Unity Networking Fundamentals

Creating Multiplayer Games with Unity

Sloan Kelly
Khagendra Kumar

Apress®

Unity Networking Fundamentals

Creating Multiplayer Games with Unity

Sloan Kelly
Khagendra Kumar

Apress®

Unity Networking Fundamentals: Creating Multiplayer Games with Unity

Sloan Kelly
Niagara Falls, ON, Canada

Khagendra Kumar
Katihar, Bihar, India

ISBN-13 (pbk): 978-1-4842-7357-9
ISBN-13 (electronic): 978-1-4842-7358-6
https://doi.org/10.1007/978-1-4842-7358-6

Managing Director, Apress Media LLC: Welmoed Spahr
Acquisitions Editor: Spandana Chatterjee
Development Editor: Laura Berendson
Coordinating Editor: Divya Modi

Cover designed by eStudioCalamar

Cover image designed by Pixabay

Distributed to the book trade worldwide by Springer Science+Business Media New York, 1 New York Plaza, Suite 4600, New York, NY 10004-1562, USA. Phone 1-800-SPRINGER, fax (201) 348-4505, e-mail orders-ny@springer-sbm.com, or visit www.springeronline.com. Apress Media, LLC is a California LLC and the sole member (owner) is Springer Science + Business Media Finance Inc (SSBM Finance Inc). SSBM Finance Inc is a **Delaware** corporation.

For information on translations, please e-mail booktranslations@springernature.com; for reprint, paperback, or audio rights, please e-mail bookpermissions@springernature.com.

Apress titles may be purchased in bulk for academic, corporate, or promotional use. eBook versions and licenses are also available for most titles. For more information, reference our Print and eBook Bulk Sales web page at http://www.apress.com/bulk-sales.

Any source code or other supplementary material referenced by the author in this book is available to readers on GitHub via the book's product page, located at www.apress.com/978-1-4842-7357-9. For more detailed information, please visit http://www.apress.com/source-code.

Printed on acid-free paper

Table of Contents

About the Authors

Sloan Kelly has worked in the games industry for more than 13 years. He worked on a number of AAA and indie titles and currently works for an educational game company. He lives in Ontario, Canada with his wife and children. Sloan is on Twitter `@codehoose` and makes YouTube videos in his spare time.

Khagendra Kumar has worked with a number of educational institutions and game studios for training and solutions. He lives in Bihar, India and spends most of his time working with game AI. He can be reached via LinkedIn at `/itskhagendra` and on Instagram `@Khagendra_Developer`.

About the Technical Reviewer

Simon Jackson is a long-time software engineer and architect with many years of Unity game development experience, as well as an author of several Unity game development titles. He both loves to both create Unity projects as well as lend a hand to help educate others, whether it's via a blog, vlog, user group, or major speaking event.

His primary focus at the moment is with the XRTK (Mixed Reality Toolkit) project, which is aimed at building a cross-platform Mixed Reality framework to enable both VR and AR developers to build efficient solutions in Unity and then build/distribute them to as many platforms as possible.

Introduction

This book sets out to demystify network programming and open you and your games up to the wider world using the Unity Engine and the C# programming language. The .NET framework that C# sits on top of has several classes that make creating networked games a little easier.

Intended Audience

This book is intended for Unity developers who are familiar with C# and want to implement their own networking framework, or those who want to have a better understanding of low-level network programming.

This is meant to be an introductory guide to networking. The book concentrates mostly on developing smaller games that can be run on your local network rather than larger multiplayer games played over the Internet. These basic concepts will help you better understand the underlying technology behind multiplayer games and the inherent constraints involved in passing data across a network. The last chapter of the book covers making your game available on the Internet using a third-party service.

Software and Hardware Requirements

The examples in this book were written using Unity 2020.1.6f1. I used both a MacBook Pro (mid-2012, Intel i7) and a PC (mid-2015, Intel i5) during

the writing of this book and the examples. To run Unity, you will need a device that meets the following requirements:

- *Windows*: Windows 7 (SP1+)/Windows 10, 64-bit only, X64 architecture CPU with SSE2 instruction set support, DX10-, DX11-, or DX12-compatible GPU.
- *MacOS*: High Sierra 10.13+, X64 architecture CPU with SSE2 instruction set support, Metal-capable Intel or AMD GPU.
- *Linux (preview support)*: Ubuntu 16.04, 18.04, or CentOS 7, X64 architecture CPU with SSE2 instruction set support, OpenGL 3.2+ or Vulkan-compatible NVIDIA and AMD GPU, GNOME desktop environment. Requires proprietary NVIDIA or AMD graphics driver.

Be sure to check the Unity system requirements page for the most up-to-date information.

How This Book Is Organized

The book is organized into the following chapters:

- Networking basics
- Serialization
- `UnityWebRequest` and RESTful APIs
- Connected services with TCP
- Connectionless services with UDP
- Common networking issues
- First person maze shooter
- Remote connections

Source Code

The source code for this book is available on GitHub via the book's product page, located at www.apress.com/978-1-4842-7357-9. The source code contains everything you need to build the following:

- Real-time weather app
- Networked tic-tac-toe
- First person maze shooter
- Basic TCP and UDP examples

Conventions Used In This Book

Various typefaces and styles are used in this book to identify code blocks, warnings, and other notices.

C# code is written in this style:

```
if (player.IsLoggedIn)
{
        print("Player is logged in");
        server.PlayerAttached(player.ID);
}
```

This book contains a list of some tools that come with your operating system to help you. These all run from the command line, also known as the terminal or DOS prompt depending, on your operating system. Command lines are written in the following style. The $ at the beginning of the line should not be typed:

```
$ ls -al
```

If an issue needs special attention, the following block is used:

Note This is a call out and will alert you to any information that is important.

CHAPTER 1

Networking Concepts

This chapter covers the very basics of networking and introduces some tools that are beneficial when you need to debug your application. This chapter includes a brief overview of the client-server model, discusses how we will build the games in this book, and covers networking fundamentals that will help you understand and debug your games when developing the networking components.

By the end of this chapter, you will be familiar with the devices used to connect your PC to the outside world, how Internet addressing works, and what the client-server model is.

If you have an email account, surf the web, use social media, or play online games, you have used networking components. The modern Internet runs on a suite of protocols based on Transportation Control Protocol/Internet Protocol (TCP/IP).

Internet browsers like Chrome and Firefox use HTTP and HTTPS (HyperText Transport Protocol and HyperText Transport Protocol Secure, respectively) to communicate with remote servers. As shown in Figure 1-1, an encyclopedia is just a click away!

S. Kelly and K. Kumar, *Unity Networking Fundamentals*,
https://doi.org/10.1007/978-1-4842-7358-6_1

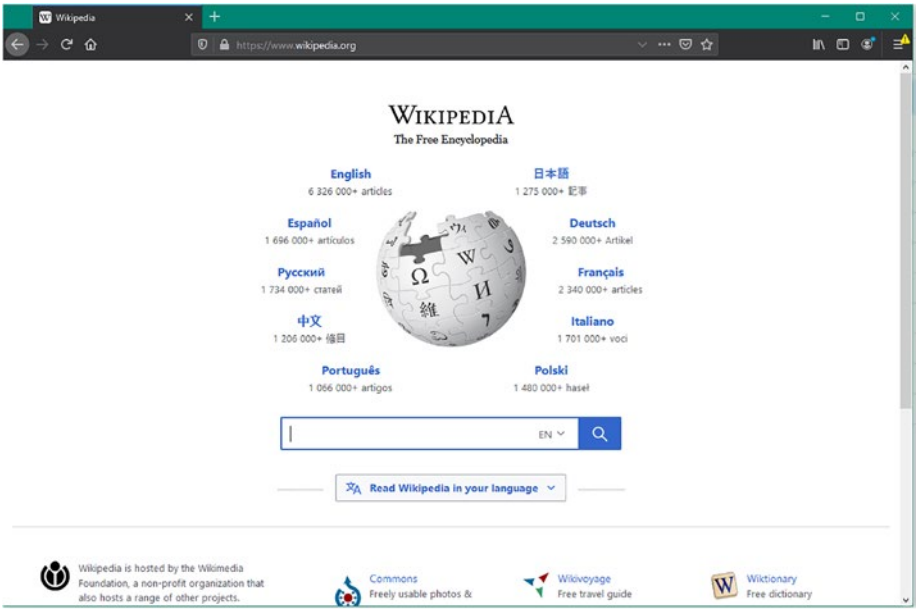

Figure 1-1. *A Firefox browser containing the wikipedia.org home page*

The Firefox (or Chrome, Safari, or Edge) browser is a *client* that requests a document from a *server*. The server sends data back to the client. The client takes that data and renders it to the window. This is called the *client-server model*.

Client-Server Model

The client-server model is a distributed application structure. The responsibilities are divided between the client and the server. The client typically makes requests to the server and then displays the fetched information for the user. The server takes the request and processes it, returning the data back to the client. The server itself rarely displays any output.

Typically, the client and server run on two separate machines and are linked using a computer network. However, it is entirely possible for the client and the server to run on the same machine.

The client-server model allows multiple clients to access a single resource. For example, printers, documents on a hard drive, and remote compute power.

The client and the server need to speak the same language. In the case of a web client/server, this is HTTP/HTTPS, depending on the security of the site. To retrieve a document, users type an address into the address bar of the browser:

```
https://www.wikipedia.org
```

The `https://` indicates the protocol that will be used when communicating with the server. The part after the double slashes is the address of the server. The web client – the browser – will then make a request to the remote server using the HTTP language.

The request part of the message uses a verb like `GET` or `POST` and a resource name. For example:

```
GET /index.html HTTP/1.1
```

This statement says "Fetch me the `index.html` page using the HTTP version 1.1. protocol." The server will interpret this message and return the document to the client, as shown in Figure 1-2.

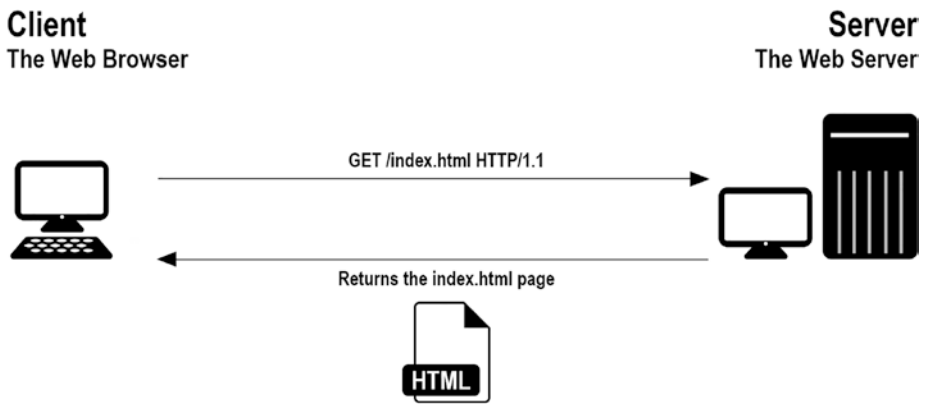

Figure 1-2. *The sequence of events fetching a document from a web server*

The web is one example of a client-server model.

The examples in this book show you how to create your own client-server model based games and how to write your own protocols to allow them to communicate effectively. In games, each player connects using a client to a remote server that contains all the rules for the game. Clients send movement information to the game server and the server updates the clients with new position and state data.

Connected vs. Connectionless Services

When you send a message across the Internet, it is split into many smaller messages called *packets*. These packets are routed all over the network and arrive at the destination. Some packets can take longer to arrive; some packets never make it. This is not good if you want your web page to arrive in one piece. Luckily, the web is a connection-oriented service and it guarantees that your message will arrive. Let's take a deeper look at connection- and connectionless-oriented services.

Packets

Your message is split into small packets and routed through the network. This is done for several reasons. It could be because a route to the server is blocked or because it is more efficient to group the packets and send them later when reaching a node in the network.

Your packets can arrive at the destination out of order, as shown in Figure 1-3.

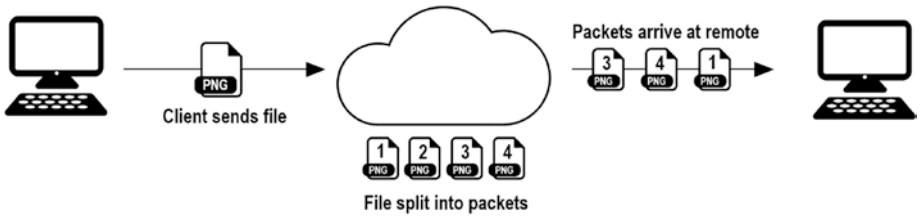

Figure 1-3. *File split into smaller packets, packet loss and packets received out of sequence*

In the example in Figure 1-3, the file is sent across the network in smaller packets numbered 1, 2, 3, and 4. As they travel through the network, Packet 2 is lost and Packet 4 arrives before Packet 3.

Sometimes packet loss is acceptable and other times it is not. It is up to the application developer to decide if packet loss or packets received out of sequence are acceptable side-effects. The developer will choose either a connection-oriented or connectionless-oriented approach, depending on the needs of the game. In a fast-moving game, some packet loss might be acceptable for the sake of maintaining speed.

Connection-Oriented Service

If you need to guarantee that messages arrive at the remote device in the correct order with no parts missing, you need to write a connection-oriented service.

Anything that uses Transport Control Protocol (TCP) will guarantee that packets arrive in order and the message you send will be intact. An example of an application that uses TCP is the web. HTTP is built using TCP to guarantee that messages arrive intact.

Using TCP, your client must establish a connection with a server to allow communication to flow between them. The connection can last any amount of time, from a couple of seconds to days. The connection is required to ensure that data is sent and received.

TCP provides error-free data transmission. If packets are dropped or corrupted, they are retransmitted. When packets arrive at their destination, the sender is notified.

Note The error-free data transmission process is handled by the TCP protocol. Your code does not have to retransmit dropped packets. This is all handled for you by TCP on your machine's operating system.

As an example, the client in Figure 1-4 is transferring a PNG file to a remote server. A packet is dropped and the receiver sends a message back to the sender asking it to resend Packet 2. The sender obliges and resends Packet 2.

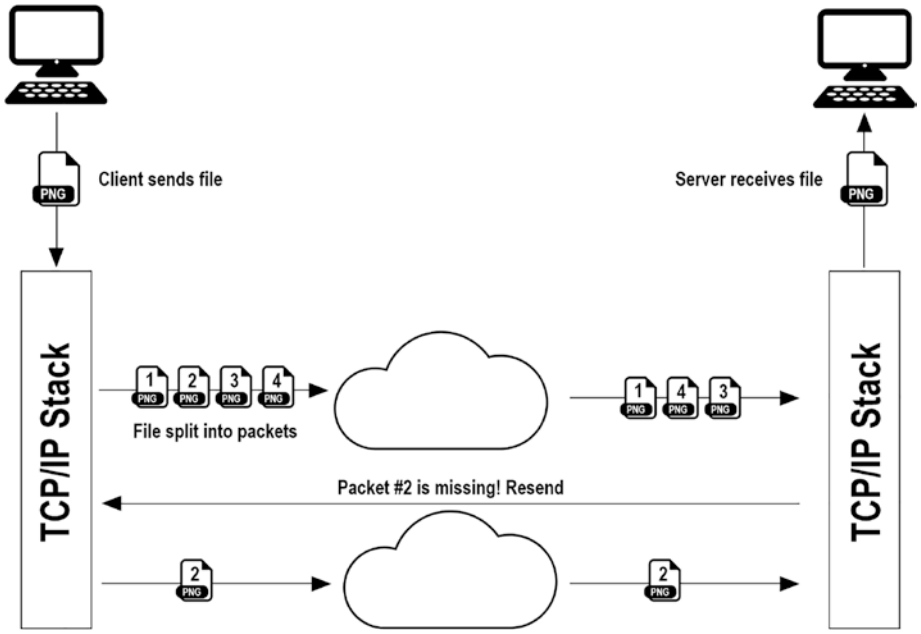

Figure 1-4. *Sequence showing the recovery of a dropped packet using TCP*

If your application needs to guarantee that transmitted messages appear in the correct order and are complete, use a TCP connection. Other applications for TCP include:

- Chat applications
- File transfer
- Mail services

The TCP/IP suite of applications is an example of a connection-oriented service. Example services are:

- POP/IMAP/SMTP for mail transfer
- FTP for file transfer
- HTTP for delivering web pages

The downside is that TCP is slower, due to the handshaking and confirmation messages that pass between the client and the server. If you do not care about the order of packets or don't care if any packets are dropped along the way, you can use a faster connectionless-oriented service.

Connectionless-Oriented Service

There are times when you do not need to guarantee delivery of packets. If they can arrive out of order, or not at all, you should consider creating a connectionless-oriented service.

In a connectionless-oriented service, the client does not connect with a server, it just sends the information. If the server cannot receive the packet, then it's lost. Because there is no connection, the number of messages sent per packet transferred is always just one – the packet being transferred.

Connectionless-oriented services are used for things like:

- Video streaming
- Multiplayer games

A video stream sends a minimum of 24 frames per second. It must get there very quickly ($1/24^{th}$ of a second) and so if one frame is lost there is not enough time to ask for another.

In multiplayer games, it is often too much overhead to use a TCP connection for game play. If the player input is sent at 60 frames per second (fps), then the occasional dropped packet will not make much difference. As you will see later in the book, there are ways around this.

The IP part of the TCP/IP offers a connectionless-oriented protocol called User Datagram Protocol (UDP). UDP allows developers to send so-called "fire and forget" messages to remote machines. There is no guarantee that the packets will arrive on time, or in sequence. If that is a sacrifice you're willing to make for speed, then UDP is a perfect choice.

Physical Network Devices

Devices like your mobile phone, PC, and tablet need to connect to a network. They do this using a *network card*. The formal name of this is a *network interface card* (NIC). Your device will connect to a local device called a *router* using either a WiFi (wireless) or an Ethernet (wired) connection.

Your router might be part of a cable or ADSL modem or a separate device altogether. The modem – short for modulator/demodulator – is a device that turns the received zeros and ones into wavelengths that can be passed down a wire and onto the Internet. Figure 1-5 shows a typical network diagram for a home network and a connection to a remote server like `www.google.com`.

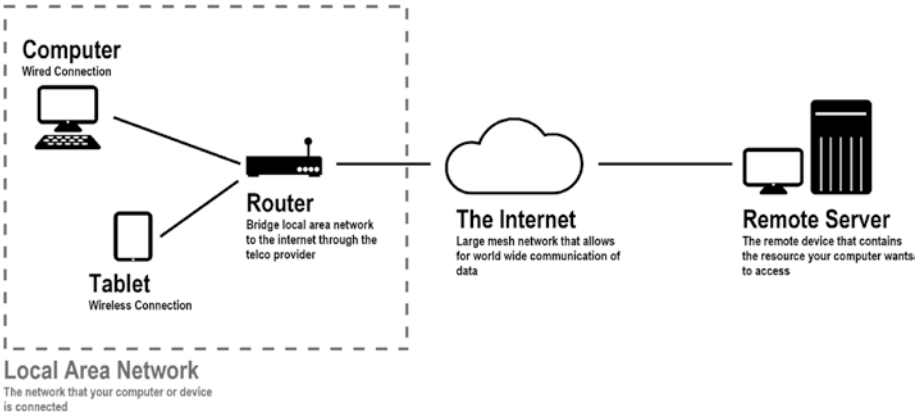

Figure 1-5. *Devices on your Local Area Network (LAN) connect through a router to the Internet*

All the traffic from your local devices to a remote server travels through the router. The router also provides another function that provides each connected device with a unique address on the LAN. This function is called *Dynamic Host Control Protocol*. This address will be used by other machines to communicate with each other.

There are two Internet Protocol address formats: IPv4 and IPv6. Each device will have an IPv4 and IPv6 address. This book uses IPv4 addresses. We cover addressing in the next section.

IPv4 and IPv6 are *logical addresses*. Each network card is assigned a *physical address* at the factory. This address is called the Media Access Control (MAC) address.

The architects of the original Internet — called DARPANet, short for Defense Advanced Research Projects Agency Network — used a mesh network, as shown in Figure 1-6.

LANs are connected to the Wider Area Network (WAN) or to the Internet using routers. The routers pass messages between each other until the destination is reached. This is why they are called *routers*; they route messages between nodes on the network. A *node* is a device connected to the network and can be a router or computer or any other network-accessible device.

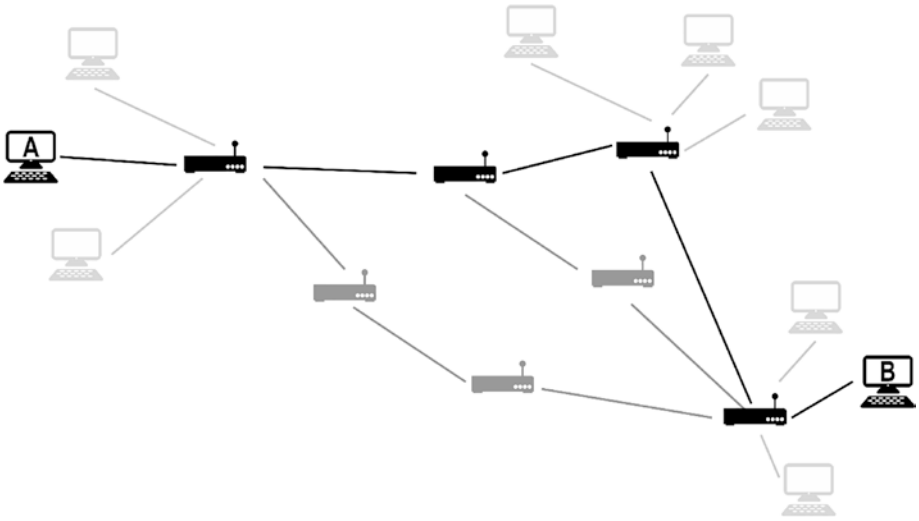

Figure 1-6. *Mesh network routing past disabled routers from Device A to B*

The mesh network allows the messages to be routed past broken or inaccessible parts of the network. When Device A wants to send a message to Device B, the devices on the network will reroute messages past the inactive nodes. Later, you will learn about a command-line tool that shows how messages are routed to a remote server.

The mesh network is an example of a network *topology*. Topology is just a fancy way of saying the shape of something. There are other network topologies:

- Bus – Each node on the network is connected to a single cable and T-connectors are used to connect PCs and other devices to the network.
- Ring – Data travels around the ring in one direction. Each device acts as a repeater to keep the signal strong. Every node is a critical link in the network.
- Star – This is the most common setup and the one that you have in your home. It's a central device, usually a router, connected to a larger network (your Internet Service Provider). Each local device connects to the router and thus out to the larger network.
- Tree – The tree topology is a combination of the star and bus.

Network Addressing

When you enter an address in the address box at the top of your browser and press Return, the page you requested appears after a few seconds. But how does the browser know where to go? This section explores the addresses used on the network, specifically with respect to the Internet.

So far, we have talked at a high level about data passing through a network using connection and connectionless services. How are these connected devices identified on the network? This section looks at how the IPv4 address system works and discusses the "uniqueness" of the number.

To connect to a network using TCP/IP, a device needs a network card of some kind. It can be wired or wireless. Each network card is given a "unique" hardware ID number called a MAC address. A network card is an electronic device that connects devices like computers, mobile phones, and games consoles to a computer network.

Media Access Control (MAC) Address

The MAC address is a group of six hexadecimal digits, like 01-23-45-67-89-ab. For example, the MAC address of the network card on my PC is C8-60-00-D0-5E-A5. MAC addresses are burned into the card and cannot be altered. This is the *physical address* of your device. The physical address is what identifies the device on the network.

The first three-digits of the MAC address identify the manufacturer. C8-60-00 is *ASUSTek Computer Inc.* That is the manufacturer of my computer's motherboard.

Because there are only six hexadecimal digits left, it would be impossible for manufacturers to give each device a unique physical ID. Instead, what they do is ship batches of network cards to different parts of the world to minimize the chances of two devices in the same location having the same address.

IP Address

The IP protocol uses a *logical address* to access devices on the network. This logical address is known as the IP address of the device. There are two ways to assign an IP address to a device; static and dynamic.

Static IP Addresses

The IP address can be set on the machine. This is a static address. This is usually only done for servers because these devices are known as endpoints in the network.

Dynamic IP Addresses

Dynamic IP addresses are assigned to each device when they boot up. The TCP/IP stack reaches out to the network to find a DHCP (Dynamic Host Control Protocol) server. The DHCP server assigns an address to the client. The dynamic addresses have a lease time, which means they expire and need to be renewed.

On my LAN, my PC seems to be given the same IP address, but it might not be the same on the network where your machine is connected.

IP Address Format

This book concentrates on IPv4 rather than IPv6. There are minor changes to the code to get it to run for IPv6, a flag or two to set.

The IPv6 address is much longer than its v4 counterpart. It consists of eight groups of four hexadecimal digits. An example IPv6 address is as follows:

```
1234:5678:9abc:def0:1234:5678:9abc:def0
```

On the contrary, IPv4 uses only four bytes separated by a period (.), such as:

```
192.168.1.1
```

Each digit in the IPv4 address is called an *octet* because it contains eight bits (one byte). The address's format is called *dotted decimal* because it contains four decimals separated by full stops. Four bytes is the same amount of space as an integer. This means that an IPv4 address can access

up to 2^32 or 4.3 billion devices. But wait – aren't there more devices in existence than that? What about all the IoT (Internet of Things) devices like light bulbs, toasters, fridges, and the like?

It turns out that IPv4 was not enough and that is why we moved to IPv6. IPv4 gets around its limited address space by using *network segmentation*.

Note For the remainder of this book, when referring to an IP address, it means an IPv4 address unless otherwise stated.

Address Classification

If you look at your machine's IP address using the `ipconfig` or `ifconfig` command (depending on your operating system), it is probably going to be something like 192.168.1.17 and your router is probably going to be located at address 192.168.1.1. A magic trick? No – most routers default to the Class C network, which is 192.168.1.x.

The IP addresses are split into several ranges. Each range represents the number of networks and the number of hosts that each network can contain. A host is just another name for a device. These ranges are called *classes*. There are also special IP addresses that you cannot use for your machine.

There are five classes of networks in the available IPv4 address ranges, called Classes A through E. Classes A to C are the ones most used because D and E are reserved classes. Table 1-1 shows each classification and describes what it means with respect to the available networks in that class and the number of hosts allowed per network.