prefix, leaving us with 8 bits to represent the host. Also assume that a host on this subnet uses the example IP address from earlier—192.168.1.23. Given this IP address and network prefix, the IP address is divided as shown in Figure 11-9.

In this example, all hosts on the local subnet have an IP address that begins with 192.168.1. Each host has a different value for the last octet, with 23 being assigned to this specific host. This example uses a 24-bit prefix length, meaning the prefix neatly aligns with the first three octets of the IP address.

This makes for a nice example, but the prefix length doesn't always align with an octet boundary. A 25-bit prefix, for example, would also include the first bit of the last octet, leaving only 7 bits for identifying the host.

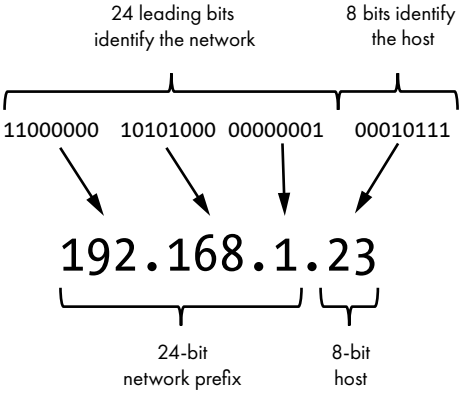



*Figure 11-9: An example IP address using a 24-bit network prefix*

The number of bits reserved for the network prefix is commonly expressed in one of two ways. *Classless Inter-Domain Routing (CIDR) notation* lists an IP address followed by a slash (/), and then the number of bits used for the network prefix. In our example this would be 192.168.1.23/24. Another common way to represent the number of prefix bits is with a *subnet mask*, a 32-bit number where a binary 1 is used for each bit that's part of the network prefix and a 0 is used for each bit that's part of the host number. Subnet masks are also written in dotted decimal notation, so our example of a 24-bit network prefix would result in a subnet mask of 255.255.255.0, as shown in Figure 11-10.

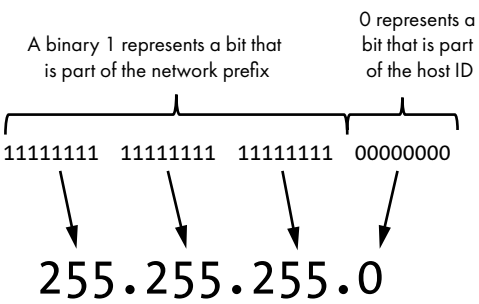



*Figure 11-10: A 24-bit network prefix expressed as a subnet mask*

Let's look at how this is useful in practice. Say your computer has an IP address of 192.168.0.133 and a subnet mask of 255.255.255.224, or, expressed in CIDR notation, 192.168.0.133/27. Your computer wishes to connect to another computer with an IP address of 192.168.0.84. As mentioned earlier, two computers can communicate directly if they are on the same subnet, and if not, they must go through a router. So your computer must determine if the other computer is on the same subnet. How can it do this?

Performing a bitwise logical AND of an IP address and its subnet mask produces the first address in a subnet. This first address, where the host bits are all 0, serves as an identifier for the subnet itself. This is commonly referred to as the *network ID*. Two computers that share a network ID are on the same subnet. A host can perform this AND operation against both its own IP address and the IP address it wishes to connect to, to see if they share a network ID and thus are on the same subnet. Let's try this with our example computer's IP address, as shown here:

```
  IP = 192.168.0.133   = 11000000.10101000.00000000.10000101
MASK = 255.255.255.224 = 11111111.11111111.11111111.11100000
 AND = 192.168.0.128   = 11000000.10101000.00000000.10000000 = The network ID
```

Now perform the same operation for the second computer in our example:

```
  IP = 192.168.0.84    = 11000000.10101000.00000000.01010100
MASK = 255.255.255.224 = 11111111.11111111.11111111.11100000
 AND = 192.168.0.64    = 11000000.10101000.00000000.01000000 = The network ID
```

As you can see from this example, this operation produced two different network IDs (`192.168.0.128` and `192.168.0.64`). This means that the second computer is not on the same subnet as your computer. To communicate, these computers need to send their messages through a router connecting the two subnets.

**EXERCISE 11-1: WHICH IPS ARE ON THE SAME SUBNET?**

Is IP address 192.168.0.200 on the same subnet as your computer? Assume your computer has an IP address of 192.168.0.133 and a subnet mask of 255.255.255.224.

Here's another way to look at this: the network prefix describes the range of addresses that can be used on a subnet. The first address in that range is defined as the network prefix bits followed by all binary 0s for the host identifier. Continuing with our example computer at 192.168.0.133, the first address on its subnet is 192.168.0.128. The last address in the range is the network prefix bits followed by all binary 1s for the host identifier. In our example that's 192.168.0.159. The first and last addresses have special meanings—the first identifies the network, the last is the *broadcast address* (used for sending a message to all hosts on the subnet). All the addresses in between can be used for hosts on the subnet. Our example IP address of 192.168.0.133 is clearly in this range (from 192.168.0.128 to 192.168.0.159), while the other computer with an IP address of 192.168.0.84 is outside this range.

You can also use the number of bits reserved for the host identifier to determine how many IP addresses are available for hosts on a subnet. In

our example, 27 bits are reserved for the network prefix, leaving 5 bits for host identifiers. These 5 bits give us 32 possible host addresses, since $2^5$ is 32. However, as mentioned earlier, the first and last addresses have special purposes, so really only 30 hosts can be identified using this network prefix. This aligns with our earlier findings: the first host identifier is 128, and 128 + 32 gives us 160, the first address in the next subnet, so 159 would be the last host in our range.

**NOTE** *Please see Project #30 on page 255, where you can look at the internet layer using your Raspberry Pi.*

### Transport Layer

The *transport layer* provides a communications channel that applications may use to send and receive data. There are two commonly used transport layer protocols: Transmission Control Protocol (TCP) and *User Datagram Protocol (UDP)*. TCP provides a reliable connection between two hosts. It ensures that errors are minimized, data arrives in order, lost data is resent, and so forth. Data sent with TCP is known as a *segment*. On the other hand, UDP is a "best effort" protocol, meaning its delivery is unreliable. UDP is preferred when speed is valued over reliability. Data sent with UDP is known as a *datagram*. Both protocols have their place, but to keep things simple, I cover only TCP for the remainder of the chapter. Figure 11-11 illustrates the idea that a TCP segment fits within a packet's data section, which in turn fits within a frame's data section.

As we saw earlier, the link layer includes a destination MAC address in the frame header to identify a local network interface, and the internet layer includes a destination IP address in the packet header to identify the host on the internet. That's enough information to get a packet to a specific device on the internet. Once a packet has reached its destination host, the transport layer header includes a destination network *port number* that identifies the specific service or process that will receive the data. A host with a single IP address can have multiple active ports, each used for performing a different type of activity on the network.

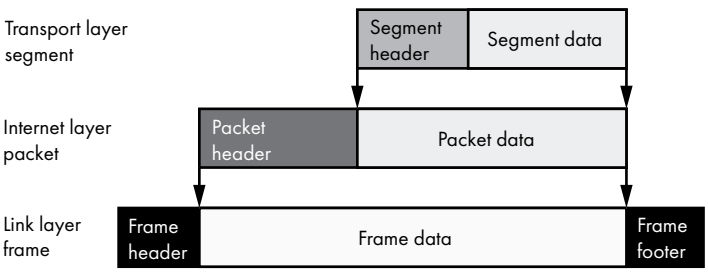



*Figure 11-11: A TCP segment is contained in the data section of an IP packet*

To use an analogy, an IP address is like the street address of an office building, and a network port number is like the office number of a worker

in that office building. The IP address uniquely identifies a host computer, just as a street address uniquely identifies an office building. Using internet protocol, a packet can be delivered to a host in the same way that a package can be delivered to an office building. However, once a packet arrives at the computer, the operating system must decide what to do with it. The packet isn't intended for the OS itself, but for some process running on the computer. In the same way, a package arriving at an office building likely isn't intended for the mailroom worker but for someone else in the building. An operating system examines the port number and delivers the inbound data to the process listening on the specified port, just as a mailroom worker examines the name or office number on the package to deliver the package to the right person.

Network ports in the range of 0 to 1,023 are called *well-known ports*, whereas ports in the range of 1,024 to 49,151 can be registered with the Internet Assigned Numbers Authority (IANA) and are known as *registered ports*. Ports with a value greater than 49,151 are *dynamic ports*. Technically, any process with sufficient privileges can listen on any port that isn't already in use on a system, potentially ignoring the typical use case for that port number. However, when a client application wishes to connect to a remote service on another computer, it needs to know what port to use, so it makes sense to standardize port numbers. For example, web servers typically listen on port 80 and port 443 (for encrypted connections). A web browser assumes that it should use port 80 or 443 unless directed otherwise.

**EXERCISE 11-2: RESEARCH COMMON PORTS**

Find the port numbers for common application layer protocols. What are the port numbers for Domain Name System (DNS), Secure Shell (SSH), and Simple Mail Transfer Protocol (SMTP)? You can find this information online with a search, or by looking at the IANA registry, here: *http://www.iana.org/assignments/port-numbers*. The IANA listings sometimes use an unexpected term for the service name. For example, DNS is simply listed as "domain."

Servers use well-known ports to make it easy for clients to connect. However, most network communication is a two-way street (a client sends a request and a server responds), so the client needs to have an open port as well so that it can receive data from the server. A client only needs to temporarily open such a port, just long enough for it to complete its communication with a server. Such ports are called *ephemeral ports* and are assigned by the networking components in the operating system. For example, a client web browser connects to a web server on port 80, and an ephemeral port on the client is also opened, let's say port number 61,348. The client sends its web request to port 80 on the server, and the server sends its response to port 61,348 on the client.

An IP address plus a port number form an *endpoint*, and an instance of an endpoint is known as a *socket*. A socket can listen for new connections, or it can represent an established connection. If multiple clients connect to the same endpoint, each has its own socket.

**NOTE** *Please see Project #31 on page 256, where you can look at the port usage of your Raspberry Pi.*

### Application Layer

The *application layer* is the final, topmost layer of the internet protocol suite. While the lower three layers provide a generalized foundation for communication over the internet, the protocols at the application layer focus on accomplishing a specific task. For example, web servers use *HyperText Transfer Protocol (HTTP)* for retrieving and updating web content. Email servers use *Simple Mail Transfer Protocol (SMTP)* for sending and receiving email messages. File transfer servers use *File Transfer Protocol (FTP)* to, you guessed it, transfer files! In other words, the application layer is where we get to the protocols that describe the behavior of applications, whereas the lower layers of the stack are the "plumbing" that enables applications to do the things they want to do over the internet. Figure 11-12 provides a completed view of the four layers.

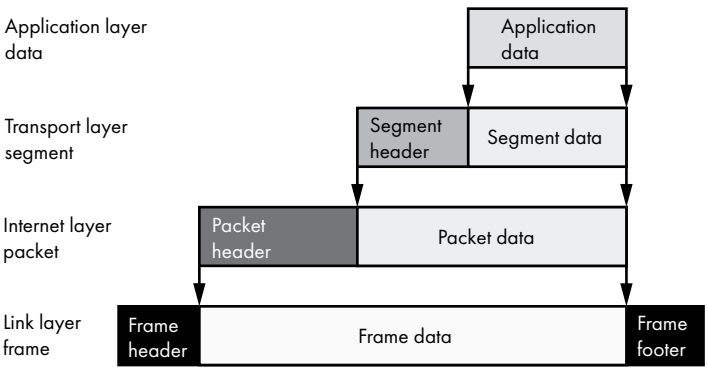



*Figure 11-12: The application layer's data is contained in the segment's data section.*

Figure 11-12 is a breakout view of how each layer fits in the lower layer's data payload. Assembling all the layers together in Figure 11-13, we can see a representation of what is contained in a frame sent to a device on the internet.

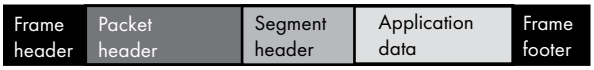



*Figure 11-13: A frame containing an IP packet, a TCP segment, and application data*

We've walked through the contents of a network frame from the bottom up, starting with the layer closest to hardware. When a frame is received by a host, it is processed by the host in that same order, from the link layer up to the application layer. In contrast, when a frame is sent from a host, the frame is assembled in the reverse order. A process prepares application data that is enclosed in a segment, a packet, and finally, a frame.

## A Trip Through the Internet

Now that you're familiar with each of the four layers in the TCP/IP networking model, let's look at an example of how data travels across the internet. We'll see how various devices along the way interact with each of the networking layers. Figure 11-14 illustrates this, showing a client in the upper left communicating with a server in the lower left.

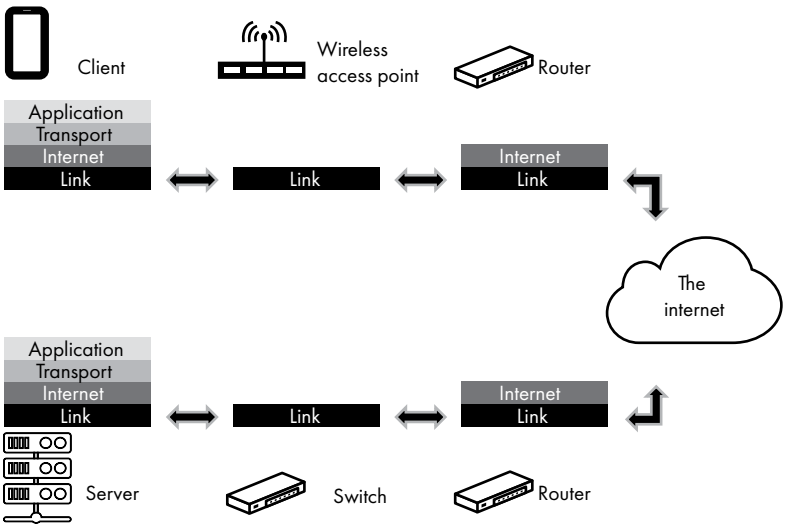



*Figure 11-14: Different devices interact at different layers of the networking stack*

I'll set up the scenario in Figure 11-14. A client device (upper left of diagram) is connected to a wireless Wi-Fi network. That network is connected to the internet via a router. Somewhere else we have a server (lower left of diagram), which has a wired connection to the internet through a switch and router. A user of the client device opens a web browser and requests a web page hosted on the server. For simplicity, let's assume that the client already knows the IP address of the server.

The web browser on the client "speaks" HTTP, the application layer protocol of the web, so it forms an HTTP request intended for the destination server. The browser then hands off the HTTP request to the operating system's TCP/IP software stack, asking that the data be delivered to the server—specifically the server's IP address and port 80, the standard port for HTTP. The TCP/IP software stack on the client operating system then encapsulates the HTTP payload in a TCP segment (transport layer), setting the destination

port to 80 in the segment header. If necessary, TCP divides the application layer data into multiple segments, each with its own header. The internet layer software on the client then wraps the TCP segment in an IP packet, which includes the destination IP address of the server in the packet header. If necessary, IP divides the packet into multiple smaller fragments in preparation for transmission over the network link. At the link layer on the client, the IP packet is encapsulated in a frame with the MAC address of the local router in its header. This frame is wirelessly transmitted by the client device's Wi-Fi hardware.

The wireless access point receives the frame. The access point, operating at the link layer, sends the frame along to the router. The router examines the internet layer packet to determine the destination IP address. To reach the server, the request needs to travel through multiple routers on the internet. The local router encapsulates the packet in a new frame, with a new destination MAC address (the address of the next router), and sends the new frame on its way. This routing process continues through multiple routers on the internet until the request reaches the router on the subnet where the server is connected.

The last router encapsulates the packet in a frame suitable for the server's local network. This frame's header includes the MAC address of the server. The switch on the server's subnet looks at the MAC address in the frame and forwards the frame out the appropriate physical port. There's no need for the switch to look at any higher layers. The server receives the frame, and the driver for the network interface passes the TCP/IP packet up to the TCP/IP software stack, which in turn, hands off the HTTP data to the process listening on TCP port 80. Web server software, listening on port 80, handles the request. This includes replying to the client, and to do that, this entire process happens again, except in reverse order.

**NOTE** *Please see Project #32 on page 258, where you can see the route from your Raspberry Pi to a host on the internet.*

## Foundational Internet Capabilities

Whereas TCP/IP provides the necessary plumbing for reliable transfer of data across the internet, other protocols provide additional foundational internet capabilities. These features are implemented as application layer protocols. Let's now look at two such protocols (DHCP and DNS) and a system for translating IP addresses (NAT).

### *Dynamic Host Configuration Protocol*

Every host on the internet needs an IP address, a subnet mask, and the IP address of its router (also called its *default gateway*) in order to communicate with other hosts. How are IP addresses assigned? A device can be given a *static IP address*, which requires someone to edit the configuration on the device and set its IP information manually. This is sometimes useful, but it requires the user to ensure that the IP address in question isn't already

taken and is valid for the subnet. Most end users don't have the expertise to manually configure the IP settings for their devices, nor do they want to deal with the hassle of manual configuration. Fortunately, most IP addresses are assigned dynamically using *Dynamic Host Configuration Protocol (DHCP)*. With DHCP, when a device connects to a network, it receives an IP address and related information without user intervention.

For DHCP to be available on a network, a device on the network must be configured as a *DHCP server*. This server has a pool of IP addresses that it's allowed to assign to devices on the network. The flow of DHCP is illustrated in Figure 11-15.

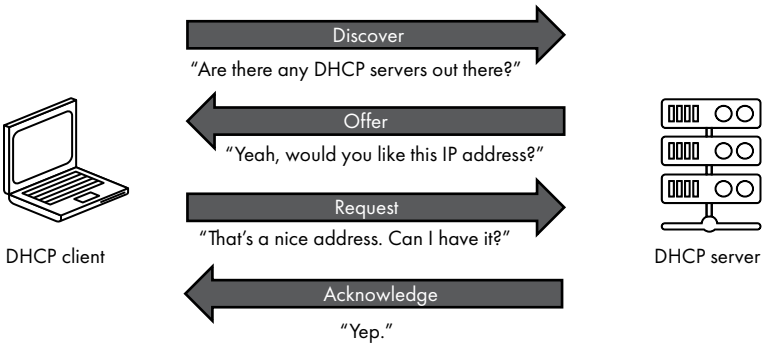



*Figure 11-15: A DHCP conversation*

Let's walk through Figure 11-15. When a device connects to a network, it broadcasts a message to discover any DHCP servers. A *broadcast* is a special kind of packet that's addressed to all hosts on the local network. When the DHCP server receives this broadcast, it offers an IP address to the client device. If the client wishes to accept the offered IP address, it replies to the server with a request for the offered address. The DHCP server then acknowledges the request, and the IP address is assigned to the client. The IP address is *leased* to the client, and it eventually expires if the client does not renew its lease.

**NOTE** *Please see Project #33 on page 258, where you can see the IP address your Raspberry Pi has leased using DHCP.*

### Private IP Addresses and Network Address Translation

The number of available IP addresses is limited, so most home internet service providers (ISPs) only assign a single IP address to a customer. This IP address is assigned to the device that's directly attached to the ISP's network, usually a router. However, many customers have multiple devices on their home network. Let's look at how multiple devices can share a single public IP address by leveraging private IP addresses and Network Address Translation.

Certain ranges of IP addresses are considered *private IP addresses*, addresses intended to be used on *private networks*, such as those in homes or offices, where the devices aren't directly connected to the internet. Any address that matches the pattern of 10.*x.x.x*, 172.16.*x.x*, or 192.168.*x.x* is a private IP address. Anyone can use these ranges of IP addresses without asking permission. The catch is that private IP addresses are nonroutable—they can't be used on the public internet. A DHCP server on a home network can assign these addresses without worrying about whether any other network is using the same addresses. Unlike public IP addresses that must be unique, private IP addresses are intended to be used simultaneously on multiple private networks. It doesn't matter if multiple networks use the same addresses, since the addresses won't ever be seen outside of the private network anyway. Private IP addresses solve the problem of an ISP only providing a single public IP address to a home or business, but how are private IP addresses useful if they aren't routable on the internet?

*Network Address Translation (NAT)* allows devices on a private network, often a home network, to all use the same public IP address on the internet. As packets flow through a NAT router, the router modifies the IP address information in those packets. When a packet originating from the private home network arrives at the NAT router, it modifies the source IP address field to match the public IP address, as shown in Figure 11-16.

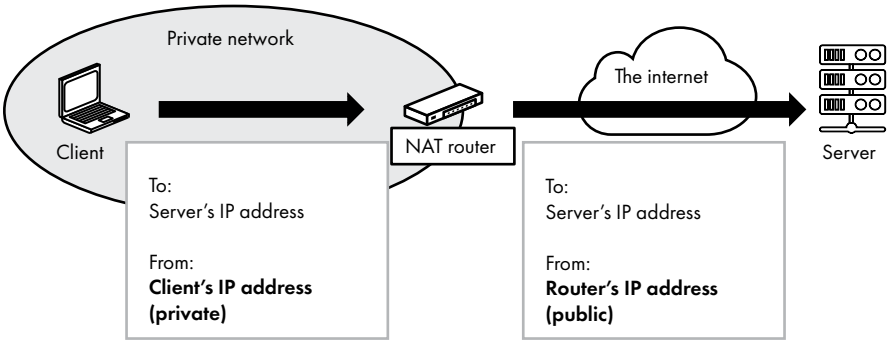



*Figure 11-16: A NAT router replaces private IP addresses with its own public IP address*

When a response comes back to the router, it sets the destination IP address to the private address of the host that originated the request. In this way, all traffic from the home appears to originate from the same public IP address, even if there are actually multiple devices on the private network. NAT also has the side benefit of security: the devices on the private network aren't directly exposed to the public internet, so a malicious user on the internet can't initiate a connection directly to a private device. Most routers sold to consumers for home use are NAT routers, often with built-in wireless access point capabilities as well.

Private IP addresses are valuable not only for home networks, but also for businesses that don't want their computers exposed to the public internet. Many corporate networks use a *proxy server* rather than a NAT router. A

proxy server is similar to a NAT router in that it allows devices on a private network to access the internet, but a proxy server differs in that it typically operates at the application layer rather than the internet layer. Proxies also usually provide additional features such as user authentication, traffic logging, and content filtering.

**NOTE** *Please see Project #34 on page 259, where you can see if the IP address your device is assigned is a public IP address or a private IP address.*

### The Domain Name System

We've seen that hosts on the internet are identified by IP addresses. However, most users of the internet rarely, if ever, directly deal with IP addresses. Although IP addresses work well for computers, they aren't very user friendly. No one wants to remember sets of four numbers separated by periods. Fortunately, we have the *Domain Name System (DNS)* to make things easier for us. DNS is an internet service that maps names to IP addresses. This allows us to refer to a host by a name like *www.example.com* rather than by its IP address.

A computer's full DNS name is known as a *fully qualified domain name*, or *FQDN*. A name like *travel.example.com* is an FQDN. This name is composed of a short, local *hostname (travel)* and a domain suffix *(example.com)*. The term *hostname* is often used interchangeably to mean either the computer's short name or the FQDN. Later in this section, we use *hostname* to mean a computer's FQDN. A *domain*, like *example.com*, represents a grouping of network resources managed by an organization. Both *example.com* and *travel.example.com* are domain names. The former represents a network domain; the latter represents a specific host on that domain.

Software needs to be able to query DNS to convert hostnames to IP addresses—this is known as *resolving* a hostname. To enable this functionality, hosts are configured with a list of the IP addresses of DNS servers. This list is usually provided by DHCP, and it typically is composed of DNS servers maintained by the internet service provider or running on the local network. When a client wants to connect to a server by name, it asks a DNS server for the IP address corresponding to that name. The server replies with the requested IP address, if it can. This is illustrated in Figure 11-17.

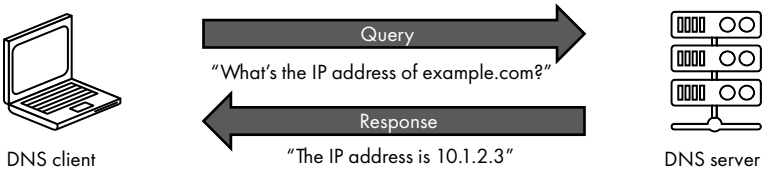



*Figure 11-17: A simplified DNS query. The IP address of* example.com *is not intended to be accurate.*

Once the client has the server's IP, it proceeds to communicate with the server using the IP address, as described earlier. I've heard DNS described as the phone book of the internet, although that analogy may fall short for some readers since phone books aren't as common as they once were!

You might assume that there's a one-to-one mapping between IP addresses and names. That actually isn't the case. A name can map to multiple IP addresses. In that scenario, different clients query DNS for a certain name, and they may all receive a different IP address as a response. This is useful for situations in which the load for a given service needs to be distributed across multiple servers. This can be done geographically, so that clients in Europe, for example, get a different IP address than clients in Asia, allowing clients in each region to connect to the IP address of a server that's physically close to them.

The reverse is possible too: multiple names can map to the same IP address. In this scenario, a query for different names may return a single IP address. This is useful when a server hosts multiple instances of the same type of service, each identified by name. This is common in web hosting, where a single server hosts multiple websites, each identified by its DNS name.

Each entry in DNS is known as a *record*. There are various kinds of records; the most basic is an *A record*, which simply maps a hostname to an IP address. Other examples are *CNAME (canonical name)* records that map one hostname to another hostname, and *MX (mail exchanger)* records used for email services.

No single organization would want to undertake the task of managing the many, many DNS records that exist today. Fortunately, this isn't needed; DNS is implemented in a way that allows for shared responsibility. A DNS name like *www.example.com* actually represents a hierarchy of records, and different DNS servers are responsible for maintaining the records at different levels of the hierarchy. The DNS hierarchy, as applied to *www.example.com*, is illustrated in Figure 11-18.

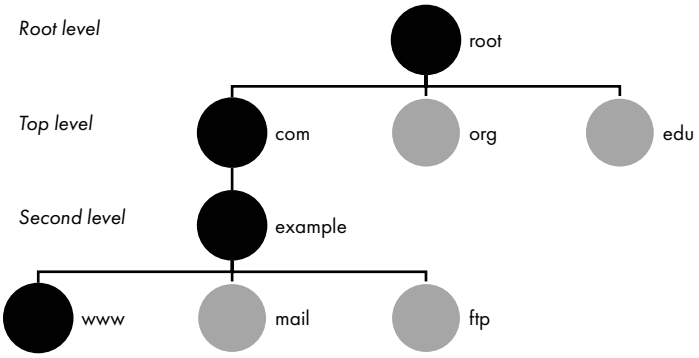



*Figure 11-18: Example DNS hierarchy, highlighting* www.example.com

At the top of this hierarchical tree is the *root domain*. The root domain doesn't get a textual representation in a DNS name like *www.example.com*, but it's an essential part of the DNS hierarchy. The root domain contains

records for all the *top-level domains (TLDs)* like *.com*, *.org*, *.edu*, *.net*, and so forth. As of 2020, there are 13 root name servers worldwide, each responsible for knowing the details of all the top-level domain servers. Let's say you want to look up a record in a domain that ends with *.com*. A root server can point you to a TLD server that knows about domains under *.com*.

A top-level DNS server is responsible for knowing about all the second-level domains under its hierarchy. A top-level DNS server for *.com* could point you to the second-level DNS server for *example.com*. The DNS servers for second-level domains maintain records for hosts and third-level domains that fall under second-level domains. This means that the DNS server(s) for *example.com* are responsible for maintaining the records for hosts like *www.example.com* and *mail.example.com*. This pattern continues, allowing for nested domains. Once a domain is registered under a top-level domain, the owner of that domain can create as many records as needed under their domain.

As mentioned earlier, when a computer needs to find the IP address for an FQDN, it sends a request to its configured DNS server. What does the DNS server do with this request? If the server has recently looked up the requested record, it may have a copy of that record stored in its cache, and it can immediately return the IP address to the client. If the DNS server doesn't have the response in cache, it may query other DNS servers as needed to get the answer. This involves starting at the root and working down the hierarchy of servers to find the record in question. Once the server has the record, it can cache it so that it can immediately respond to future queries for that record. Eventually the cached record is removed to ensure that the server always provides reasonably recent data.

**NOTE** *Please see Project #35 on page 260, where you can look up information in DNS.*

## Networking Is Computing

Let's take a moment to consider how the internet fits in with the broader picture of computing that we've already covered in this book. Networking may seem like a tangential topic, but really it isn't so far removed from computing in general. The internet is composed of hardware and software working together to allow communication between devices. Data sent over the internet boils down to 0s and 1s, represented in various forms such as voltages on a wire. From the perspective of a computer, a networking interface like a Wi-Fi or Ethernet adapter is just another I/O device. An operating system interacts with such adapters via device drivers, and the OS includes software libraries that allow applications to easily communicate over the internet. Networking devices like routers and switches are computers too, although highly specialized ones. The internet, and networking in general, is an extension of local computing, allowing for data transfer and processing beyond the boundaries of a single device.

## Summary

In this chapter we covered the internet, a globally connected set of computer networks that all use a suite of common protocols. You learned about the four layers of the internet protocol suite—the link layer, the internet layer, the transport layer, and the application layer. You saw how data travels through the internet and how devices interact at various layers. You learned how DHCP provides networking configuration data, how NAT allows devices on private networks to connect to the internet, and how DNS provides friendly names that can be used in place of IP addresses. In the next chapter you'll learn about the World Wide Web, a set of resources delivered by HTTP over the internet.

### PROJECT #29: EXAMINE THE LINK LAYER

Prerequisite: A Raspberry Pi, running Raspberry Pi OS. I recommend that you flip to Appendix B and read the entire "Raspberry Pi" section if you haven't already.

In this project, you'll use your Raspberry Pi to check out the link layer of your local network. Let's start with the following command, which lists the MAC address of your Ethernet adapter:

```
$ ifconfig eth0 | grep ether
```

The output should look something like the following:

```
        ether b8:27:eb:12:34:56  txqueuelen 1000  (Ethernet)
```

In this example, the MAC address is `b8:27:eb:12:34:56`. That's a hexadecimal representation of a 48-bit number. Remember, each hex character represents 4 bits, so that's 12 characters × 4 bits = 48 bits.

The first 24 bits of a MAC address represent the vendor/manufacturer of the hardware. This number is known as an *organizationally unique identifier (OUI)* and is managed by the *Institute of Electrical and Electronics Engineers (IEEE)*. In this case the OUI is B827EB, which is assigned to the Raspberry Pi Foundation. You can see the current OUI listings here: *http://standards-oui.ieee.org/oui.txt*.

Your Raspberry Pi's Wi-Fi adapter has its own MAC address. View it like this:

```
$ ifconfig wlan0 | grep ether
        ether b8:27:eb:78:9a:bc  txqueuelen 1000  (Ethernet)
```

On my system, the OUI (the first 24 bits of the MAC address) of the Wi-Fi adapter is the same as the OUI of the Ethernet adapter. This is because both adapters are internal Raspberry Pi hardware and use the OUI for the Raspberry Pi Foundation.

From your Raspberry Pi, you can also see the MAC address of other devices on your local network. To do this you can use a tool called `arp-scan` that attempts to connect to every computer on your local network and retrieve its MAC address.

First, install the tool:

```
$ sudo apt-get install arp-scan
```

Then run this command (that's a lowercase L at the end of the command, not the number 1):

```
$ sudo arp-scan -l
```

You should get a list of IP addresses (which we cover elsewhere in this chapter) and MAC addresses, plus a column that attempts to match the MAC prefix to the manufacturer. I got 10 results on my local network, some of which I didn't immediately recognize. You may see some duplicate results returned, indicated with `DUP` in the third column. The returned list typically does not include the address of the computer from which you ran the scan.

You may have some results in the third column that show as (Unknown). This means that the arp-scan tool wasn't able to match the OUI number to a known manufacturer, probably because the tool is using an outdated version of the OUI list. You can try to fix this by downloading the current list of OUI numbers from IEEE and then running the scan again, like this:

```
$ get-oui
$ sudo arp-scan -l
```

When I see multiple devices on my home network that I can't identify right away, I have an immediate urge to figure out what they are! As a bonus challenge for you, identify every device returned from arp-scan. Now, this may be impractical if you're running this tool on a network you don't control (say, at a coffee shop or library), but if you're at home, this is something you can do. You'll probably need to log on to each device on your network and dig through its settings to find its IP address or MAC address and see if it matches one of the entries returned from arp-scan. Hint: use the ifconfig utility on Linux or Mac, or the ipconfig tool on Windows. On mobile devices, look at the user interface for network settings.

### PROJECT #30: EXAMINE THE INTERNET LAYER

Prerequisite: A Raspberry Pi, running Raspberry Pi OS.

In this project, you'll look at the internet layer using your Raspberry Pi. Let's begin with the following command, which lists all the network interfaces on your device and their associated IP addresses.

```
$ ifconfig
```

You'll typically see three interfaces: eth0, lo, and wlan0. The lo interface is a special case; it's the *loopback* interface. It's used for processes running on the Pi that wish to communicate with each other using TCP/IP, but without actually sending any traffic over the network. That is, the traffic stays on the device. The loopback interface has an IP address of 127.0.0.1. This is a special address that is not routable and can't be used as an address on the local subnet, because any attempt to deliver messages to that address results in the messages coming right back to the sending computer. In other words, every computer considers 127.0.0.1 to be its own IP address.

As we covered in the previous project, eth0 is the wired Ethernet interface and wlan0 is the wireless Wi-Fi interface. If you're connected to a network on either or both of these interfaces, you should see an IP address beside the text inet in the ifconfig output. You may also see an IPv6 address listed beside inet6. Here's example wlan0 output from ifconfig:

```
wlan0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.1.138  netmask 255.255.255.0  broadcast 192.168.1.255
        inet6 fe80::8923:91b2:13e0:ed2a  prefixlen 64  scopeid 0x20<link>
```

*(continued)*

In this output you can see that the assigned IP address is `192.168.1.138`. The `netmask` value (subnet mask) is `255.255.255.0`, and the `broadcast` address is `192.168.1.255`.

The `ifconfig` command gives us information about the various network interfaces on the Raspberry Pi, but it doesn't tell us about how routing is configured. Let's take a look at that using the `ip route` command. I've included sample output here; your results may vary.

```
$ ip route
default via 192.168.1.1 dev wlan0 src 192.168.1.138 metric 303
192.168.1.0/24 dev wlan0 proto kernel scope link src 192.168.1.138 metric 303
```

This command's output can be a little difficult to interpret. In short, the first line gives the default route. This is where packets should be sent when there isn't a specific route that applies. In this particular example, every packet that doesn't match a specific routing rule should be sent to `192.168.1.1`. That means that `192.168.1.1` is the IP address of the local router, also known as the *default gateway*.

The next line is a routing entry that tells you that any packet sent to an IP address in the range of `192.168.1.0/24` should be sent through device `wlan0`. That's the Wi-Fi adapter on the local subnet. In other words, this routing rule ensures that communication with IP addresses on the local subnet happens directly, without going through a router.

To summarize, this output tells you that any packet sent to an IP address that matches `192.168.1.0/24` should be sent directly to the destination address via the `wlan0` interface. Any other traffic uses the default route, which sends traffic to the router at `192.168.1.1`. The end result is that local subnet traffic is sent directly to the target device, while traffic to devices on other subnets, likely on the internet, is sent to the default gateway.

## PROJECT #31: EXAMINE PORT USAGE

Prerequisite: A Raspberry Pi, running Raspberry Pi OS.

In this project, you'll see which network ports are in use on a Raspberry Pi. You'll then examine ports on other computers. Let's begin with the following command that shows you the listening and established TCP sockets on your Raspberry Pi.

```
$ netstat -nat
```

Let's break down the `-nat` options used in the command. The `n` option indicates that numeric output should be used to show the port numbers. The `a` option means show all connections (both listening and established), and `t` means limit the output to TCP. On my device, I see a list like so:

```
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp        0      0 0.0.0.0:22              0.0.0.0:*               LISTEN
tcp        0     36 192.168.1.138:22         192.168.1.125:52654    ESTABLISHED
tcp        0      0 192.168.1.138:22         192.168.1.125:51778    ESTABLISHED
tcp6       0      0 :::22                   :::*                    LISTEN
```

Here you see four sockets, all related to SSH. I can tell they are related to SSH because all the sockets are using port 22. I have SSH enabled on my Raspberry Pi to allow remote terminal connections. The first and last lines show that the Pi is listening on port 22 for new incoming SSH connections using both TCP and TCP over IPv6. The middle two lines show that I have two established SSH connections to this device, both of them from my laptop (with an IP of 192.168.1.125) to the Pi (with an IP of 192.168.1.138). Note how both the established connections go to the same server port on the Pi (22), whereas the client port on my laptop varies (52654 and 51778), as they are ephemeral ports.

Run the command again, this time adding the **p** option and prefixing the command with **sudo**:

```
$ sudo netstat -natp
```

This gives you the same list, but with the process ID (PID) and program name to which the socket belongs. Any traffic sent to the socket is directed to the PID, which handles the traffic and responds as needed. On my computer I see that the program using this port is sshd—the daemon for SSH.

Now that you've examined which ports are in use on your Raspberry Pi, let's examine the ports on a remote computer. For this, you'll use a tool called nmap, which must first be installed on your Raspberry Pi:

```
$ sudo apt-get install nmap
```

Once the tool is installed, select a target host that you wish to scan. This can be either a device on your network (say your router or a laptop) or a host on the internet. Note that repeatedly scanning a host that you don't control may look suspicious to the administrators of that server, so I strongly recommend that you only scan devices that you own.

In my case, I decided to scan my default gateway, which happens to be at 192.168.1.1. The following nmap command scans for open TCP ports on the specified IP address. Try this on your Raspberry Pi, replacing the IP address with the address of the device you wish to scan. If you want to scan your own router, see Project #30 for a reminder of how to get the IP address of your default gateway.

```
$ nmap -sT 192.168.1.1
```

A partial listing of the results from my scan showed these ports:

```
PORT      STATE SERVICE
53/tcp    open  domain
80/tcp    open  http
```

This tells me that the device acts not only as a router, but as a DNS server (port 53) and web server (port 80). It's normal for a home router to provide these services.

## PROJECT #32: TRACE THE ROUTE TO A HOST ON THE INTERNET

Prerequisite: A Raspberry Pi, running Raspberry Pi OS.

In this project, you'll examine the route a packet travels from your Raspberry Pi to a host on the internet. First, you need to choose a host on the internet. This can be a website like *www.example.com*, or the IP address or FQDN of any internet host you happen to know. Once you've decided on a host, enter the following command, replacing *www.example.com* with the name or IP address of the host you wish to see.

```
$ traceroute www.example.com
```

The `traceroute` tool attempts to show the routers that are encountered on a packet's journey across the internet. The output should be read line by line. Each line is sequentially numbered and shows the name (if available) and IP address of the router encountered at that step of the packet's trip. If there is no response after a short time, the output displays an asterisk (*) and moves on to the next router. You may also see more than one IP address per line, indicating multiple possible routes.

## PROJECT #33: SEE YOUR LEASED IP ADDRESS

Prerequisite: A Raspberry Pi, running Raspberry Pi OS.

In this project, you'll look at the lease information associated with your Raspberry Pi's IP address obtained from a DHCP server. Of course, this assumes that your Raspberry Pi is configured to use DHCP (which is the default) rather than a static IP address. To do this, look at the system log:

```
$ cat /var/log/syslog | grep leased
```

Expect to see output similar to the following:

```
Jan 24 19:17:09 pi dhcpcd[341]: eth0: leased 192.168.1.104 for 604800 seconds
```

Here you can see that IP address `192.168.1.104` was leased from a DHCP server for use on network interface `eth0`, the Ethernet interface on the Raspberry Pi. Your output likely shows a different IP address and perhaps a different interface, maybe `wlan0`.

By default, the *syslog* file is periodically cleared, and its contents are moved to a backup file. Because of this, you may not see a DHCP entry in your *syslog* file. You can release your current IP address, request a new one, and look again for the lease entry like so:

```
$ sudo dhclient -r wlan0
$ sudo dhclient wlan0
$ cat /var/log/syslog | grep leased
```

Replace `wlan0` with `eth0` if you want to do this for Ethernet rather than Wi-Fi.

## PROJECT #34: IS YOUR DEVICE'S IP PUBLIC OR PRIVATE?

Prerequisite: A Raspberry Pi, running Raspberry Pi OS.

In this project, you'll see if the IP address of your Raspberry Pi is public or private. If your device has a private IP address, you'll also find the public IP address that is used for your communication over the internet. As before, you can use the following utility to see your device's assigned IP address(es).

```
$ ifconfig
```

When looking for your device's assigned IP address, you'll likely see an entry for 127.0.0.1; you can ignore this one since it's used for loopback (see Project #30). As mentioned earlier, any address that matches the pattern of 10.*x.x.x*, 172.16.*x.x*, or 192.168.*x.x* is a private IP address. Now, even if you have a private IP address like one of these, when you access resources on the internet, you're also indirectly making use of a public IP address. This is the address that websites or other internet services see when you connect to them. If you're on a home network, that public IP address is likely assigned to your router. If you're on a business network, that public IP address may be assigned to a proxy device on the edge of your corporate network. In either case, all the network traffic from inside your local network to the internet originates from that public address.

To find the public IP address that your device uses when connecting to a device on the internet, one option is to log on to your router or proxy server and check its network configuration. If you know how to query your router or proxy server for this information, feel free to do so. However, since every model of network device is somewhat different, I won't walk you through the steps here.

A more universal option is to query an online service that can return your current public IP address. This is possible because every internet server that your device connects to knows your IP address; it's simply a matter of finding a service that's willing to tell you what IP address it sees. If you're running a web browser on your device, perhaps the simplest thing to do is query Google for something like "my IP address." This usually returns the information you want.

If you're working from a terminal, like on the Raspberry Pi, you can use the `curl` utility to make an HTTP request to a website that returns your current IP address. The following are a few examples of services that are available for this at the time of this writing:

```
$ curl http://ipinfo.io/ip
$ curl http://checkip.amazonaws.com/
$ curl http://ipv4.icanhazip.com/
$ curl http://ifconfig.me/ip
```

Any of these should return your public IP address to the terminal window. Compare this address with the address you got earlier from `ifconfig`. If they are the same, then your device is directly assigned a public IP address. If they differ, then your device likely has a private IP assigned to it, and you're connecting to the internet through a NAT router or proxy server.

### PROJECT #35: FIND INFORMATION IN DNS

Prerequisite: A Raspberry Pi, running Raspberry Pi OS.

In this project, you'll use your Raspberry Pi to query DNS records. Let's begin by looking up the IP address of a website. You'll use the `host` utility to do this. The following command returns the IP address of *www.example.com*, the hostname of the site I'm interested in. Feel free to substitute the name of another host you wish to look up.

```
$ host www.example.com
```

You should see output that gives the IP address of the host. You may also see an IPv6 address. Depending on the hostname you queried, you may get back multiple records, since a DNS name can map to multiple IP addresses. You may also learn that the name you typed is actually an alias for a different name, which in turn maps to an IP address.

DNS also allows for reverse lookups, where you specify an IP address and a hostname is returned. This doesn't always work, since DNS records need to be in place to support it. To give this a try, just use `host` with an IP address. In the following command, replace *a.b.c.d* with your public IP address that you found in Project #34, or any other public IP address you wish to query. Again, this works only for IP addresses that have DNS records in place to support reverse lookups.

```
$ host a.b.c.d
```

By default, the `host` utility uses the DNS server your device is configured to use. You may also query a specific DNS server using `host` by specifying the IP address of that server. Internet service providers include DNS services for their customers, but many free alternate DNS services are available as well. For example, at the time of this writing, Google provides a DNS server at 8.8.8.8 and Cloudflare provides a DNS server at 1.1.1.1. If you want to use the DNS server at 1.1.1.1 to look up *www.example.com*, you could enter this:

```
$ host www.example.com 1.1.1.1
```

This should output IP address information as before, along with some text indicating which DNS server was used for the lookup.

If you're curious about the details of the DNS query, you can use the `-v` option with the `host` command, which provides verbose output.

```
$ host -v www.example.com
```