Fundamentals of Web Development

Randy Connolly • Ricardo Hoar

ALWAYS LEARNING

PEARSON

ONLINE ACCESS

Thank you for purchasing a new copy of ***Fundamentals of Web Development, 1/e*, Global Edition**. Your textbook includes eighteen months of prepaid access to the book's Companion Website. This prepaid subscription provides you with full access to the following student support areas:

- Online Labs
- Case Studies
- Source Code

Use a coin to scratch off the coating and reveal your student access code.
Do not use a knife or other sharp object as it may damage the code.

To access the ***Fundamentals of Web Development, 1/e*, Global Edition,** Companion Website for the first time, you will need to register online using a computer with an Internet connection and a web browser. The process takes just a couple of minutes and only needs to be completed once.

1. Go to **http://www.pearsonglobaleditions.com/connolly**
2. Click on **Companion Website**.
3. Click on the **Register** button.
4. On the registration page, enter your student access code* found beneath the scratch-off panel. Do not type the dashes. You can use lower- or uppercase.
5. Follow the on-screen instructions. If you need help at any time during the online registration process, simply click the **Need Help?** icon.
6. Once your personal Login Name and Password are confirmed, you can begin using the ***Fundamentals of Web Development*** Companion Website!

To log in after you have registered:

You only need to register for this Companion Website once. After that, you can log in any time at **http://www.pearsonglobaleditions.com/connolly** by providing your Login Name and Password when prompted.

*Important: The access code can only be used once. This subscription is valid for eighteen months upon activation and is not transferable. If this access code has already been revealed, it may no longer be valid.

Fundamentals of Web Development

Fundamentals of Web Development

Randy Connolly
Mount Royal University, Calgary

Ricardo Hoar
Mount Royal University, Calgary

Global Edition contributions by

Soumen Mukherjee
RCC Institute of Information Technology, Kolkata

Arup Kumar Bhattacharjee
RCC Institute of Information Technology, Kolkata

PEARSON

Boston Columbus Indianapolis New York San Francisco Upper Saddle River
Amsterdam Cape Town Dubai London Madrid Milan Munich Paris Montreal Toronto
Delhi Mexico City São Paulo Sydney Hong Kong Seoul Singapore Taipei Tokyo

Editorial Director: *Marcia Horton*
Acquisitions Editor: *Matt Goldstein*
Editorial Assistant: *Kelsey Loanes*
Program Manager: *Kayla Smith-Tarbox*
Marketing Coordinator: *Jon Bryant*
Managing Editor: *Scott Disanno*
Head, Learning Asset Acquisition, Global Edition: *Laura Dent*
Acquisitions Editor, Global Edition: *Karthik Subramaniun*
Project Editor, Global Edition: *Anuprova Dey Chowdhuri*
Operations Supervisor: *Vincent Scelta*
Manufacturing Buyer: *Linda Sager*
Text Designer: *Jerilyn Bockorick, Cenveo® Publisher Services*
Cover Designer: *Shree Mohanambal Inbakumar, Lumina Datamatics*
Manager, Rights and Permissions: *Timothy Nicholls*
Text Permission Coordinator: *Jenell Forschler*
Cover Art: *© Robert Kneschke/Shutterstock*
Full-Service Project Management: *Hardik Popli, Cenveo Publisher Services*
Interior Printer/Bindery: *Neografia*
Cover Printer: *Neografia*

Pearson Education Limited
Edinburgh Gate
Harlow
Essex CM20 2JE
England

and Associated Companies throughout the world

Visit us on the World Wide Web at:
www.pearsonglobaleditions.com

ISBN 10: 1292057092
ISBN 13: 978-1-29-205709-5

10 9 8 7 6 5 4 3 2 1
14 13 12 11 10

British Library Cataloguing-in-Publication Data
A catalogue record for this book is available from the British Library

Typeset in 10 SabonLTStd-Roman by Cenveo Publisher Services

Printed and bound by Neografia in Slovakia.

The publisher's policy is to use paper manufactured from sustainable forests.

To Janet, for your intelligence, support, beauty, and love.

Randy Connolly

Thanks be to you Joanne for the love and joy you bring to our family.

Ricardo Hoar

Brief Table of Contents

Table of Contents

Preface

Welcome to the *Fundamentals of Web Development*. This textbook is intended to cover the broad range of topics required for modern web development and is suitable for intermediate to upper-level computing students. A significant percentage of the material in this book has also been used by the authors to teach web development principles to first-year computing students and to non-computing students as well.

One of the difficulties that we faced when planning this book is that web development is taught in a wide variety of ways and to a diverse student audience. Some instructors teach a single course that focuses on server-side programming to third-year students; other instructors teach the full gamut of web development across two or more courses, while others might only teach web development indirectly in the context of a networking, HCI, or capstone project course. We have tried to create a textbook that supports learning outcomes in all of these teaching scenarios.

What Is Web Development?

Web development is a term that takes on different meanings depending on the audience and context. In practice, web development requires people with complementary but distinct expertise working together toward a single goal. Whereas a graphic designer might regard web development as the application of good graphic design strategies, a database administrator might regard it as a simple interface to an underlying database. Software engineers and programmers might regard web development as a classic software development task with phases and deliverables, where a systems administrator sees a system that has to be secured from attackers. With so many different classes of user and meanings for the term, it's no wonder that web development is often poorly understood. Too often, in an effort to fully cover one aspect of web development, the other principles are ignored altogether, leaving students without a sense of where their skills fit into the big picture.

A true grasp of web development requires an understanding of multiple perspectives. As you will see, the design and layout of a website are closely related to the code and the database. The quality of the graphics is related to the performance and configuration of the server, and the security of the system spans every aspect of

development. All of these seemingly independent perspectives are interrelated and therefore a web developer (of any type) should have a foundational understanding of all aspects, even if they only possess expertise in a handful of areas.

Features of the Book

To help students master the fundamentals of web development, this book has the following features:

- **Covers both the concepts and the practice of the entire scope of web development.** Web development can be a difficult subject to teach because it involves covering a wide range of theoretical material that is technology independent as well as practical material that is very specific to a particular technology. This book comprehensively covers both the conceptual and practical side of the entire gamut of the web development world.
- **Focused on the web development reality of today's world and in anticipation of future trends.** The world of web development has changed remarkably in the past decade. For instance, fewer and fewer sites are being created from scratch; instead, a great deal of current web development makes use of existing sophisticated frameworks and environments such as jQuery, WordPress, HTML5, and Facebook. We believe it is important to integrate this new world of web development into any web development textbook.
- **Sophisticated, realistic, and engaging case studies.** Rather than using simplistic "Hello World" style web projects, this book makes extensive use of three case studies: an art store, a travel photo sharing community, and a customer relations management system. For all the case studies, supporting material such as the business cases, use cases, design documentation, visual design, images, and databases are included. We have found that students are more enthusiastic and thus work significantly harder with attractive and realistic cases.
- **Comprehensive coverage of a modern Internet development platform.** In order to create any kind of realistic Internet application, readers require detailed knowledge of and practice with a single specific Internet development platform. This book covers HTML5, CSS3, JavaScript, and the LAMP stack (that is, Linux, Apache, MySQL, and PHP). Other important technologies covered include jQuery, XML, WordPress, Bootstrap, and a variety of third-party APIs that include Facebook, Twitter, and Google and Bing Maps.
- **Content presentation suitable for visually oriented learners.** As long-time instructors, the authors are well aware that today's students are often extremely reluctant to read long blocks of text. As a result, we have tried to

make the content visually pleasing and to explain complicated ideas not only through text but also through diagrams.

- **Content that is the result of over twenty years of classroom experience** (in college, university, and adult continuing education settings) teaching web development. The book's content also reflects the authors' deep experience engaging in web development work for a variety of international clients.
- **Tutorial-driven programming content available online**. Rather than using long programming listings to teach ideas and techniques, this book uses a combination of illustrations, short color-coded listings, and separate tutorial exercises. These step-by-step tutorials are not contained within the book, but are available online at www.pearsonglobaleditions.com/connolly. Throughout the book you will find frequent links to these tutorial exercises.
- **Complete pedagogical features for the student**. Each chapter includes learning objectives, margin notes, links to step-by-step tutorials, advanced tips, keyword highlights, end-of-chapter review questions, and three different case study exercises.

Organization of the Book

The chapters in *Fundamentals of Web Development* can be organized into three large sections.

- **Foundational client-side knowledge (Chapters 1–7)**. These first chapters cover the foundational knowledge needed by any web developer. This includes how the web works (Chapter 1), HTML (Chapters 2 and 4), CSS (Chapters 3 and 5), JavaScript (Chapter 6), and web media (Chapter 7). Not every course would need to cover each of these chapters. Depending on the course, some instructors might skip Chapters 1, 5, 6, or 7.
- **Essential server-side development (Chapters 8–13)**. Despite the increasing importance of JavaScript-based development, learning server-side development is still the essential skill taught in most web development courses. The basics of PHP are covered in Chapters 8 and 9. Object-oriented PHP is covered in Chapter 10, and depending on the instructor, could be skipped (though PHP classes and objects are used in places in subsequent chapters). Database-driven web development is covered in Chapter 11, while state management and error handling are covered in Chapters 12 and 13.
- **Specialized topics (Chapters 14–21)**. Contemporary web development has become a very complex field, and different instructors will likely have different interest areas beyond the foundational topics. As such, our book provides specialized chapters that cover a variety of different interest areas.

Chapter 14 covers web application design for those interested more in software engineering and programming design. Chapter 15 includes advanced JavaScript and jQuery programming. Chapter 16 covers the vital topic of web security. Chapter 17 covers another programming topic: namely, consuming and creating web services. Chapter 18 covers the increasingly important topic of integrating with (and customizing) content management systems. The next two chapters address two important non-development topics: web server administration (Chapter 19) and search engines (Chapter 20). Finally, Chapter 21 covers another increasingly important topic: how to integrate a site into third-party social networks.

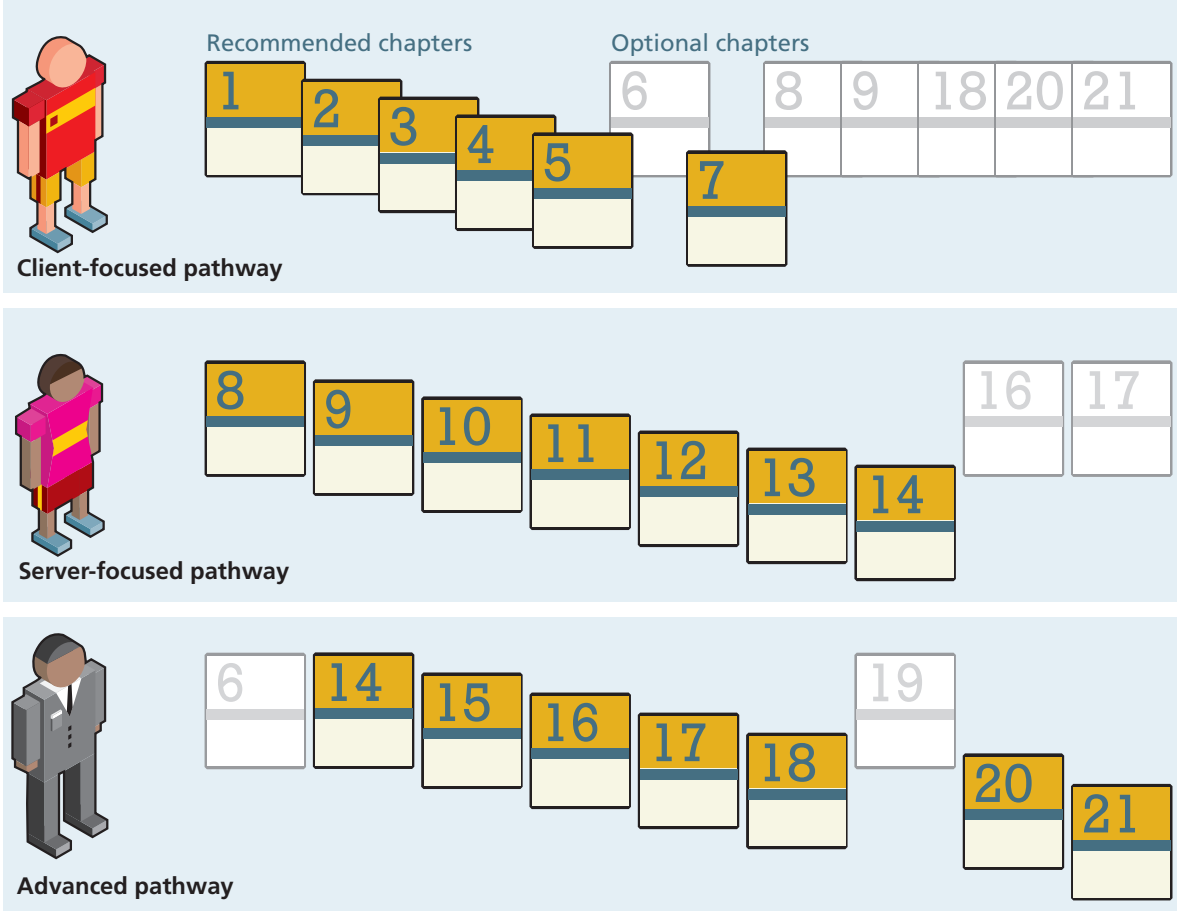
Pathways through this Book

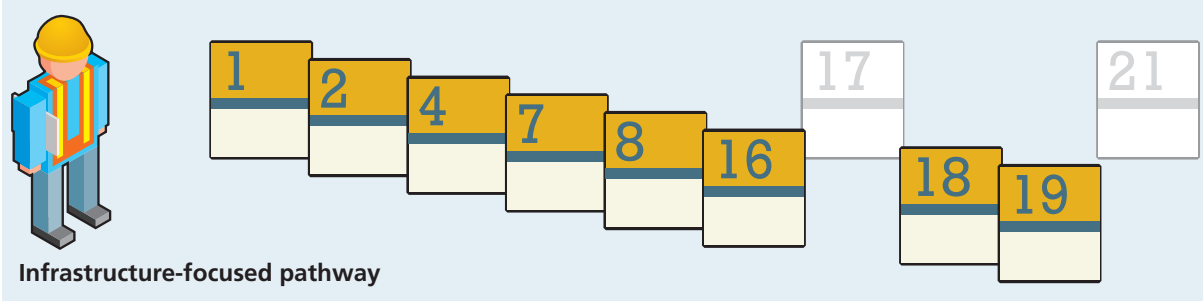
There are many approaches to teach web development and our book is intended to work with most of these approaches. It should be noted that this book has more material than can be plausibly covered in a single semester course. This is by design as it allows different instructors to chart their own unique way through the diverse topics that make up contemporary web development.

We do have some suggested pathways through the materials (though you are welcome to chart your own course), which you can see illustrated in the pathway diagrams.

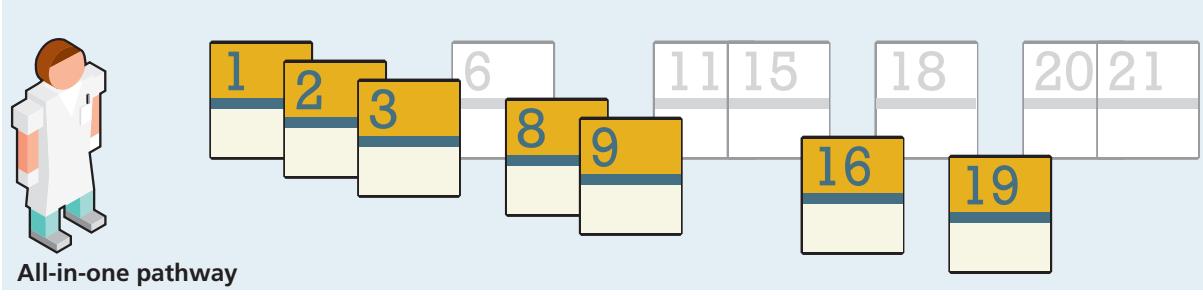
- **All the web in a single course.** Many computing programs only have space for a single course on web development. This is typically an intermediate or upper-level course in which students will be expected to do a certain amount of learning on their own. In this case, we recommend covering Chapters 1, 2, 3, 4, 8, 9, 11, and 13. A semester-long course might also cover Chapters 6 and 16 as well.
- **Client-focused course for introductory students.** Some computing programs have a web course with minimal programming that may be open to non-major students or which acts as an introductory course to web development for major students. For such a course, we recommend covering Chapters 1, 2, 3, 4, 5, and 7. You can use Chapter 6 to introduce client-side scripting if desired. If some server-side web programming is going to be introduced, you can also cover Chapters 8 and 9. If no programming is going to be covered, you might consider adding some parts of Chapters 18, 20, and 21.
- **Server-focused course for intermediate students.** If students have already taken a client-focused course (or you want the students to learn the client content quickly on their own), then Chapters 8–14 and perhaps Chapters 16 and 17 would provide the students with a very solid foundation in server-side development.

- **Advanced web development course.** Some programs offer a web development course for upper-level students in which it is assumed that the students already know the foundational topics and are also experienced with the basics of server-side development. Such courses probably have the widest range of possible topics. One example of such a course that we have taught covers the content in Chapters 6 14–18, and 20–21.
- **Infrastructure-focused course.** In some computing programs the emphasis is less on the particulars of web programming and more on integrating web technologies into the overall computing infrastructure within an organization. Such a course might cover Chapters 1, 2, 4, 7, 8, 16, 18, 19, and part of Chapters 17 and 21.





Infrastructure-focused pathway



All-in-one pathway

For the Instructor

Web development courses have been called "unteachable" and indeed teaching web development has many challenges. We believe that using our book will make teaching web development significantly less challenging.

The following instructor resources are available at www.pearsonglobaleditions.com/connolly:

- Attractive and comprehensive PowerPoint presentations (one for each chapter).
- Images and databases for all the case studies.
- Solutions to end-of-chapter exercises and to tutorial exercises.

Why This Book?

The ACM computing curricula for computer science, information systems, information technology, and computing engineering all recommend at least a single course devoted to web development. As a consequence, almost every post-secondary computing program offers at least one course on web development.

Despite this universality, we could not find a suitable textbook for these courses that addressed both the theoretical underpinnings of the web together with modern web development practices. Complaints about this lack of breadth and depth have been well documented in published accounts in the computing education research literature. Although there are a number of introductory textbooks devoted to HTML and CSS, and, of course, an incredibly large number of trade books focused on specific web technologies, many of these are largely unsuitable for computing major students. Rather than illustrating how to create simple pages using HTML and JavaScript with very basic server-side capabilities, we believed that instructors increasingly need a textbook that guides students through the development of realistic, enterprise-quality web applications using contemporary Internet development platforms and frameworks.

This book is intended to fill this need. It covers the required ACM web development topics in a modern manner that is closely aligned with contemporary best practices in the real world of web development. It is based on our experience teaching a variety of different web development courses since 1997, our working professionally in the web development industry, our research in published accounts in the computing education literature, and in our corresponding with colleagues across the world. We hope that you find that this book does indeed satisfy your requirements for a web development textbook!

Acknowledgments

A book of this scale and scope incurs many debts of gratitude. We are first and foremost exceptionally grateful to Matt Goldstein, the Acquisitions Editor at Pearson, who championed the book and guided the overall process of bringing the book to market. Joan Murray and Shannon Bailey from Pearson played crucial roles in getting the initial prospectus considered. Kayla Smith-Tarbox was the Program Manager and ably handled the very tricky job of coordinating between the writers and the production team. Scott Disanno and Jenah Blitz-Soehr at Pearson also contributed in the early stages. We would like to thank Hardik Popli and his team at Cenveo Publisher Services for the work they did on the post-production side. We would also like to thank Margaret Berson, proofreader, who made sure that the words and illustrations actually work to tell a story that makes sense.

Reviewers help ensure that a textbook reflects more than just the authors' perspective. We were truly blessed in having two extraordinary reviewers: Jordan Pratt of Mount Royal University and Jamel Schiller of University of Wisconsin, Green Bay, who carefully examined every single chapter.

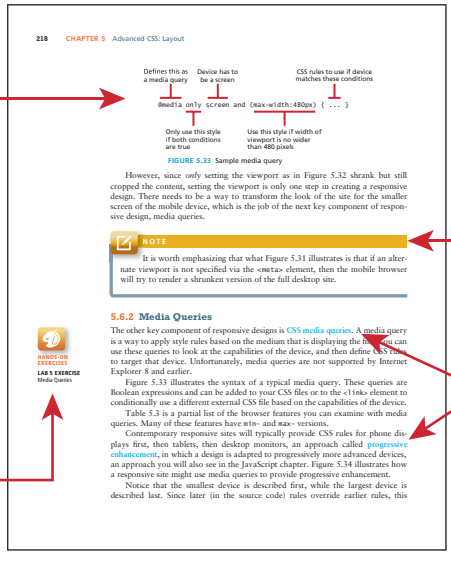
There are many others who helped guide our thinking, provided suggestions, or made our administrative and teaching duties somewhat less onerous. While we cannot thank everyone, we are grateful to Mount Royal University for granting a semester break for one of the authors, Peter Alston (now at the University of Liverpool) and his colleagues at Edge Hill University for hosting one of the authors for an important week early in the book's composition, and Amber Settle of De Paul University, who provided invaluable feedback on an early paper in which the rationale for the textbook was first hatched. Our long-time colleagues Paul Pospisil and Charles Hepler provided very helpful diversions from web development, which were always appreciated. And of course we would like to acknowledge all our students who have improved our insight and who acted as non-voluntary guinea pigs in the evolution of our thinking on teaching web development.

From its earliest inception in May of 2012 all the way to its conclusion in the early months of 2014, Dr. Janet Miller provided incredible and overwhelming encouragement, understanding, and feedback for which Randy Connolly will be always grateful. Joanne Hoar, an M.Sc. in computer science, made this book possible for Ricardo Hoar with continuous emotional support and professional feedback, all while maintaining a stable household for their three children under the age of 4 (and looking beautiful the whole time). Finally, we want to thank our children, Alexander Connolly, Benjamin Connolly, Archimedes Hoar, Curia Hoar, and Hypatia Hoar, who saw less of their fathers during this time but were always on our minds.

Pearson would like to thank Soumen Mukherjee and Arup Kumar Bhattacharjee of RCC Institute of Information Technology, Kolkata, and Manasa S. of NMAM Institute of Technology, Bangalore for reviewing the Global Edition.

Visual Walkthrough

Hundreds of color-coded illustrations clarify key concepts.



218 CHAPTER 5 Advanced CSS: Layout

Defines this as a media query | Device has to be a screen | CSS rules to use if device matches these conditions

```
@media only screen and (max-width:480px) { ... }
```

Only use this style if both conditions are true | Use this style if width of viewport is no wider than 480 pixels

FIGURE 5.33 Sample media query

However, since *only* setting the viewport as in Figure 5.32 shrank but still cropped the content, setting the viewport is only one step in creating a responsive design. There needs to be a way to transform the look of the site for the smaller screen of the mobile device, which is the job of the next key component of responsive design, media queries.

NOTE

It is worth emphasizing that what Figure 5.31 illustrates is that if an alternate viewport is not specified via the <meta> element, then the mobile browser will try to render a shrunken version of the full desktop site.

5.6.2 Media Queries

The other key component of responsive designs is CSS media queries. A media query is a way to apply style rules based on the medium that is displaying the page. You can use these queries to look at the capabilities of the device, and then define CSS rules to target that device. Unfortunately, media queries are not supported by Internet Explorer 8 and earlier.

Figure 5.33 illustrates the syntax of a typical media query. These queries are Boolean expressions and can be added to your CSS files or to the <link> element to conditionally use a different external CSS file based on the capabilities of the device.

Table 5.3 is a partial list of the browser features you can examine with media queries. Many of these features have `min-` and `max-` versions.

Contemporary responsive sites will typically provide CSS rules for phone displays first, then tablets, then desktop monitors, an approach called progressive enhancement, in which a design is adapted to progressively more advanced devices, an approach you will also see in the JavaScript chapter. Figure 5.34 illustrates how a responsive site might use media queries to provide progressive enhancement.

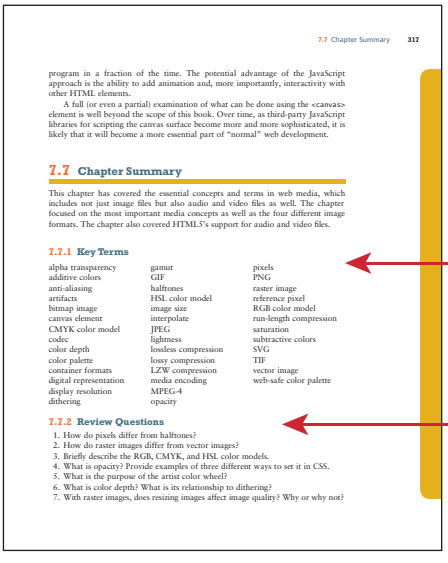
Notice that the smallest device is described first, while the largest device is described last. Since later (in the source code) rules override earlier rules, this

HANDS ON EXERCISES
LAB 5 EXERCISE
Media Queries

Security, Pro Tip, and Note boxes emphasize important concepts and practical advice.

Key terms are highlighted in consistent color.

Separate hands-on exercises (available online) give readers opportunity to practically apply concepts and techniques covered in the text.



7.7 Chapter Summary 317

program in a fraction of the time. The potential advantage of the JavaScript approach is the ability to add animation and, more importantly, interactivity with other HTML elements.

A full (or even a partial) examination of what can be done using the <canvas> element is well beyond the scope of this book. Over time, as third-party JavaScript libraries for scripting the canvas surface become more and more sophisticated, it is likely that it will become a more essential part of "normal" web development.

7.7 Chapter Summary

This chapter has covered the essential concepts and terms in web media, which includes not just image files but also audio and video files as well. The chapter focused on the most important media concepts as well as the four different image formats. The chapter also covered HTML5's support for audio and video files.

7.7.1 Key Terms

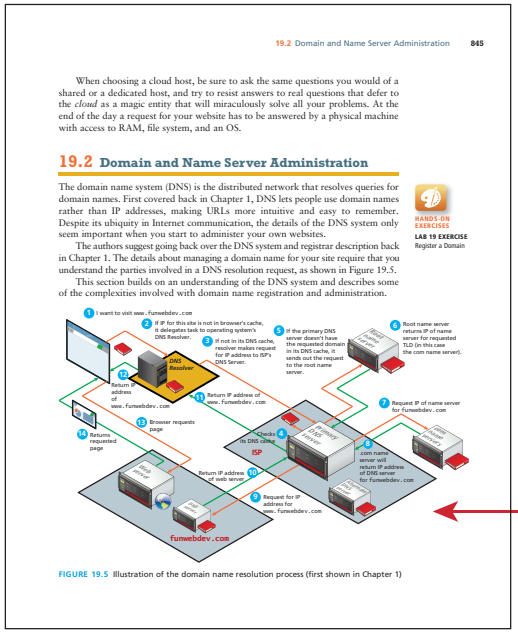
alpha transparency
additive colors
anti-aliasing
artifacts
bitmap image
canvas element
CMYK color model
codec
color depth
color palette
container formats
digital representation
display resolution
dithering
gamut
GIF
halftones
HSL color model
image size
interpolate
JPEG
lightness
lossless compression
lossy compression
LZW compression
media encoding
MPEG-4
opacity
pixels
PNG
raster image
reference pixel
RGB color model
run-length compression
saturation
subtractive colors
SVG
TIF
vector image
web-safe color palette

7.7.2 Review Questions

1. How do pixels differ from halftones?
2. How do raster images differ from vector images?
3. Briefly describe the RGB, CMYK, and HSL color models.
4. What is opacity? Provide examples of three different ways to set it in CSS.
5. What is the purpose of the artist color wheel?
6. What is color depth? What is its relationship to dithering?
7. With raster images, does resizing images affect image quality? Why or why not?

Key terms appear again at end of chapter.

Review questions at end of chapter provide opportunity for self-testing.



19.2 Domain and Name Server Administration 845

When choosing a cloud host, be sure to ask the same questions you would of a shared or a dedicated host, and try to resist answers to real questions that defer to the *cloud* as a magic entity that will miraculously solve all your problems. At the end of the day a request for your website has to be answered by a physical machine with access to RAM, file system, and an OS.

19.2 Domain and Name Server Administration

The domain name system (DNS) is the distributed network that resolves queries for domain names. First covered back in Chapter 1, DNS lets people use domain names rather than IP addresses, making URLs more intuitive and easy to remember. Despite its ubiquity in Internet communication, the details of the DNS system only seem important when you start to administer your own websites.

HANDS-ON EXERCISES
LAB 19 EXERCISE
Register a Domain

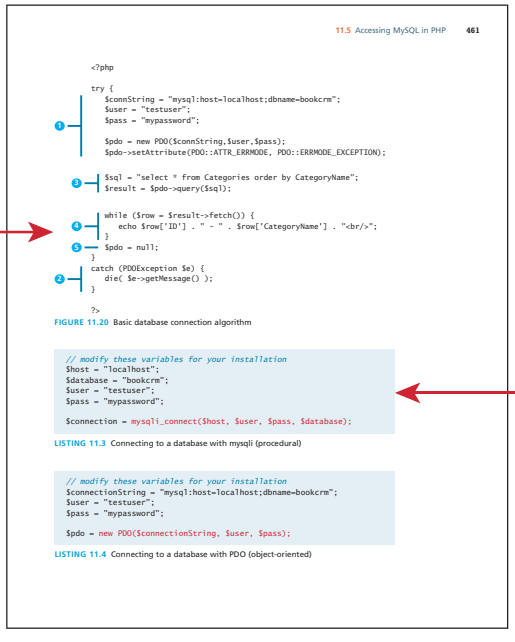
The authors suggest going back over the DNS system and registrar description back in Chapter 1. The details about managing a domain name for your site require that you understand the parties involved in a DNS resolution request, as shown in Figure 19.5.

This section builds on an understanding of the DNS system and describes some of the complexities involved with domain name registration and administration.

FIGURE 19.5 Illustration of the domain name resolution process (first shown in Chapter 1)

Illustrations help explain especially complicated processes.

Important algorithms are illustrated visually to help clarify understanding.



11.5 Accessing MySQL in PHP 461

```
<?php
try {
    $connString = "mysql:host=localhost;dbname=bookcrm";
    $user = "testuser";
    $pass = "mypassword";

    $pdo = new PDO($connString,$user,$pass);
    $pdo->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);

    $sql = "select * from Categories order by CategoryName";
    $result = $pdo->query($sql);

    while ($row = $result->fetch()) {
        echo $row['ID'] . " - " . $row['CategoryName'] . "<br/>";
    }
    $pdo = null;
}
catch (PDOException $e) {
    die( $e->getMessage() );
}
?>
```

FIGURE 11.20 Basic database connection algorithm

```
// modify these variables for your installation
$host = "localhost";
$database = "bookcrm";
$user = "testuser";
$pass = "mypassword";

$connection = mysqli_connect($host, $user, $pass, $database);
```

LISTING 11.3 Connecting to a database with mysqli (procedural)

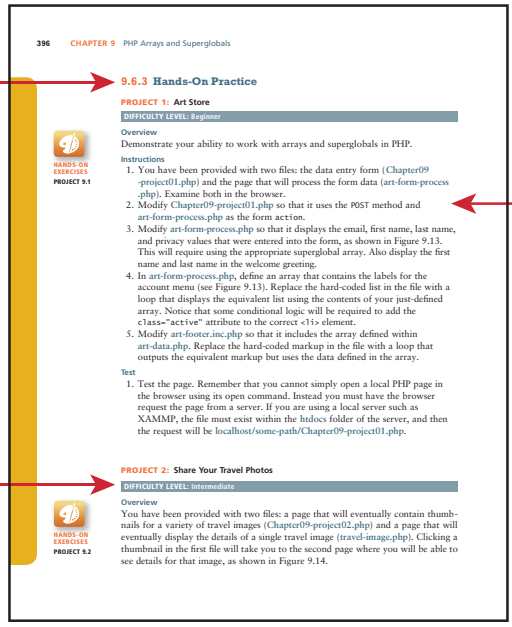
```
// modify these variables for your installation
$connectionString = "mysql:host=localhost;dbname=bookcrm";
$user = "testuser";
$pass = "mypassword";

$pdo = new PDO($connectionString, $user, $pass);
```

LISTING 11.4 Connecting to a database with PDO (object-oriented)

Color-coded source code listings emphasize important elements and visually separate comments from the code.

Each chapter ends with three case study exercises that allow the reader to practice the material covered in the chapter within a realistic context.



396 CHAPTER 9 PHP Arrays and Superglobals

9.6.3 Hands-On Practice

PROJECT 1: Art Store

DIFFICULTY LEVEL: Beginner

HANDS-ON EXERCISES PROJECT 9.1

Overview

Demonstrate your ability to work with arrays and superglobals in PHP.

Instructions

1. You have been provided with two files: the data entry form (Chapter09-project01.php) and the page that will process the form data (art-form-process.php). Examine both in the browser.
2. Modify Chapter09-project01.php so that it uses the `POST` method and art-form-process.php as the form `action`.
3. Modify art-form-process.php so that it displays the email, first name, last name, and privacy values that were entered into the form, as shown in Figure 9.13. This will require using the appropriate superglobal array. Also display the first name and last name in the welcome greeting.
4. In art-form-process.php, define an array that contains the labels for the account menu (see Figure 9.13). Replace the hard-coded list in the file with a loop that displays the equivalent list using the contents of your just-defined array. Notice that some conditional logic will be required to add the `class="active"` attribute to the correct `<li>` element.
5. Modify art-footer.inc.php so that it includes the array defined within art-data.php. Replace the hard-coded markup in the file with a loop that outputs the equivalent markup but uses the data defined in the array.

Test

1. Test the page. Remember that you cannot simply open a local PHP page in the browser using its open command. Instead you must have the browser request the page from a server. If you are using a local server such as XAMMP, the file must exist within the htdocs folder of the server, and then the request will be localhost/some-path/Chapter09-project01.php.

PROJECT 2: Share Your Travel Photos

DIFFICULTY LEVEL: Intermediate

HANDS-ON EXERCISES PROJECT 9.2

Overview

You have been provided with two files: a page that will eventually contain thumbnails for a variety of travel images (Chapter09-project02.php) and a page that will eventually display the details of a single travel image (travel-image.php). Clicking a thumbnail in the first file will take you to the second page where you will be able to see details for that image, as shown in Figure 9.14.

Exercises contain step-by-step instructions of varying difficulty.

Exercises increase in complexity and can be assigned separately by the instructor.

Attractive and realistic case studies help engage the readers' interest.

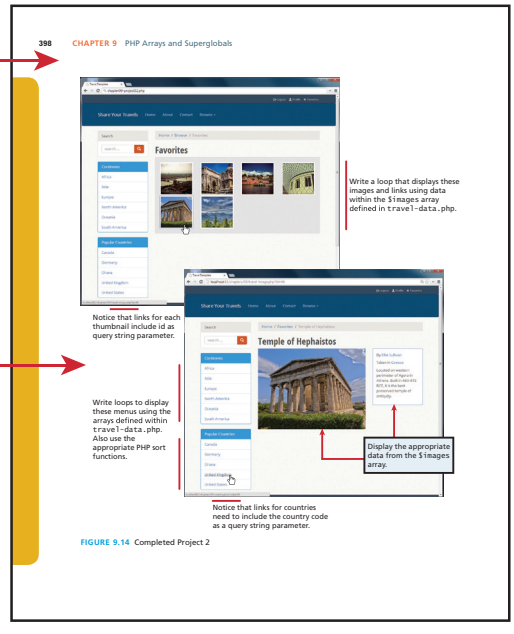


FIGURE 9.14 Completed Project 2

All images, pages, classes, databases, and other material for each of the case studies are available for download.

How the Web Works

1

CHAPTER OBJECTIVES

In this chapter you will learn . . .

- The history of the Internet and World Wide Web
- Fundamental concepts and protocols that support the Internet
- About the hardware and software that supports the Internet
- How a web page is actually retrieved and interpreted

This chapter introduces the World Wide Web (WWW). The WWW relies on a number of systems, protocols, and technologies all working together in unison. Before learning about HTML markup, CSS styling, JavaScript, and PHP programming, you must understand how the Internet makes web applications possible. This chapter begins with a brief history of the Internet and provides an overview of key Internet and WWW technologies applicable to the web developer. To truly understand these concepts in depth, one would normally take courses in computer science or information technology (IT) covering networking principles. If you find some of these topics too in-depth or advanced, you may decide to skip over some of the details here and return to them later.

1.1 Definitions and History

The World Wide Web (WWW or simply the Web) is certainly what most people think of when they see the word "Internet." But the WWW is only a subset of the Internet, as illustrated in Figure 1.1.

1.1.1 A Short History of the Internet

The history of telecommunication and data transport is a long one. There is a strategic advantage in being able to send a message as quickly as possible (or at least, more quickly than your competition). The Internet is not alone in providing instantaneous digital communication. Earlier technologies like radio, telegraph, and the telephone provided the same speed of communication, albeit in an analog form.

Telephone networks in particular provide a good starting place to learn about modern digital communications. In the telephone networks of old, calls were routed through operators who physically connected caller and receiver by connecting a wire to a switchboard to complete a circuit. These operators were around in some areas for almost a century before being replaced with automatic mechanical switches, which did the same job: physically connect caller and receiver.

One of the weaknesses of having a physical connection is that you must establish a link and maintain a dedicated circuit for the duration of the call. This type of network connection is sometimes referred to as **circuit switching** and is shown in Figure 1.2.

The problem with circuit switching is that it can be difficult to have multiple conversations simultaneously (which a computer might want to do). It also requires more **bandwidth** since even the silences are transmitted (that is, unused capacity in the network is not being used efficiently).

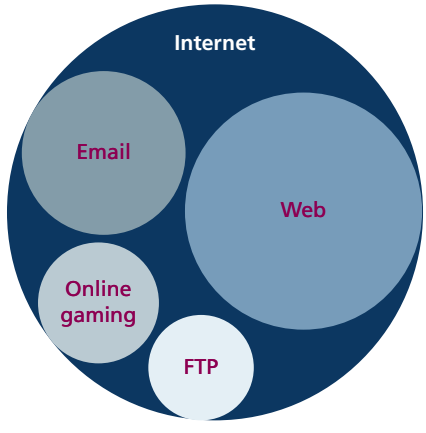


FIGURE 1.1 The web as a subset of the Internet

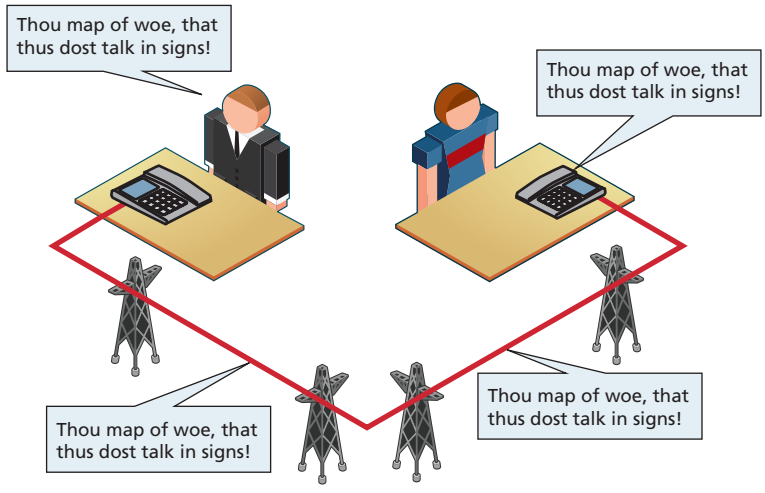


FIGURE 1.2 Telephone network as example of circuit switching

Bandwidth is a measurement of how much data can (maximally) be transmitted along an Internet connection. Normally measured in bits per second (bps), this measurement differs according to the type of Internet access technology you are using. A dial-up 56-Kbps modem has far less bandwidth than a 10-Gbps fiber optic connection.

In the 1960s, as researchers explored digital communications and began to construct the first networks, the research network ARPANET was created. ARPANET did not use circuit switching but instead used an alternative communications method called **packet switching**. A packet-switched network does not require a continuous connection. Instead it splits the messages into smaller chunks called **packets** and routes them to the appropriate place based on the destination address. The packets can take different routes to the destination, as shown in Figure 1.3. This may seem a more complicated and inefficient approach than circuit switching, but is in fact more robust (it is not reliant on a single pathway that may fail) and a more efficient use of network resources (since a circuit can communicate data from multiple connections).

This early ARPANET network was funded and controlled by the United States government, and was used exclusively for academic and scientific purposes. The early network started small with just a handful of connected university campuses and research institutions and companies in 1969 and grew to a few hundred by the early 1980s.

At the same time, alternative networks were created like X.25 in 1974, which allowed (and encouraged) business use. USENET, built in 1979, had fewer restrictions still, and as a result grew quickly to 550 hosts by 1981. Although there was growth in these various networks, the inability for them to communicate with each

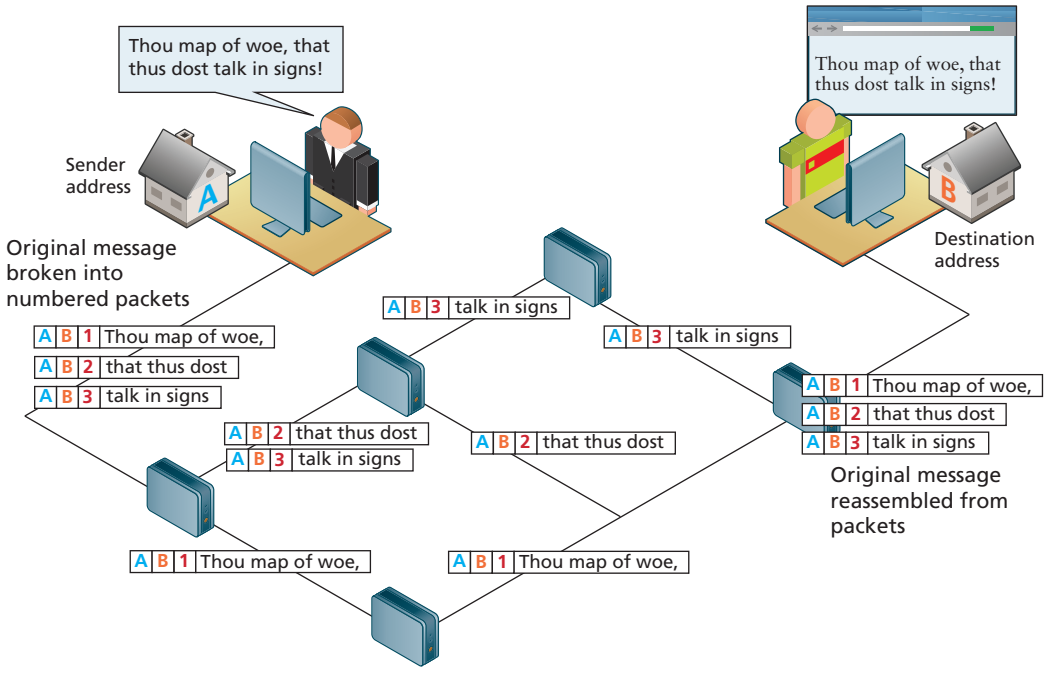


FIGURE 1.3 Internet network as example of packet switching

other was a real limitation. To promote the growth and unification of the disparate networks, a suite of **protocols** was invented to unify the networks. A protocol is the name given to a formal set of publicly available rules that manage data exchange between two points. Communications protocols allow any two computers to talk to one another, so long as they implement the protocol.

By 1981 protocols for the Internet were published and ready for use.[1,2] New networks built in the United States began to adopt the **TCP/IP (Transmission Control Protocol/Internet Protocol)** communication model (discussed in the next section), while older networks were transitioned over to it.

Any organization, private or public, could potentially connect to this new network so long as they adopted the TCP/IP protocol. On January 1, 1983, TCP/IP was adopted across all of ARPANET, marking the end of the research network that spawned the Internet.[3] Over the next two decades, TCP/IP networking was adopted across the globe.

1.1.2 The Birth of the Web

The next decade saw an explosion in the numbers of users, but the Internet of the late 1980s and the very early 1990s did not resemble the Internet we know today. During these early years, email and text-based systems were the extent of the Internet experience.

This transition from the old terminal and text-only Internet of the 1980s to the Internet of today is of course due to the invention and massive growth of the World Wide Web. This invention is usually attributed to the British Tim Berners-Lee (now Sir Tim Berners-Lee), who, along with the Belgian Robert Cailliau, published a proposal in 1990 for a hypertext system while both were working at CERN in Switzerland. Shortly thereafter Berners-Lee developed the main features of the web.[4]

This early web incorporated the following essential elements that are still the core features of the web today:

- A Uniform Resource Locator (URL) to uniquely identify a resource on the WWW.
- The Hypertext Transfer Protocol (HTTP) to describe how requests and responses operate.
- A software program (later called web server software) that can respond to HTTP requests.
- Hypertext Markup Language (HTML) to publish documents.
- A program (later called a browser) that can make HTTP requests from URLs and that can display the HTML it receives.

HTML will require several chapters to cover in this book. URLs and the HTTP are covered in this chapter. This chapter will also provide a little bit of insight into the nature of web server software; Chapter 20 will examine the inner workings of server software in more detail.

So while the essential outline of today's web was in place in the early 1990s, the web as we know it did not really begin until **Mosaic**, the first popular graphical browser application, was developed at the National Center for Supercomputing Applications at the University of Illinois Urbana-Champaign and released in early 1993 by Eric Bina and Marc Andreessen (who was a computer science undergraduate student at the time). Andreessen later moved to California and cofounded Netscape Communications, which released **Netscape Navigator** in late 1994. Navigator quickly became the principal web browser, a position it held until the end of the 1990s, when Microsoft's Internet Explorer (first released in 1995) became the market leader, a position it would hold for over a decade.

Also in late 1994, Berners-Lee helped found the **World Wide Web Consortium (W3C)**, which would soon become the international standards organization that would oversee the growth of the web. This growth was very much facilitated by the decision of CERN to not patent the work and ideas done by its employee and instead leave the web protocols and code-base royalty free.

To illustrate the growth of the Internet, Figure 1.4 graphs the count of hosts connected to the Internet from 1990 until 2010. You can see that the last decade in particular has seen an enormous growth, during which social networks, web

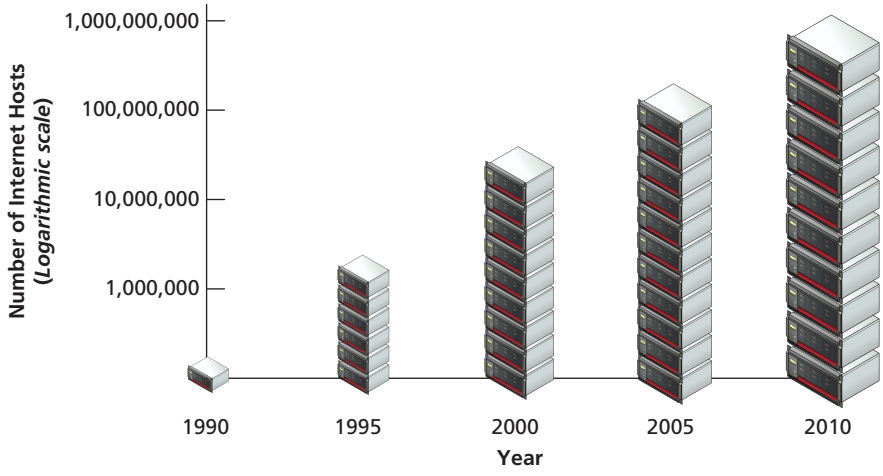


FIGURE 1.4 Growth in Internet hosts/servers based on data from the Internet Systems Consortium.[5]

services, asynchronous applications, the semantic web, and more have all been created (and will be described fully in due course in this textbook).

BACKGROUND

The Request for Comments (RFC) archive lists all of the Internet and WWW protocols, concepts, and standards. It started out as an unofficial repository for ARPANET information and eventually became the de facto official record. Even today new standards are published there.

1.1.3. Web Applications in Comparison to Desktop Applications

The user experience for a website is unlike the user experience for traditional desktop software. The location of data storage, limitations with the user interface, and limited access to operating system features are just some of the distinctions. However, as web applications have become more and more sophisticated, the differences in the user experience between desktop applications and web applications are becoming more and more blurred.

There are a variety of advantages and disadvantages to web-based applications in comparison to desktop applications. Some of the advantages of web applications include:

- Accessible from any Internet-enabled computer.
- Usable with different operating systems and browser applications.

- Easier to roll out program updates since only software on the server needs to be updated and not on every desktop in the organization.
- Centralized storage on the server means fewer security concerns about local storage (which is important for sensitive information such as health care data).

Unfortunately, in the world of IT, for every advantage, there is often a corresponding disadvantage; this is also true of web applications. Some of these disadvantages include:

- Requirement to have an active Internet connection (the Internet is not always available everywhere at all times).
- Security concerns about sensitive private data being transmitted over the Internet.
- Concerns over the storage, licensing, and use of uploaded data.
- Problems with certain websites on certain browsers not looking quite right.
- Restrictions on access to the operating system can prevent software and hardware from being installed or accessed (like Adobe Flash on iOS).

In addition, clients or their IT staff may have additional plugins added to their browsers, which provide added control over their browsing experience, but which might interfere with JavaScript, cookies, or advertisements. We will continually try to address these challenges throughout the book.

BACKGROUND

One of the more common terms you might encounter in web development is the term "**intranet**" (with an "**a**"), which refers to an Internet network that is local to an organization or business. Intranet resources are often private, meaning that only employees (or authorized external parties such as customers or suppliers) have access to those resources. Thus Internet (with an "**e**") is a broader term that encompasses both private (intranet) and public networked resources.

Intranets are typically protected from unauthorized external access via security features such as firewalls or private IP ranges, as shown in Figure 1.5. Because intranets are private, search engines such as Google have limited or no access to content within them.

Due to this private nature, it is difficult to accurately gauge, for instance, how many web pages exist within intranets, and what technologies are more common in them. Some especially expansive estimates guess that almost half of all web resources are hidden in private intranets.

Being aware of intranets is also important when one considers the job market and market usage of different web technologies. If one focuses just on the

(*continued*)

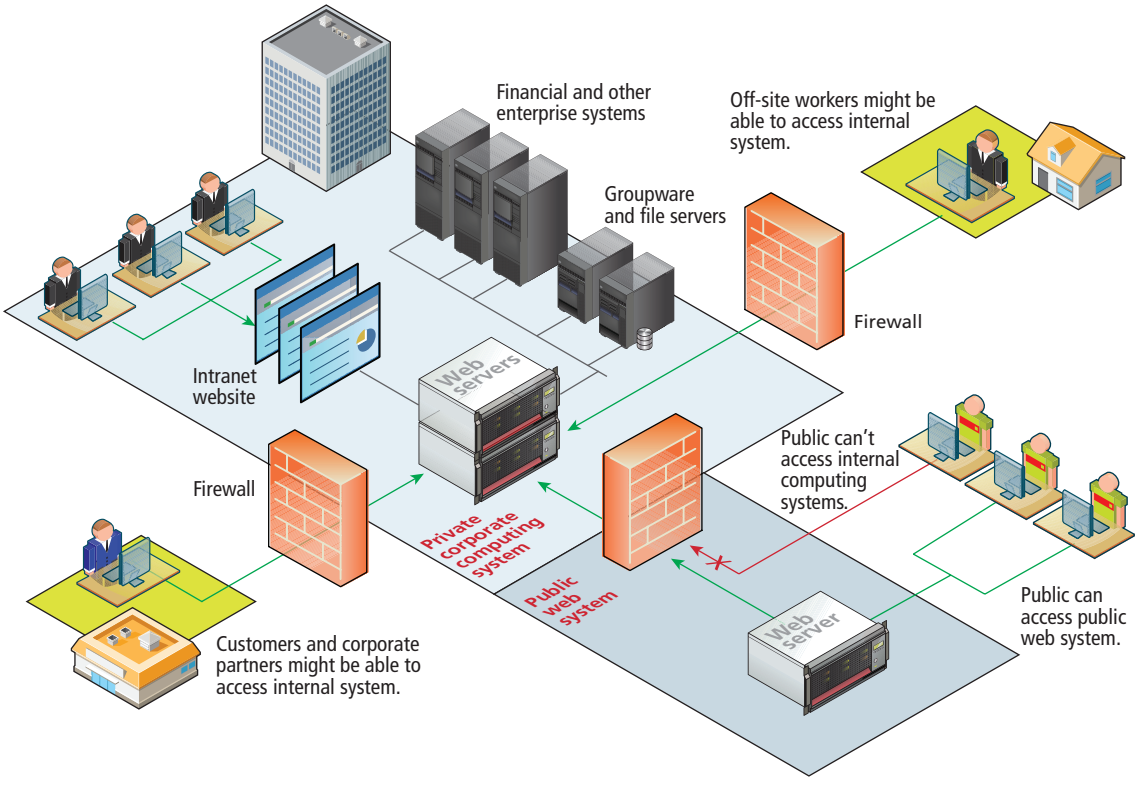


FIGURE 1.5 Intranet versus Internet

public Internet, it will appear that PHP, MySQL, and WordPress are the most commonly used web development stack. But when one adds in the private world of corporate intranets, other technologies such as ASP.NET, JSP, SharePoint, Oracle, SAP, and IBM WebSphere are just as important.

1.1.4 Static Websites versus Dynamic Websites

In the earliest days of the web, a **webmaster** (the term popular in the 1990s for the person who was responsible for creating and supporting a website) would publish web pages and periodically update them. Users could read the pages but could not provide feedback. The early days of the web included many encyclopedic, collection-style sites with lots of content to read (and animated icons to watch).

In those early days, the skills needed to create a website were pretty basic: one needed knowledge of the HTML and perhaps familiarity with editing and creating images. This type of website is commonly referred to as a **static website**, in that it consists

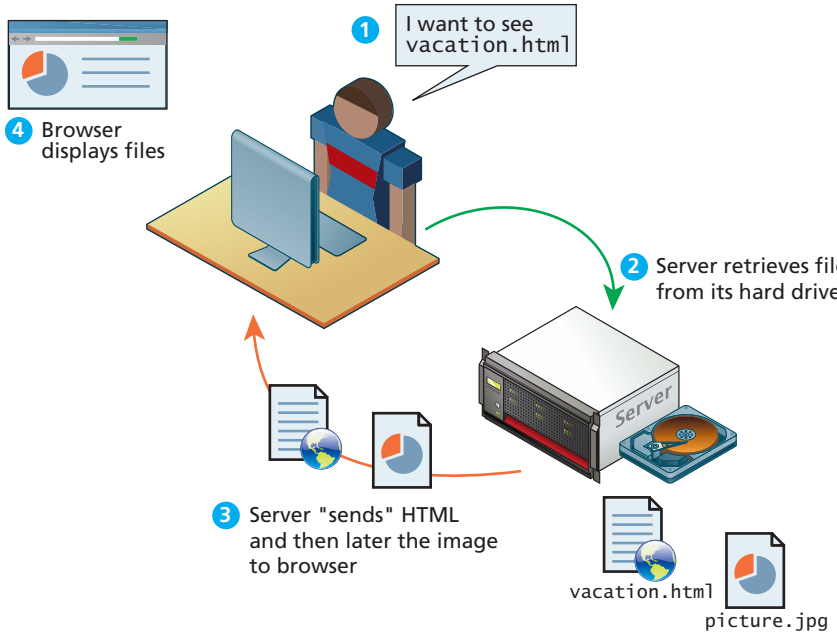


FIGURE 1.6 Static website

only of HTML pages that look identical for all users at all times. Figure 1.6 illustrates a simplified representation of the interaction between a user and a static website.

Within a few years of the invention of the web, sites began to get more complicated as more and more sites began to use programs running on web servers to generate content dynamically. These server-based programs would read content from databases, interface with existing enterprise computer systems, communicate with financial institutions, and then output HTML that would be sent back to the users' browsers. This type of website is called here in this text a **dynamic website** because the page content is being created at run time by a program created by a programmer; this page content can vary from user to user. Figure 1.7 illustrates a very simplified representation of the interaction between a user and a dynamic website.

So while knowledge of HTML was still necessary for the creation of these dynamic websites, it became necessary to have programming knowledge as well. And by the late 1990s, other knowledge and skills were becoming necessary, such as CSS, usability, and security.

1.1.5 Web 2.0 and Beyond

In the mid-2000s, a new buzzword entered the computer lexicon: Web 2.0. This term had two meanings, one for users and one for developers. For the users, Web 2.0

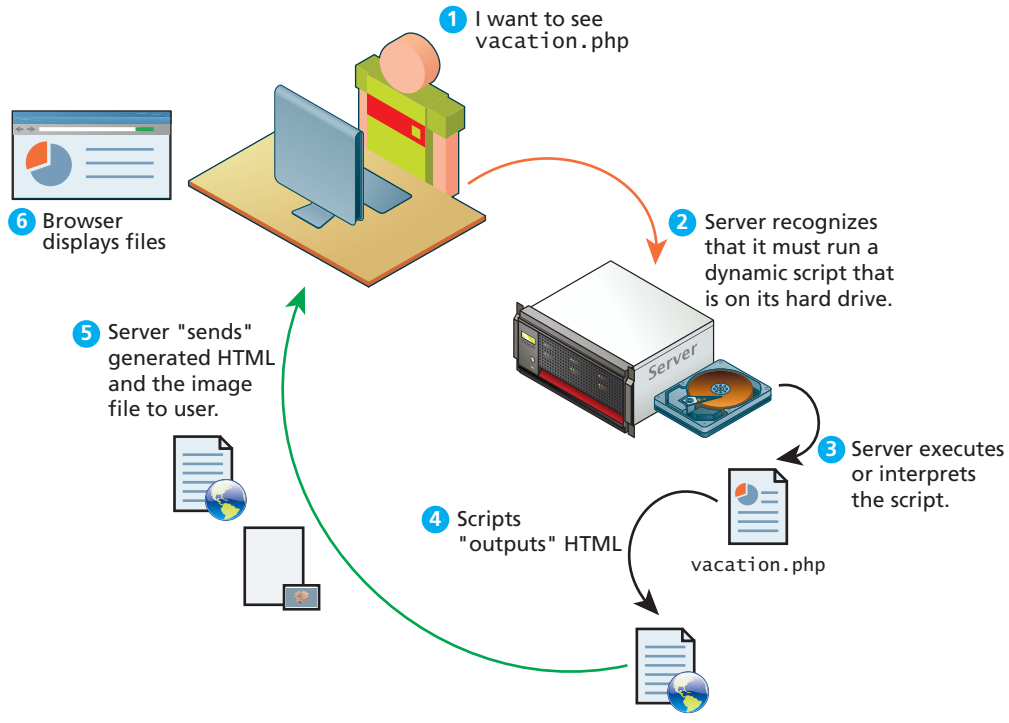


FIGURE 1.7 Dynamic website

referred to an interactive experience where users could contribute *and* consume web content, thus creating a more user-driven web experience. Some of the most popular websites fall into this category: Facebook, YouTube, and Wikipedia. This shift to allow feedback from the user, such as comments on a story, threads in a message board, or a profile on a social networking site has revolutionized what it means to use a web application.

For software developers, Web 2.0 also referred to a change in the paradigm of how dynamic websites are created. Programming logic, which previously existed only on the server, began to migrate to the browser. This required learning JavaScript, a rather tricky programming language that runs in the browser, as well as mastering the rather difficult programming techniques involved in asynchronous communication.

Web development in the Web 2.0 world is significantly more complicated today than it was even a decade ago. While this book attempts to cover all the main topics in web development, in practice, it is common for a certain division of labor to exist. The skills to create a good-looking static web page are not the same skill set that is required to write software that facilitates user interactions. Many programmers are

poor visual user interface designers, and most designers can't program. This separation of software system and visual user interface is essential to any Web 2.0 application.

Chapters on HTML and CSS are essential for learning about layout and design best practices. Later chapters on server and client-side programming build on those design skills, but go far beyond them. To build modern applications you must have both sets of skills on your team.

BACKGROUND

When a system is known by a 1.0 and 2.0, people invariably speculate on what the 3.0 version will look like. If there is a Web 3.0, it is currently uncertain and still under construction. Some people have, however, argued that Web 3.0 will be something called the **semantic web**.

Semantic is a word from linguistics that means, quite literally, "meaning." The semantic web thus adds context and meaning to web pages in the form of special markup. These semantic elements would allow search engines and other data mining agents to make sense of the content.

Currently a block of text on the web could be anything: a poem, an article, or a copyright notice. Search engines at present mainly just match the text you are searching for with text in the page. Currently these search engines have to use sophisticated algorithms to try to figure out the meaning of the page. The goal of the semantic web is to make it easier to figure out those meanings, thereby dramatically improving the nature of search on the web. Currently there are a number of semi-standardized approaches for adding semantic qualifiers to HTML; some examples include RDF (Resource Description Framework), OWL (Web Ontology Language), and SKOS (Simple Knowledge Organization System).

1.2 Internet Protocols

The Internet exists today because of a suite of interrelated communications protocols. A **protocol** is a set of rules that partners in communication use when they communicate. We have already mentioned one of these essential Internet protocols, namely TCP/IP.

These protocols have been implemented in every operating system, and make fast web development possible. If web developers had to keep track of packet routing, transmission details, domain resolution, checksums, and more, it would be hard to get around to the matter of actually building websites. Despite the fact that these protocols work behind the scenes for web developers, having some general awareness of what the suite of Internet protocols does for us can at times be helpful.

1.2.1 A Layered Architecture

The TCP/IP Internet protocols were originally abstracted as a four-layer stack.[6,7] Later abstractions subdivide it further into five or seven layers.[8] Since we are focused on the top layer anyhow, we will use the earliest and simplest four-layer network model shown in Figure 1.8.

Layers communicate information up or down one level, but needn't worry about layers far above or below. Lower layers handle the more fundamental aspects of transmitting signals through networks, allowing the higher layers to think about how a client and server interact. The web requires all layers to operate, although in web development we will focus on the highest layer, the application layer.

1.2.2 Link Layer

The link layer is the lowest layer, responsible for both the physical transmission across media (wires, wireless) and establishing logical links. It handles issues like

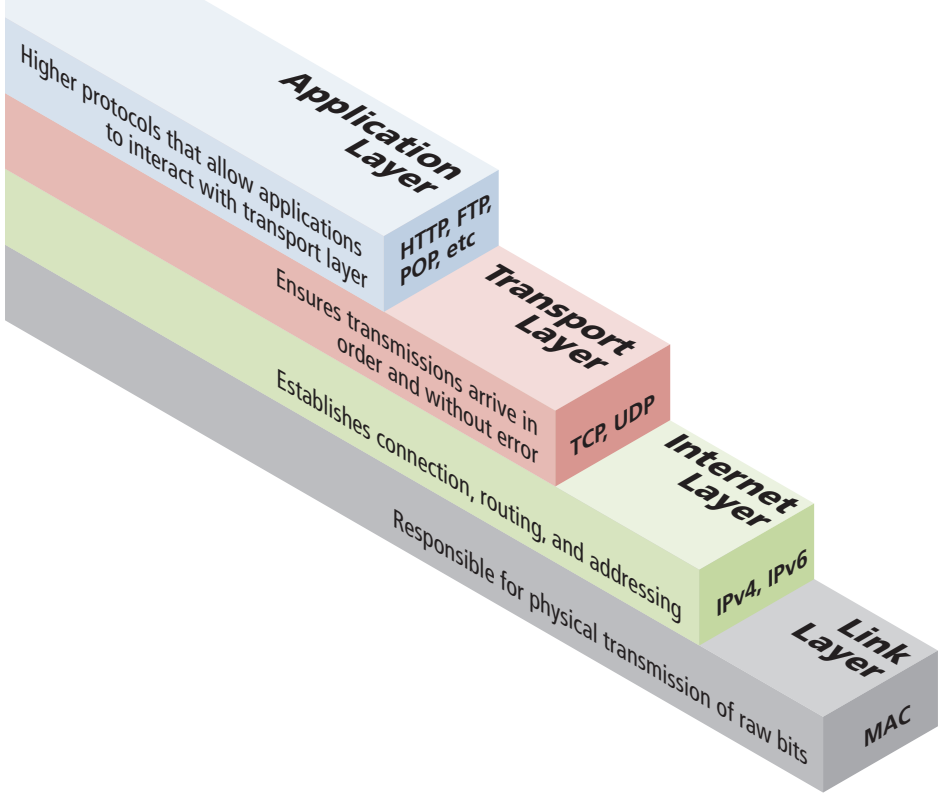


FIGURE 1.8 Four-layer network model

packet creation, transmission, reception, error detection, collisions, line sharing, and more. The one term here that is sometimes used in the Internet context is that of MAC (media access control) addresses. These are unique 48- or 64-bit identifiers assigned to network hardware and which are used at the physical networking level. We will not focus on this layer, although you can learn more in a computer networking course or text.

1.2.3 Internet Layer

The Internet layer (sometimes also called the IP Layer) routes packets between communication partners across networks. The Internet layer provides "best effort" communication. It sends out the message to the destination, but expects no reply, and provides no guarantee the message will arrive intact, or at all.

The Internet uses the Internet Protocol (IP) addresses to identify destinations on the Internet. As can be seen in Figure 1.9, every device connected to the Internet has an IP address, which is a numeric code that is meant to uniquely identify it.

The details of the IP addresses can be important to a web developer. There are occasions when one needs to track, record, and compare the IP address of a given web request. Online polls, for instance, need to compare IP addresses to ensure the same address does not vote more than once.

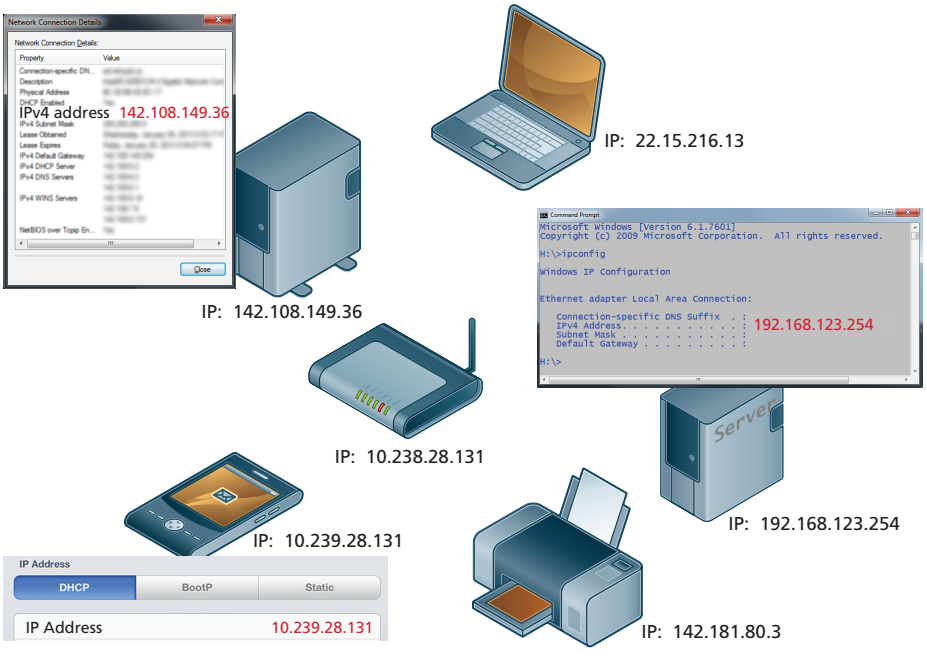


FIGURE 1.9 IP addresses and the Internet

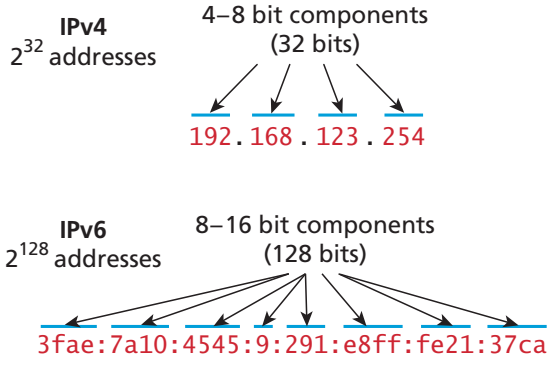


FIGURE 1.10 IPv4 and IPv6 comparison

There are two types of IP addresses: IPv4 and IPv6. **IPv4** addresses are the IP addresses from the original TCP/IP protocol. In IPv4, 12 numbers are used (implemented as four 8-bit integers), written with a dot between each integer (Figure 1.10). Since an unsigned 8-bit integer's maximum value is 255, four integers together can encode approximately 4.2 billion unique IP addresses.

HANDS-ON EXERCISES

LAB 1 EXERCISE
Your IP address

Your IP address will generally be assigned to you by your Internet service provider (ISP). In organizations, large and small, purchasing extra IP addresses from the ISP is not cost effective. In a local network, computers can share a single external IP address between them. IP addresses in the range of **192.168.0.0** to **192.168.255**, for example, are reserved for exactly this local area network use. Your connection therefore might have an internal IP of **192.168.0.15** known only to the internal network, and another public IP address that is your address to the world.

The decision to make IP addresses 32 bits limited the number of hosts to 4.2 billion. As more and more devices connected to the Internet the supply was becoming exhausted, especially in some local areas that had already distributed their share.

To future-proof the Internet against the 4.2 billion limit, a new version of the IP protocol was created, **IPv6**. This newer version uses eight 16-bit integers for 2^{128}

BACKGROUND

You may be wondering who gives an ISP its IP addresses. The answer is ultimately the **Internet Assigned Numbers Authority (IANA)**. This group is actually a department of ICANN, the Internet Corporation for Assigned Names and Numbers, which is an internationally organized nonprofit organization responsible for the global coordination of IP addresses, domains, and Internet protocols. IANA allocates IP addresses from pools of unallocated addresses to Regional Internet Registries such as AfriNIC (for Africa) or ARIN (for North America).

unique addresses, over a billion *billion* times the number in IPv4. These 16-bit integers are normally written in hexadecimal, due to their longer length. This new addressing system is currently being rolled out with a number of transition mechanisms, making the rollout seamless to most users and even developers.

Figure 1.10 compares the IPv4 and IPv6 address schemes.

1.2.4 Transport Layer

The transport layer ensures transmissions arrive in order and without error. This is accomplished through a few mechanisms. First, the data is broken into packets formatted according to the Transmission Control Protocol (TCP). The data in these packets can vary in size from 0 to 64K, though in practice typical packet data size is around 0.5 to 1K. Each data packet has a header that includes a sequence number, so the receiver can put the original message back in order, no matter when they arrive. Secondly, each packet is acknowledged back to the sender so in the event of a lost packet, the transmitter will realize a packet has been lost since no ACK arrived for that packet. That packet is retransmitted, and although out of order, is reordered at the destination, as shown in Figure 1.11. This means you have a guarantee that messages sent will arrive and in order. As a consequence, web developers don't have to worry about pages not getting to the users.

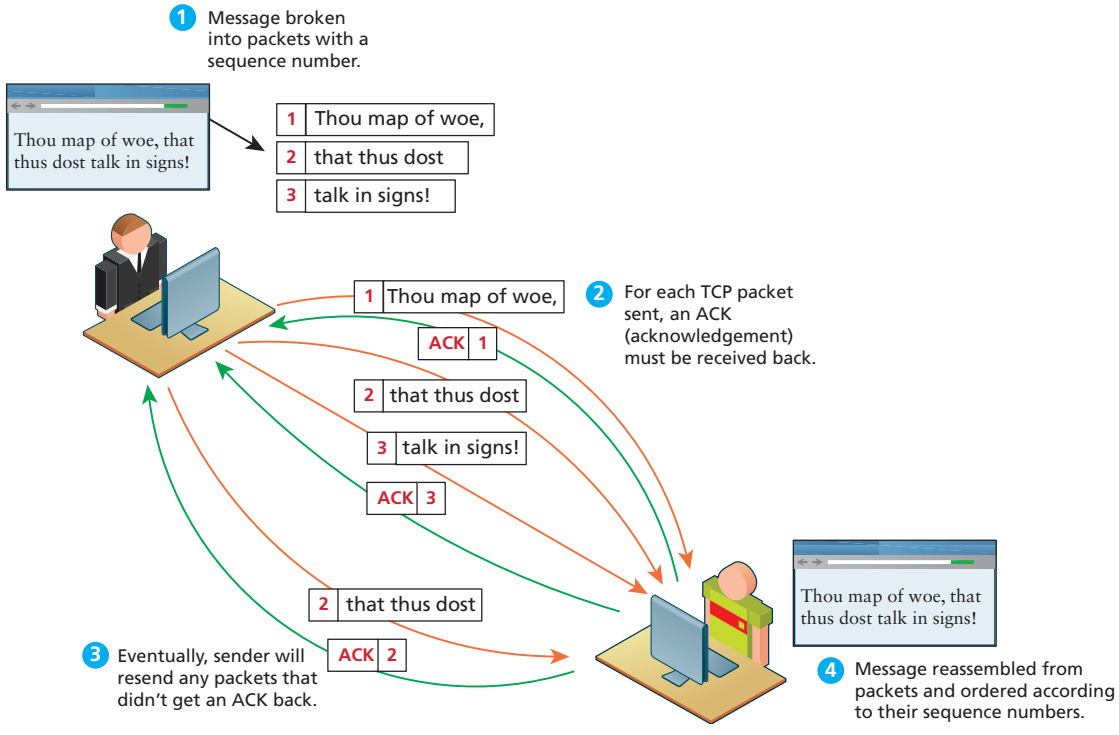


FIGURE 1.11 TCP packets

PRO TIP

Sometimes we do not want guaranteed transmission of packets.

Consider a live multicast of a soccer game, for example. Millions of subscribers may be streaming the game, and we can't afford to track and retransmit every lost packet. A small loss of data in the feed is acceptable, and the customers will still see the game. An Internet protocol called User Datagram Protocol (UDP) is used in these scenarios in lieu of TCP. Other examples of UDP services include Voice Over IP, many online games, and Domain Name System (DNS).

1.2.5 Application Layer

With the application layer, we are at the level of protocols familiar to most web developers. Application layer protocols implement process-to-process communication and are at a higher level of abstraction in comparison to the low-level packet and IP address protocols in the layers below it.

There are many application layer protocols. A few that are useful to web developers include:

- **HTTP.** The Hypertext Transfer Protocol is used for web communication.
- **SSH.** The Secure Shell Protocol allows remote command-line connections to servers.
- **FTP.** The File Transfer Protocol is used for transferring files between computers.
- **POP/IMAP/SMTP.** Email-related protocols for transferring and storing email.
- **DNS.** The Domain Name System protocol used for resolving domain names to IP addresses.

NOTE

We will discuss the HTTP and the DNS protocols later in this chapter. SSH will be covered later in the book in the chapter on security.

1.3 The Client-Server Model

The web is sometimes referred to as a client-server model of communications. In the client-server model, there are two types of actors: clients and servers. The server is a computer agent that is normally active 24 hours a day, 7 days a week, listening

for queries from any client who make a request. A **client** is a computer agent that makes requests and receives responses from the server, in the form of response codes, images, text files, and other data.

1.3.1 The Client

Client machines are the desktops, laptops, smart phones, and tablets you see everywhere in daily life. These machines have a broad range of specifications regarding operating system, processing speed, screen size, available memory, and storage. In the most familiar scenario, client requests for web pages come through a web browser. But a client can be more than just a web browser. When your word processor's help system accesses online resources, it is a client, as is an iOS game that communicates with a game server using HTTP. Sometimes a server web program can even act as a client. For instance, later in Chapter 17, our sample PHP websites will consume web services from service providers such as Flickr and Microsoft; in those cases, our PHP application will be acting as a client.

The essential characteristic of a client is that it can make requests to particular servers for particular resources using URLs and then wait for the response. These requests are processed in some way by the server.

1.3.2 The Server

The server in this model is the central repository, the command center, and the central hub of the client-server model. It hosts web applications, stores user and program data, and performs security authorization tasks. Since one server may serve many thousands, or millions of client requests, the demands on servers can be high. A site that stores image or video data, for example, will require many terabytes of storage to accommodate the demands of users.

The essential characteristic of a server is that it is listening for requests, and upon getting one, responds with a message. The exchange of information between the client and server is summarized by the request-response loop.

1.3.3 The Request-Response Loop

Within the client-server model, the **request-response loop** is the most basic mechanism on the server for receiving requests and transmitting data in response. The client initiates a **request** to a server and gets a **response** that could include some resource like an HTML file, an image, or some other data, as shown in Figure 1.12. This response can also contain other information about the request, or the resource provided such as response codes, cookies, and other data.

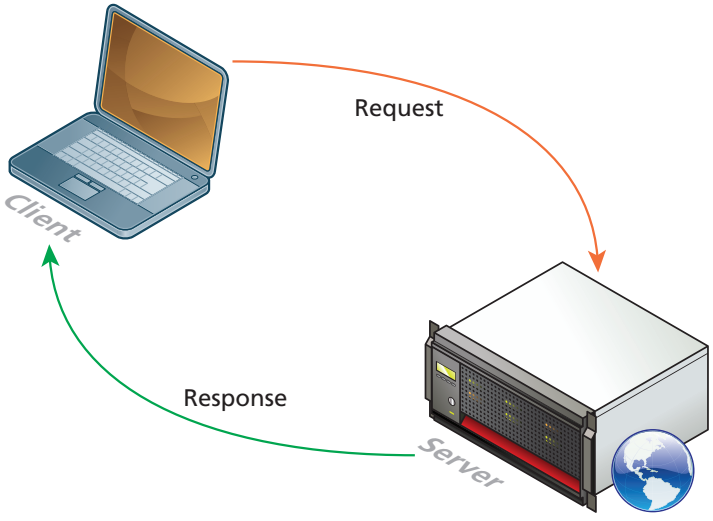


FIGURE 1.12 Request-response loop

1.3.4 The Peer-to-Peer Alternative

It may help your understanding to contrast the client-server model with a different network topology. In the peer-to-peer model, shown in Figure 1.13, where each computer is functionally identical, each node is able to send and receive data directly with one another. In such a model, each peer acts as both a client and server, able to upload and download information. Neither is required to be connected 24/7, and with each computer being functionally equal, there is less distinction between peers. The client-server model, in contrast, defines clear and distinct roles for the server. Video chat and bit torrent protocols are examples of the peer-to-peer model.

1.3.5 Server Types

In Figure 1.12, the server was shown as a single machine, which is fine from a conceptual standpoint. Clients make requests for resources from a URL; to the client, the server *is* a single machine.

However, most real-world websites are typically not served from a single server machine, but by many servers. It is common to split the functionality of a website between several different types of server, as shown in Figure 1.14. These include:

- Web servers. A web server is a computer servicing HTTP requests. This typically refers to a computer running web server software such as Apache or Microsoft IIS (Internet Information Services).

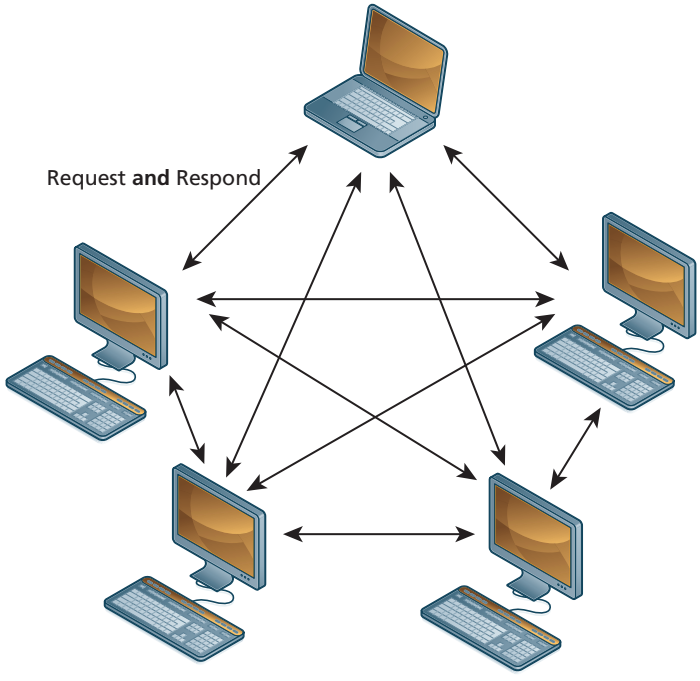


FIGURE 1.13 Peer-to-peer model

- **Application servers**. An application server is a computer that hosts and executes web applications, which may be created in PHP, ASP.NET, Ruby on Rails, or some other web development technology.
- **Database servers**. A database server is a computer that is devoted to running a Database Management System (DBMS), such as MySQL, Oracle, or SQL Server, that is being used by web applications.
- **Mail servers**. A mail server is a computer creating and satisfying mail requests, typically using the Simple Mail Transfer Protocol (SMTP).
- **Media servers**. A media server (also called a streaming server) is a special type of server dedicated to servicing requests for images and videos. It may run special software that allows video content to be streamed to clients.
- **Authentication servers**. An authentication server handles the most common security needs of web applications. This may involve interacting with local networking resources such as LDAP (Lightweight Directory Access Protocol) or Active Directory.

In smaller sites, these specialty servers are often the same machine as the web server.

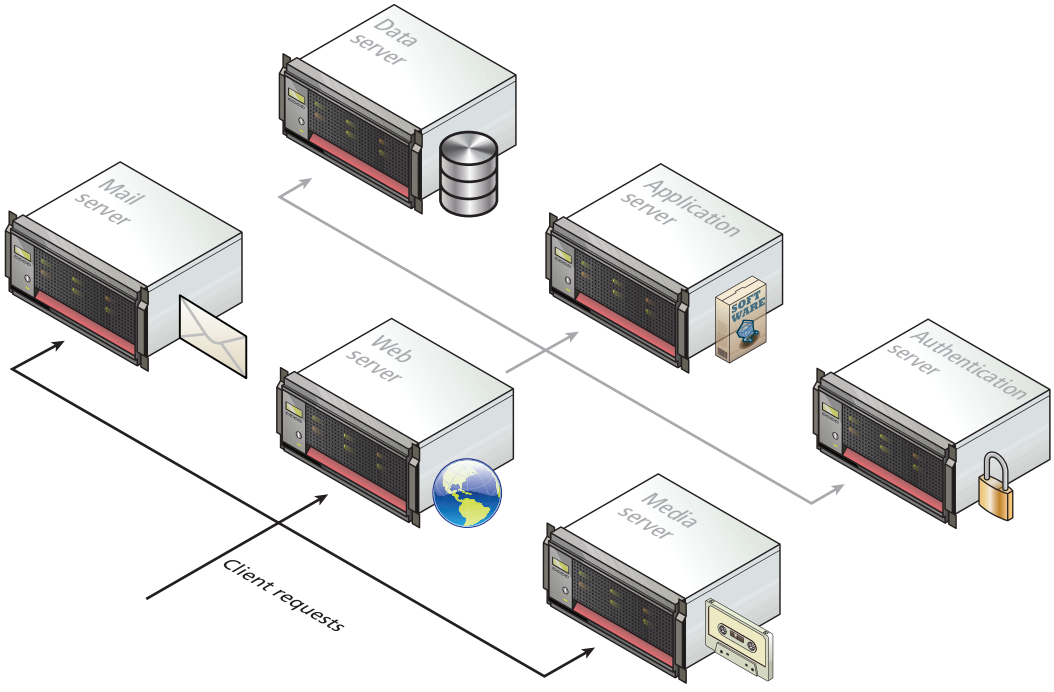


FIGURE 1.14 Different types of server

1.3.6 Real-World Server Installations

The previous section briefly described the different types of server that one might find in a real-world website. In such a site, not only are there different types of server, but there is often replication of each of the different server types. A busy site can receive thousands or even tens of thousands of requests a second; globally popular sites such as Facebook receive millions of requests a second.

A single web server that is also acting as an application or database server will be hard-pressed to handle more than a few hundred requests a second, so the usual strategy for busier sites is to use a **server farm**. The goal behind server farms is to distribute incoming requests between clusters of machines so that any given web or data server is not excessively overloaded, as shown in Figure 1.15. Special devices called **load balancers** distribute incoming requests to available machines.

Even if a site can handle its load via a single server, it is not uncommon to still use a server farm because it provides **failover redundancy**; that is, if the hardware fails in a single server, one of the replicated servers in the farm will maintain the site's availability.

In a server farm, the computers do not look like the ones in your house. Instead, these computers are more like the plates stacked in your kitchen cabinets. That is, a farm will have its servers and hard drives stacked on top of each other in **server**

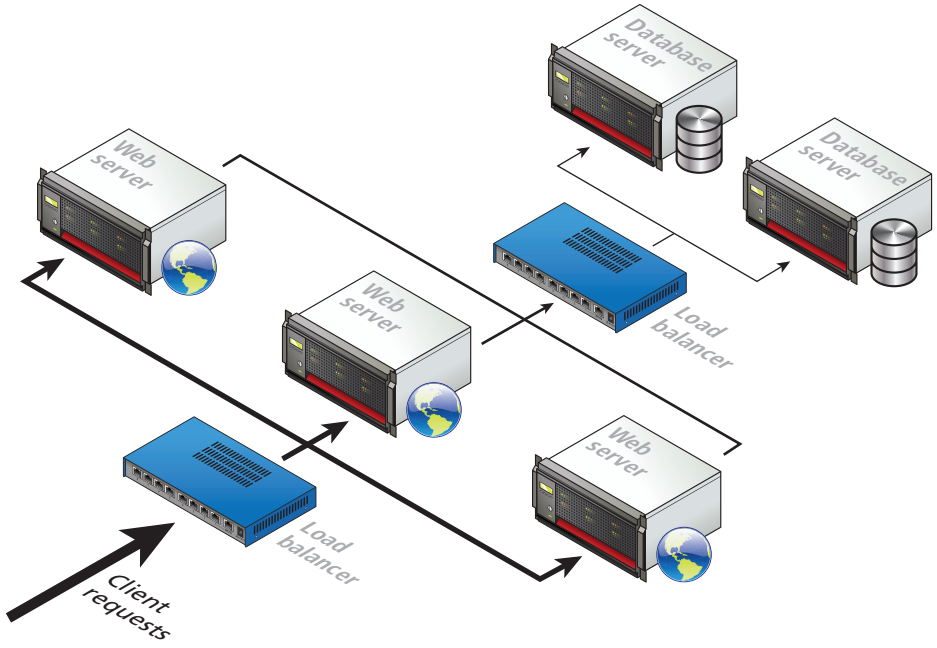


FIGURE 1.15 Server farm

racks. A typical server farm will consist of many server racks, each containing many servers, as shown in Figure 1.16.

Server farms are typically housed in special facilities called data centers. A **data center** will contain more than just computers and hard drives; sophisticated air conditioning systems, redundancy power systems using batteries and generators, and security personnel are all part of a typical data center, as shown in Figure 1.17.

To prevent the potential for site down times, most large websites will exist in mirrored data centers in different parts of the country, or even the world. As a consequence, the costs for multiple redundant data centers are quite high (not only due to the cost of the infrastructure but also due to the very large electrical power consumption used by data centers), and only larger web companies can afford to create and manage their own. Most web companies will instead lease space from a third-party data center.

The scale of the web farms and data centers for large websites can be astonishingly large. While most companies do not publicize the size of their computing infrastructure, some educated guesses can be made based on the publicly known IP address ranges and published records of a company's energy consumption and their power usage effectiveness.

For instance, a 2012 estimate argued that Amazon Web Services is using almost half a million servers spread across seven different data centers.[9] In 2012, an

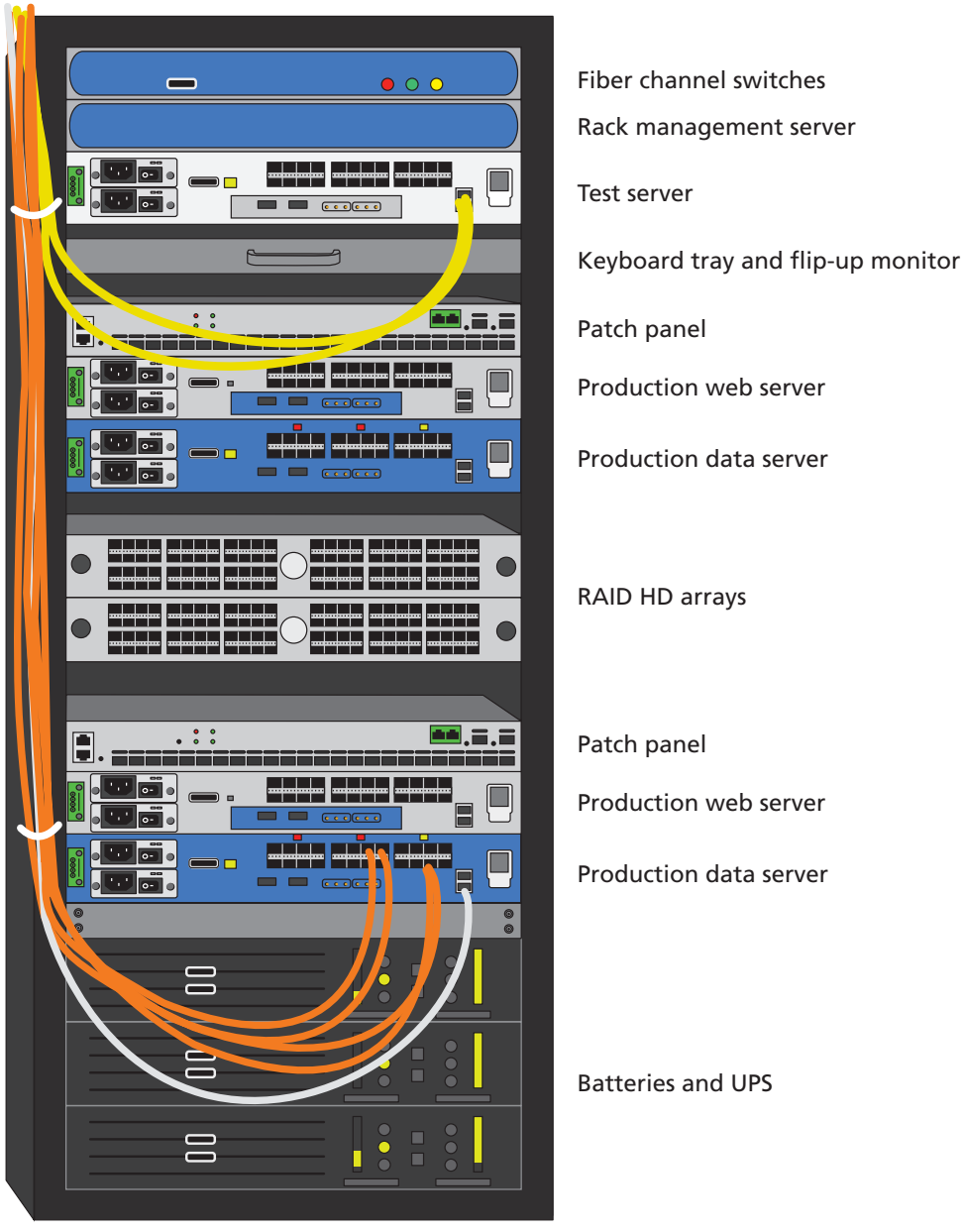


FIGURE 1.16 Sample server rack

infrastructure engineer at Amazon using a much more conservative estimation algorithm concluded that Facebook is using about 200,000 servers while Google is using around a million servers.[10]

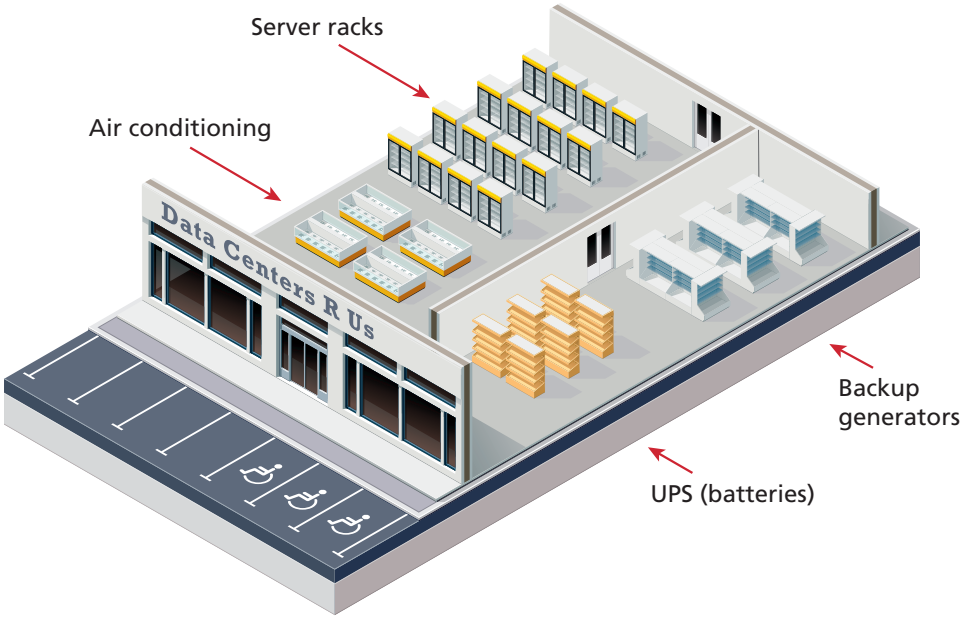


FIGURE 1.17 Hypothetical data center

BACKGROUND

It is also common for the reverse to be true—that is, a single server machine may host multiple sites. Large commercial web hosting companies such as GoDaddy, BlueHost, Dreamhost, and others will typically host hundreds or even thousands of sites on a single machine (or mirrored on several servers).

This type of server is sometimes referred to as a **virtual server** (or virtual private server). In this approach, each virtual server runs its own copy of the operating system web server software and thus emulates the operations of a dedicated physical server.

1.4 Where Is the Internet?

It is quite common for the Internet to be visually represented as a cloud, which is perhaps an apt way to think about the Internet given the importance of light and magnetic pulses to its operation. To many people using it, the Internet does seem to lack a concrete physical manifestation beyond our computer and cell phone screens.

But it is important to recognize that our global network of networks does not work using magical water vapor, but is implemented via millions of miles of copper wires and fiber optic cables, as well as via hundreds of thousands or even millions

HANDS-ON EXERCISES

LAB 1 EXERCISE
Tracing a Packet

of server computers and probably an equal number of routers, switches, and other networked devices, along with many thousands of air conditioning units and specially constructed server rooms and buildings.

The big picture of all the networking hardware involved in making the Internet work is far beyond the scope of this text. We should, however, try to provide at least some sense of the hardware that is involved in making the web possible.

1.4.1 From the Computer to the Local Provider

Andrew Blum, in his eye-opening book, *Tubes: A Journey to the Center of the Internet*, tells the reader that he decided to investigate the question "Where is the Internet" when a hungry squirrel gnawing on some outdoor cable wires disrupted his home connection thereby making him aware of the real-world texture of the Internet. While you may not have experienced a similar squirrel problem, for many of us, our main experience of the hardware component of the Internet is that which we experience in our homes. While there are many configuration possibilities, Figure 1.18 does provide an approximate simplification of a typical home to local provider setup.

The **broadband modem** (also called a cable modem or DSL modem) is a bridge between the network hardware outside the house (typically controlled by a phone or cable company) and the network hardware inside the house. These devices are often supplied by the ISP.

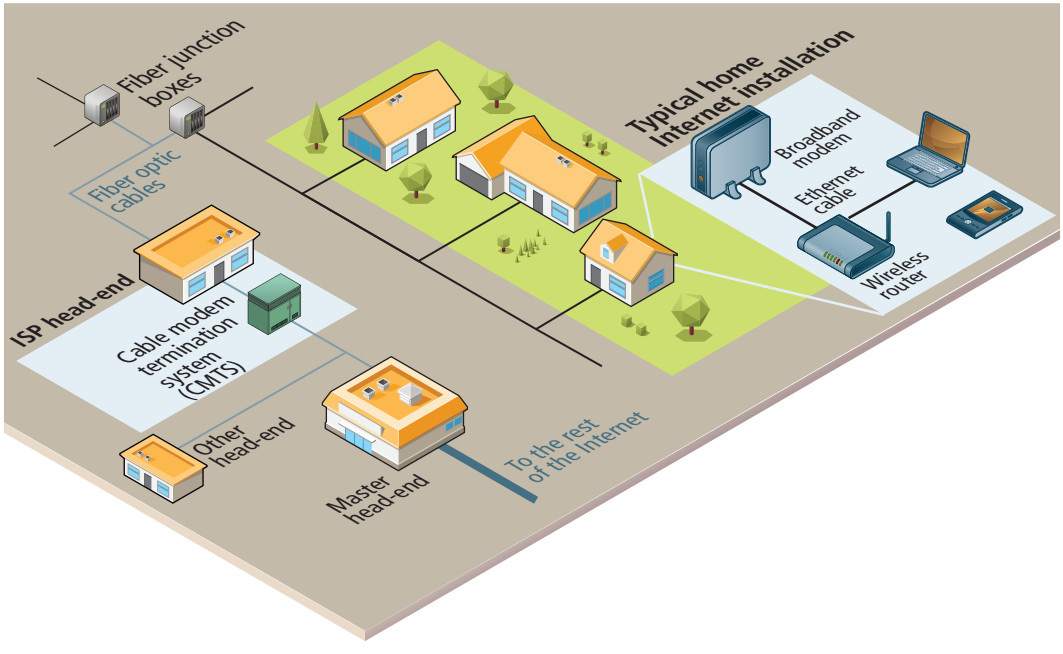


FIGURE 1.18 Internet hardware from the home computer to the local Internet provider

The wireless router is perhaps the most visible manifestation of the Internet in one's home, in that it is a device we typically need to purchase and install. Routers are in fact one of the most important and ubiquitous hardware devices that make the Internet work. At its simplest, a **router** is a hardware device that forwards data packets from one network to another network. When the router receives a data packet, it examines the packet's destination address and then forwards it to another destination by deciding the best path to send the packets.

A router uses a **routing table** to help determine where a packet should be sent. It is a table of connections between target addresses and the node (typically another router) to which the router can deliver the packet. In Figure 1.19, the different routing tables use **next-hop routing**, in which the router only knows the address of the next step of the path to the destination; it leaves it to the next step to continue routing the packet to the appropriate destination. The packet thus makes a variety of successive hops until it reaches its destination. There are a lot of details that have been left out of this particular illustration. Routers will make use of submasks,

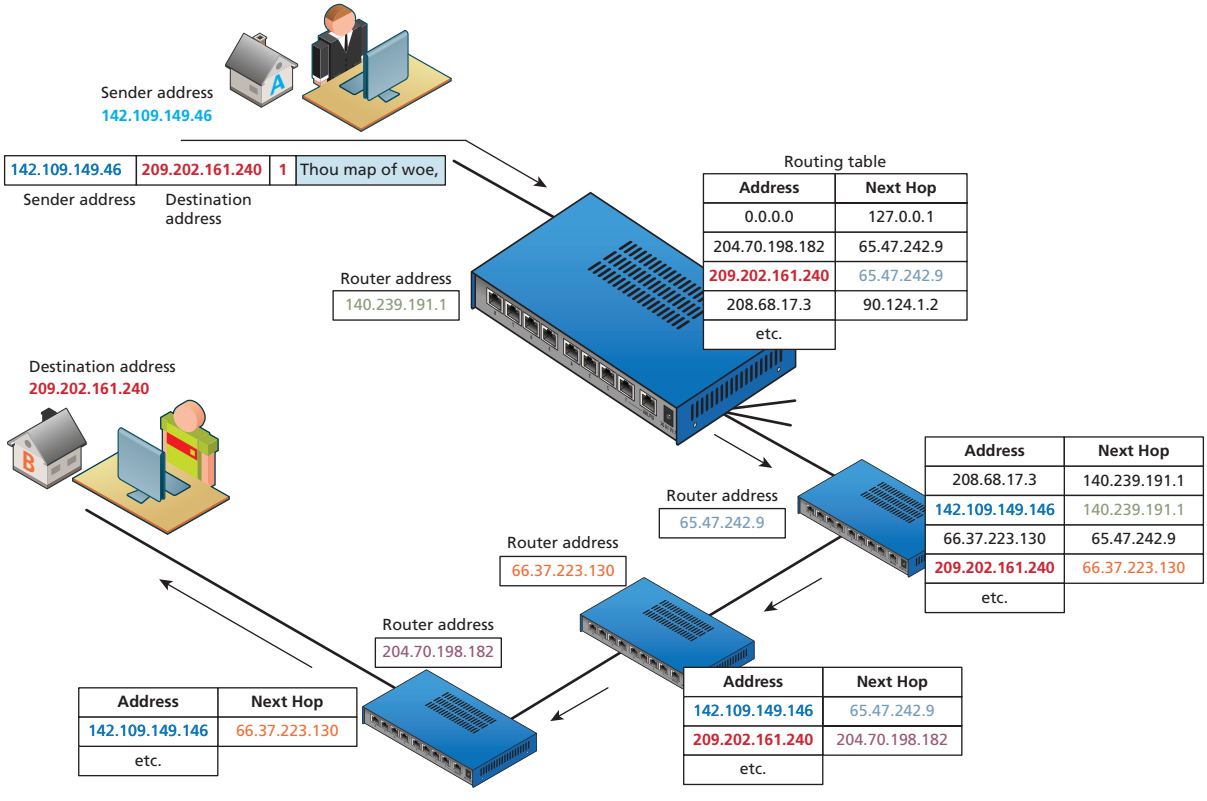


FIGURE 1.19 Simplified routing tables

timestamps, distance metrics, and routing algorithms to supplement or even replace routing tables; but those are all topics for a network architecture course.

Once we leave the confines of our own homes, the hardware of the Internet becomes much murkier. In Figure 1.18, the various neighborhood broadband cables (which are typically using copper, aluminum, or other metals) are aggregated and connected to fiber optic cable via fiber connection boxes. **Fiber optic cable** (or simply optical fiber) is a glass-based wire that transmits light and has significantly greater bandwidth and speed in comparison to metal wires. In some cities (or large buildings), you may have fiber optic cable going directly into individual buildings; in such a case the fiber junction box will reside in the building.

These fiber optic cables eventually make their way to an ISP's **head-end**, which is a facility that may contain a **cable modem termination system** (CMTS) or a digital subscriber line access multiplexer (DSLAM) in a DSL-based system. This is a special type of very large router that connects and aggregates subscriber connections to the larger Internet. These different head-ends may connect directly to the wider Internet, or instead be connected to a master head-end, which provides the connection to the rest of the Internet.

1.4.2 From the Local Provider to the Ocean's Edge

Eventually your ISP has to pass on your requests for Internet packets to other networks. This intermediate step typically involves one or more regional network hubs. Your ISP may have a large national network with optical fiber connecting most of the main cities in the country. Some countries have multiple national or regional networks, each with their own optical network. Canada, for instance, has three national networks that connect the major cities in the country as well as connect to a couple of the major Internet exchange points in the United States, as well as several provincial networks that connect smaller cities within one or two provinces. Alternatively, your smaller regional ISP may have transit arrangements with a larger national network (that is, they lease the use of part of their optical fiber network's bandwidth).

A general principle in network design is that the fewer the router hops (and thus the more direct the path), the quicker the response. Figure 1.20 illustrates some hypothetical connections between several different networks spread across four countries. As you can see, just like in the real world, the countries in the illustration differ in their degree of internal and external interconnectedness.

The networks in Country A are all interconnected, but rely on Network A1 to connect them to the networks in Country B and C. Network B1 has many connections to other countries' networks. The networks within Country C and D are not interconnected, and thus rely on connections to international networks in order to transfer information between the two domestic networks. For instance, even though the actual distance between a node in Network C1 and a node in C2 might only be a few miles, those packets might have to travel many hundreds or even thousands of miles between networks A1 and/or B1.

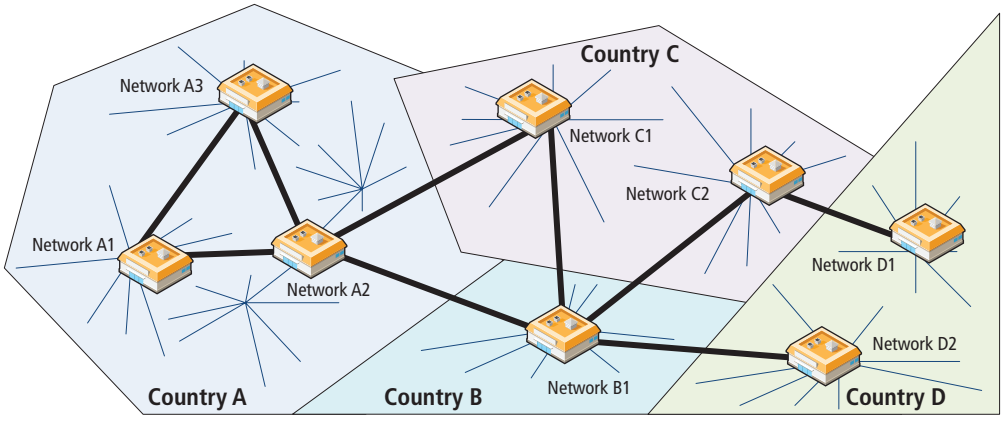


FIGURE 1.20 Connecting different networks within and between countries

Clearly this is an inefficient system, but is a reasonable approximation of the state of the Internet in the late 1990s (and in some regions of the world this is still the case), when almost all Internet traffic went through a few **Network Access Points (NAP)**, most of which were in the United States.

This type of network configuration began to change in the 2000s, as more and more networks began to interconnect with each other using an **Internet exchange point (IX or IXP)**. These IXPs allow different ISPs to **peer** with one another (that is, interconnect) in a shared facility, thereby improving performance for each partner in the peer relationship.

Figure 1.21 illustrates how the configuration shown in Figure 1.20 changes with the use of IXPs.

As you can see, IXPs provide a way for networks within a country to interconnect. Now networks in Countries C and D no longer need to make hops out of their

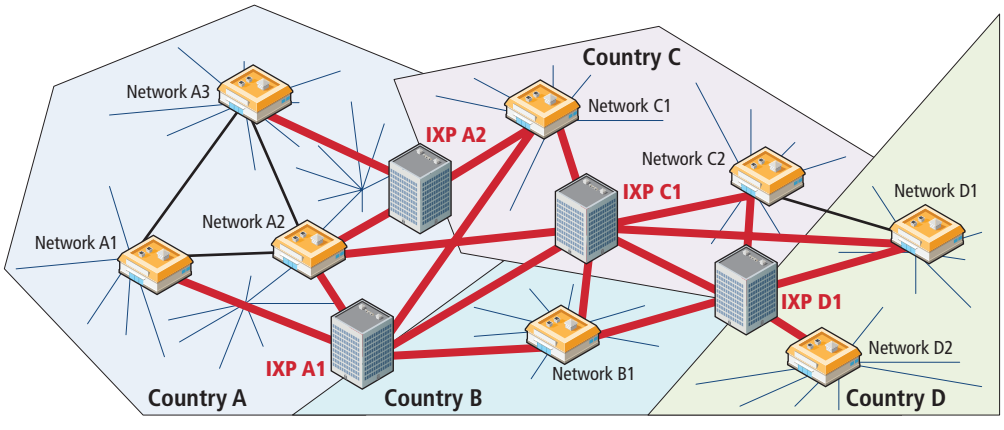


FIGURE 1.21 National and regional networks using Internet exchange points

country for domestic communications. Notice as well that for each of the IXPs, there are connections not just with networks within their country, but also with other countries' networks as well. Multiple paths between IXPs provide a powerful way to handle outages and keep packets flowing. Another key strength of IXPs is that they provide an easy way for networks to connect to many other networks at a single location.[11]

As you can see in Figure 1.22, different networks connect not only to other networks within an IXP, but now large websites such as Microsoft and Facebook are also connecting to multiple other networks simultaneously as a way of improving the performance of their sites. Real IXPs, such as at Palo Alto (PAIX), Amsterdam (AMS-IX), Frankfurt (CE-CIX), and London (LINX), allow many hundreds of networks and companies to interconnect and have throughput of over 1000 gigabits per second. The scale of peering in these IXPs is way beyond that shown in Figure 1.22 (which shows peering with only five others); companies within these IXPs use large routers from Cisco and Brocade that have hundreds of ports allowing hundreds of simultaneous peering relationships.

In recent years, major web companies have joined the network companies in making use of IXPs. As shown in Figure 1.23, this sometimes involves mirroring (duplicating) a site's infrastructure (i.e., web and data servers) in a data center located near the IXP. For instance, Equinix Ashburn IX in Ashburn, Virginia, is surrounded by several gigantic data centers just across the street from the IXP.

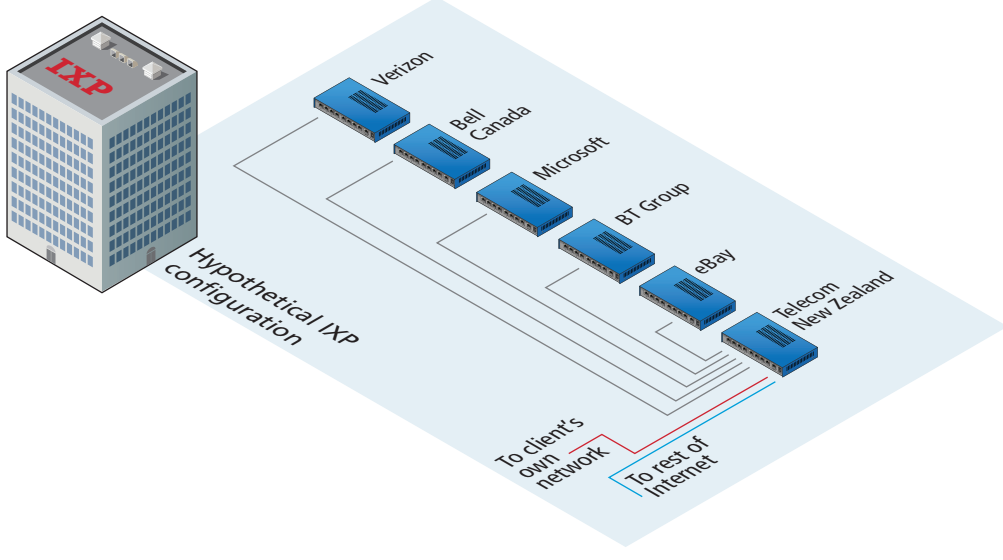


FIGURE 1.22 Hypothetical Internet exchange point

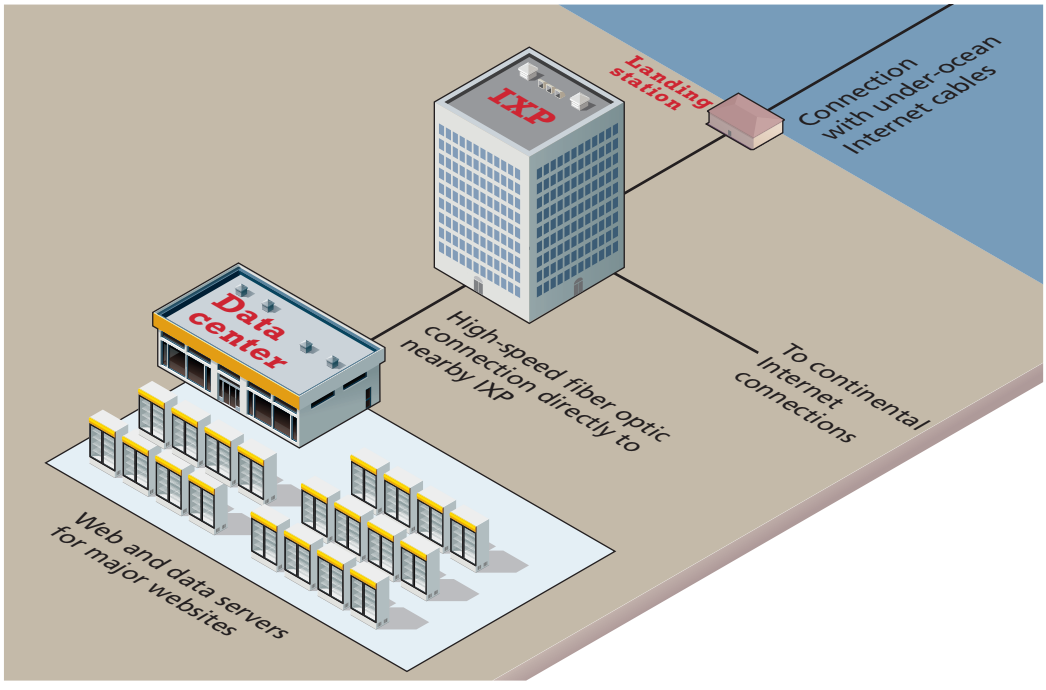


FIGURE 1.23 IXPs and data centers

This concrete geography to the digital world encapsulates an arrangement that benefits both the networks and the web companies. The website will have incremental speed enhancements (by reducing the travel distance for these sites) across all the networks it is peered with at the IXP, while the network will have improved performance for its customers when they visit the most popular websites.

1.4.3 Across the Oceans

Eventually, international Internet communication will need to travel underwater. The amount of undersea fiber optic cable is quite staggering and is growing yearly. As can be seen in Figure 1.24, over 250 undersea fiber optic cable systems operated by a variety of different companies span the globe. For places not serviced by undersea cable (such as Antarctica, much of the Canadian Arctic islands, and other small islands throughout the world), Internet connectivity is provided by orbiting satellites. It should be noted that satellite links (which have smaller bandwidth in comparison to fiber optic) account for an exceptionally small percentage of oversea Internet communication.

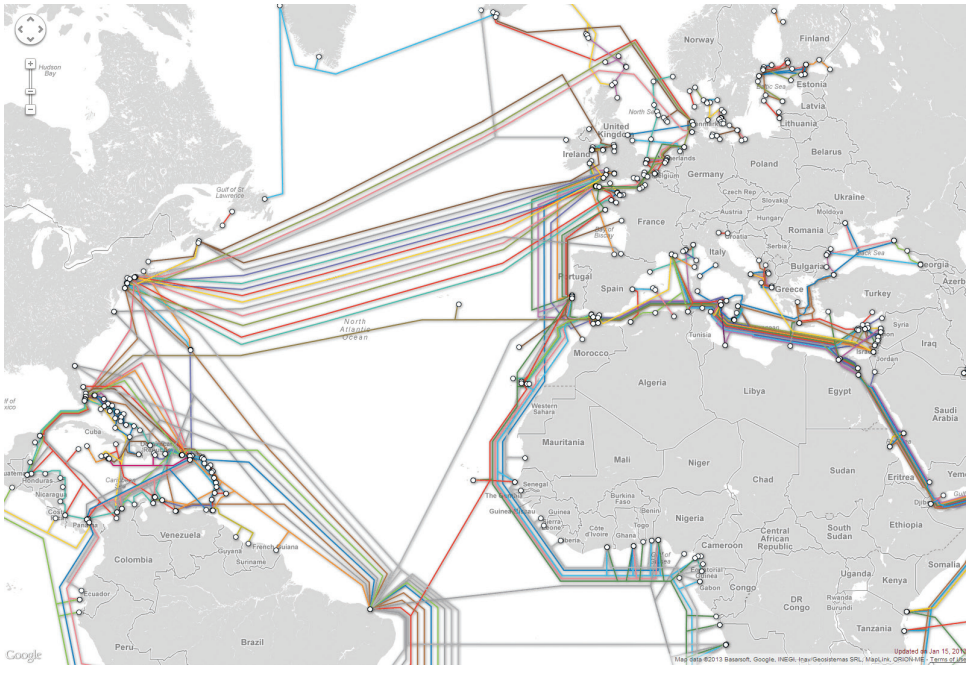


FIGURE 1.24 Undersea fiber optic cables (courtesy TeleGeography/www.submarinecablemap.com)

1.5 Domain Name System

HANDS-ON EXERCISES
LAB 1 EXERCISE
Name Servers

Back in Section 1.2, you learned about IP addresses and how they are an essential feature of how the Internet works. As elegant as IP addresses may be, human beings do not enjoy having to recall long strings of numbers. One can imagine how unpleasant the Internet would be if you had to remember IP addresses instead of domains. Rather than **google.com**, you'd have to type **173.194.33.32**. If you had to type in **69.171.237.24** to visit Facebook, it is quite likely that social networking would be a less popular pastime.

Even as far back as the days of ARPANET, researchers assigned domain names to IP addresses. In those early days, the number of Internet hosts was small, so a list of a few hundred domain and IP addresses could be downloaded as needed from the Stanford Research Institute (now SRI International) as a **hosts** file (see Pro Tip). Those key-value pairs of domain names and IP addresses allowed people to use the domain name rather than the IP address.[12]

As the number of computers on the Internet grew, this **hosts** file had to be replaced with a better, more scalable, and distributed system. This system is called the **Domain Name System (DNS)** and is shown in its most simplified form in Figure 1.25.

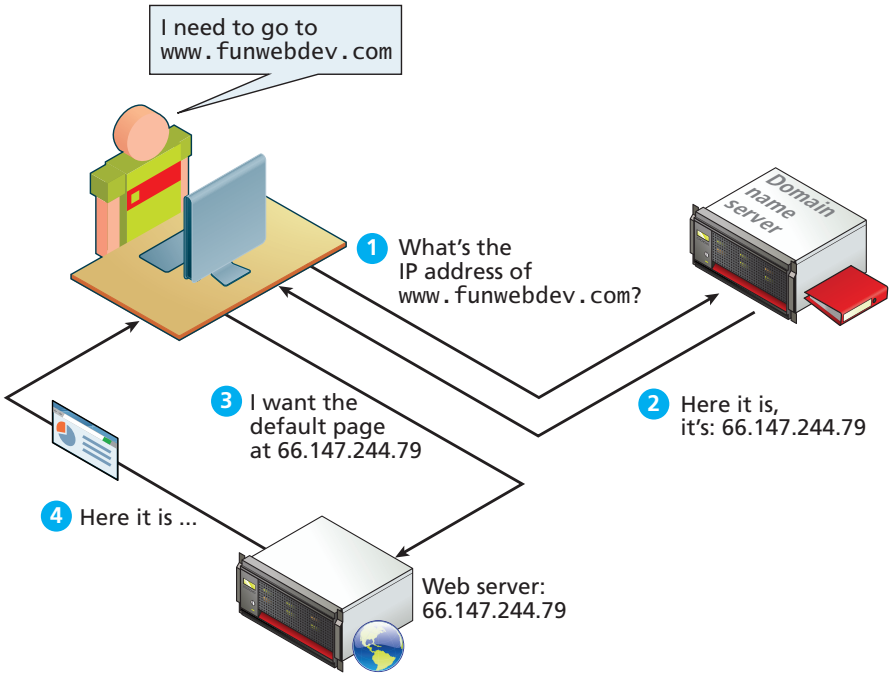


FIGURE 1.25 DNS overview

DNS is one of the core systems that make an easy-to-use Internet possible (DNS is used for email as well). The DNS system has another benefit besides ease of use. By separating the domain name of a server from its IP location, a site can move to a different location without changing its name. This means that sites and email systems can move to larger and more powerful facilities without disrupting service.

Since the entire request-response cycle can take less than a second, it is easy to forget that DNS requests are happening in all your web and email applications. Awareness and understanding of the DNS system is essential for success in developing, securing, deploying, troubleshooting, and maintaining web systems.

PRO TIP

A remnant of those earliest days still exists on most modern computers, namely the **hosts** file. Inside that file (in Unix systems typically at **/etc/hosts**) you will see domain name mappings in the following format:

```
127.0.0.1 Localhost SomeLocalDomainName.com
```

This mechanism will be used in this book to help us develop websites on our own computers with real domain names in the address bar.

(*continued*)

The same hosts file mechanism could also allow a malicious user to reroute traffic destined for a particular domain. If a malicious user ran a server at 123.56.789.1 they could modify a user's hosts to make facebook.com point to their malicious server. The end client would then type facebook.com into his browser and instead of routing that traffic to the legitimate facebook.com servers, it would be sent to the malicious site, where the programmer could phish, or steal data.

```
123.456.678.1 facebook.com
```

For this reason many system administrators and most modern operating systems do not allow access to this file without an administrator password.

1.5.1 Name Levels

A domain name can be broken down into several parts. They represent a hierarchy, with the rightmost parts being closest to the root at the "top" of the Internet naming hierarchy. All domain names have at least a top-level domain (TLD) name and a second-level domain (SLD) name. Most websites also maintain a third-level WWW subdomain and perhaps others. Figure 1.26 illustrates a domain with four levels.

The rightmost portion of the domain name (to the right of the rightmost period) is called the top-level domain. For the top level of a domain, we are limited to two broad categories, plus a third reserved for other use. They are:

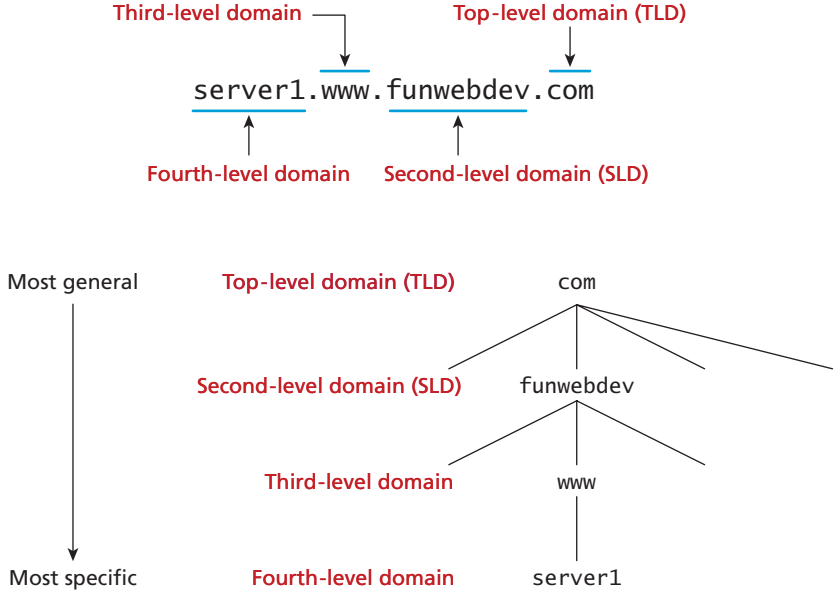


FIGURE 1.26 Domain levels

- **Generic top-level domain (gTLD)**
  - **Unrestricted.** TLDs include **.com**, **.net**, **.org**, and **.info**.
  - **Sponsored.** TLDs including **.gov**, **.mil**, **.edu**, and others. These domains can have requirements for ownership and thus new second-level domains must have permission from the sponsor before acquiring a new address.
  - **New.** From January to May of 2012, companies and individuals could submit applications for new TLDs. TLD application results were announced in June 2012, and include a wide range of both contested and single applicant domains. These include corporate ones like **.apple**, **.google**, and **.macdonalds**, and contested ones like **.buy**, **.news**, and **.music**.[13]

- **Country code top-level domain (ccTLD)**
  - TLDs include **.us**, **.ca**, **.uk**, and **.au**. At the time of writing, there were 252 codes registered.[14] These codes are under the control of the countries which they represent, which is why each is administered differently. In the United Kingdom, for example, commercial entities and businesses must register subdomains to **co.uk** rather than second-level domains directly. In Canada **.ca** domains can be obtained by any person, company, or organization living or doing business in Canada. Other countries have peculiar extensions with commercial viability (such as **.tv** for Tuvalu) and have begun allowing unrestricted use to generate revenue.
  - Since some nations use nonwestern characters in their native languages, the concept of the **internationalized top-level domain name (IDN)** has also been tested with great success in recent years. Some IDNs include Greek, Japanese, and Arabic domains (among others) which have test domains at **http://παράδειγμα.δοκιμή**, **http://例え.テスト**, and **http://مثال.إختبار**, respectively.

- `arpa`
  - The domain **.arpa** was the first assigned top-level domain. It is still assigned and used for **reverse DNS lookups** (i.e., finding the domain name of an IP address).

In a domain like **funwebdev.com**, the "**.com**" is the top-level domain and **funwebdev** is called the second-level domain. Normally it is the second-level domains that one registers.

There are few restrictions on second-level domains aside from those imposed by the registrar (defined in the next section below). Except for internationalized domain names, we are restricted to the characters A-Z, 0-9, and the "-" character. Since domain names are case-insensitive characters, a-z can also be used interchangeably.

The owner of a second-level domain can elect to have **subdomains** if they so choose, in which case those subdomains are prepended to the base hostname. For example, we can create **exam-answers.webdevfun.com** as a domain name, where **exam-answers** is the subdomain (don't bother checking . . . it doesn't exist).

NOTE

We could go further creating sub-subdomains if we wanted to. Each further level of subdomain is prepended to the front of the hostname. This allows third level, fourth, and so on. This can be used to identify individual computers on a network all within a domain.

1.5.2 Name Registration

As we have seen, domain names provide a human-friendly way to identify computers on the Internet. How then are domain names assigned? Special organizations or companies called **domain name registrars** manage the registration of domain names. These domain name registrars are given permission to do so by the appropriate generic top-level domain (gTLD) registry and/or a country code top-level domain (ccTLD) registry.

In the 1990s, a single company (Networks Solutions Inc.) handled the **com**, **net**, and **org** registries. By 1999, the name registration system changed to a market system in which multiple companies could compete in the domain name registration business. A single organization—the nonprofit **Internet Corporation for Assigned Names and Numbers (ICANN)**—still oversees the management of top-level domains, accredits registrars, and coordinates other aspects of DNS. At the time of writing this chapter, there were almost 1000 different ICANN-accredited registrars worldwide.

Figure 1.27 illustrates the process involved in registering a domain name.

1.5.3 Address Resolution

While domain names are certainly an easier way for users to reference a website, eventually your browser needs to know the IP address of the website in order to request any resources from it. DNS provides a mechanism for software to discover this numeric IP address. This process is referred to here as **address resolution**.

As shown back in Figure 1.25, when you request a domain name, a computer called a domain name server will return the IP address for that domain. With that IP address, the browser can then make a request for a resource from the web server for that domain.

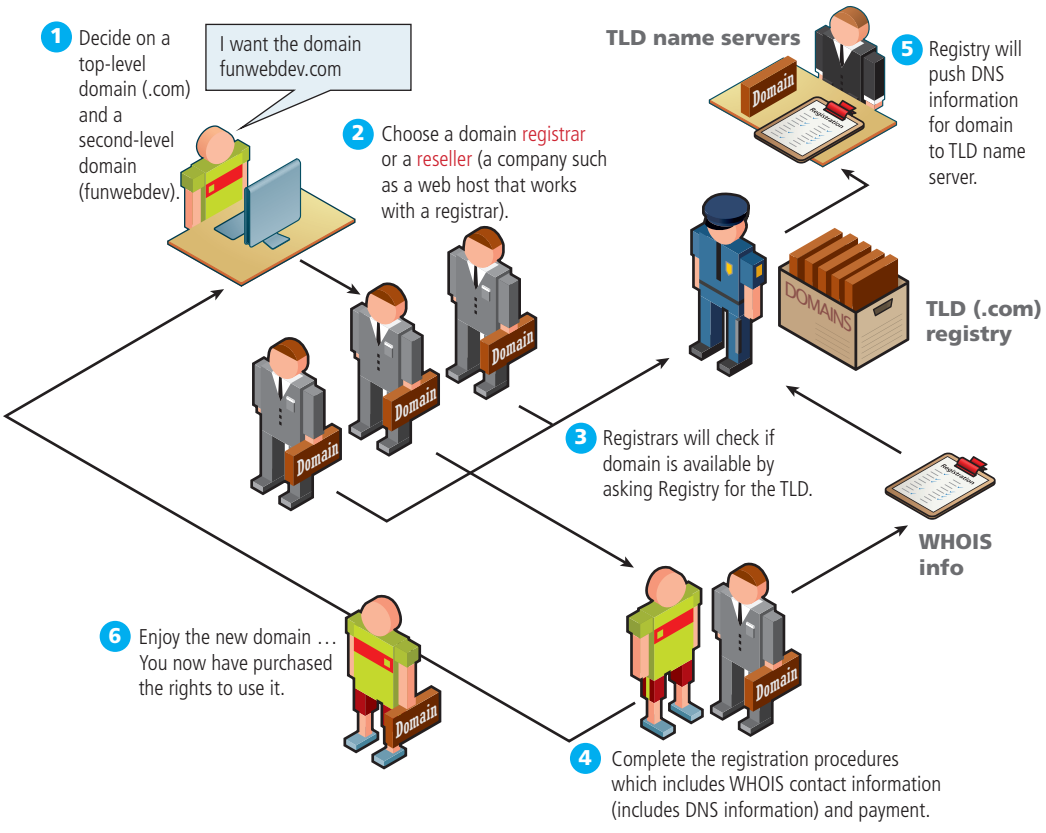


FIGURE 1.27 Domain name registration process

While Figure 1.25 provides a clear overview of the address resolution process, it is quite simplified. What actually happens during address resolution is more complicated, as can be seen in Figure 1.28.

DNS is sometimes referred to as a distributed database system of name servers. Each server in this system can answer, or look for the answer to questions about domains, caching results along the way. From a client's perspective, this is like a phonebook, mapping a unique name to a number.

Figure 1.28 is one of the more complicated ones in this text, so let's examine the address resolution process in more detail.

1. The resolution process starts at the user's computer. When the domain **www.funwebdev.com** is requested (perhaps by clicking a link or typing in a URL), the browser will begin by seeing if it already has the IP address for the domain in its **cache**. If it does, it can jump to step 13 in the diagram.
2. If the browser doesn't know the IP address for the requested site, it will delegate the task to the DNS resolver, a software agent that is part of the operating

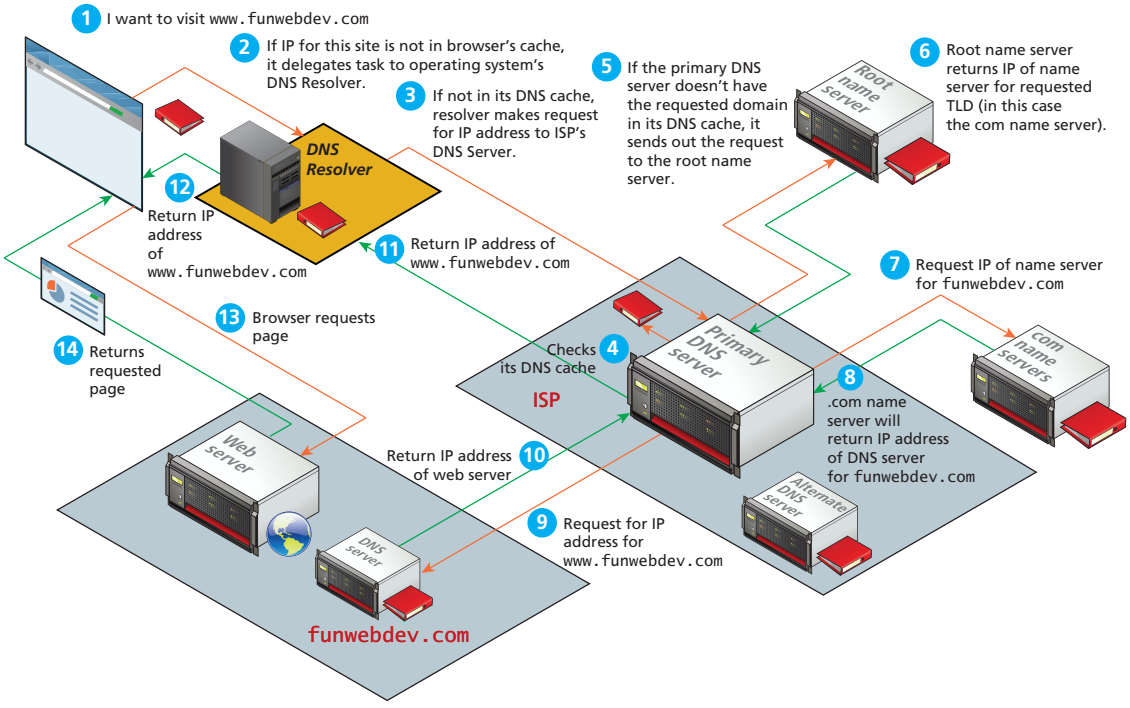


FIGURE 1.28 Domain name address resolution process

system. The DNS resolver also keeps a cache of frequently requested domains; if the requested domain is in its cache, then the process jumps to step 12.

3. Otherwise, it must ask for outside help, which in this case is a nearby DNS server, a special server that processes DNS requests. This might be a computer at your Internet service provider (ISP) or at your university or corporate IT department. The address of this local DNS server is usually stored in the network settings of your computer's operating system, as can be seen in Figure 1.9. This server keeps a more substantial cache of domain name/IP address pairs. If the requested domain is in its cache, then the process jumps to step 11.
4. If the local DNS server doesn't have the IP address for the domain in its cache, then it must ask other DNS servers for the answer. Thankfully, the domain system has a great deal of redundancy built into it. This means that in general there are many servers that have the answers for any given DNS request. This redundancy exists not only at the local level (for instance, in Figure 1.28, the ISP has a primary DNS server and an alternative one as well) but at the global level as well.
5. If the local DNS server cannot find the answer to the request from an alternate DNS server, then it must get it from the appropriate top-level

domain (TLD) name server. For **funwebdev.com** this is **.com**. Our local DNS server might already have a list of the addresses of the appropriate TLD name servers in its cache. In such a case, the process can jump to step 7.

6. If the local DNS server does not already know the address of the requested TLD server (for instance, when the local DNS server is first starting up it won't have this information), then it must ask a **root name server** for that information. The DNS root name servers store the addresses of TLD name servers. IANA (Internet Assigned Numbers Authority) authorizes 13 root servers, so all root requests will go to one of these 13 roots. In practice, these 13 machines are mirrored and distributed around the world (see **http://www.root-servers.org/** for an interactive illustration of the current root servers); at the time of writing there are a total of 350 root server machines. With the creation of new commercial top-level domains in 2012, approximately 2000 or so new TLDs will be coming online; this will create a heavier load on these root name servers.
7. After receiving the address of the TLD name server for the requested domain, the local DNS server can now ask the TLD name server for the address of the requested domain. As part of the domain registration process (see Figure 1.27), the address of the domain's DNS servers are sent to the TLD name servers, so this is the information that is returned to the local DNS server in step 8.
8. The user's local DNS server can now ask the DNS server (also called a second-level name server) for the requested domain (**www.funwebdev.com**); it should receive the correct IP address of the web server for that domain. This address will be stored in its own cache so that future requests for this domain will be speedier. That IP address can finally be returned to the DNS resolver in the requesting computer, as shown in step 11.
9. The browser will eventually receive the correct IP address for the requested domain, as shown in step 12. *Note*: If the local DNS server was unable to find the IP address, it would return a failed response, which in turn would cause the browser to display an error message.
10. Now that it knows the desired IP address, the browser can finally send out the request to the web server, which should result in the web server responding with the requested resource (step 14).

This process may seem overly complicated, but in practice it happens very quickly because DNS servers cache results. Once the server resolves **funwebdev.com**, subsequent requests for resources on **funwebdev.com** will be faster, since we can use the locally stored answer for the IP address rather than have to start over again at the root servers.

To facilitate system-wide caching, all DNS records contain a time to live (TTL) field, recommending how long to cache the result before requerying the name

server. Although this mechanism improves the efficiency and response time of the DNS system, it has a consequence of delaying propagation of changes throughout all servers. This is why administrators, after updating a DNS entry, must wait for propagation to all client ISP caches.

For more hands-on practice with the Domain Names System, please refer to Chapter 19 on Deployment.

NOTE

Every web developer should understand the practice of pointing the name servers to the web server hosting the site. Quite often, domain registrars can convince customers into purchasing hosting together with their domain. Since most users are unaware of the distinction, they do not realize that the company from which you buy web space does not need to be the same place you registered the domain. Those name servers can then be updated at the registrar to point to any name servers you use. Within 48 hours, the IP-to-domain name mapping should have propagated throughout the DNS system so that anyone typing the newly registered domain gets directed to your web server.

1.6 Uniform Resource Locators

In order to allow clients to request particular resources from the server, a naming mechanism is required so that the client knows how to ask the server for the file. For the web that naming mechanism is the Uniform Resource Locator (URL). As illustrated in Figure 1.29, it consists of two required components: the protocol used to connect, and the domain (or IP address) to connect to. Optional components of the URL are the path (which identifies a file or directory to access on that server), the port to connect to, a query string, and a fragment identifier.

1.6.1 Protocol

The first part of the URL is the protocol that we are using. Recall that in Section 1.2 we listed several application layer protocols on the TCP/IP stack. Many of those protocols can appear in a URL, and define what application protocols to use. Requesting ftp://example.com/abc.txt sends out an FTP request on port 21, while http://example.com/abc.txt would transmit on port 80.

FIGURE 1.29 URL components

1.6.2 Domain

The domain identifies the server from which we are requesting resources. Since the DNS system is case insensitive, this part of the URL is case insensitive. Alternatively, an IP address can be used for the domain.

1.6.3 Port

The optional port attribute allows us to specify connections to ports other than the defaults defined by the IANA authority. A **port** is a type of software connection point used by the underlying TCP/IP protocol and the connecting computer. If the IP address is analogous to a building address, the port number is analogous to the door number for the building.

Although the port attribute is not commonly used in production sites, it can be used to route requests to a test server, to perform a stress test, or even to circumvent Internet filters. If no port is specified, the protocol component of a URL determines which port to use.

The syntax for the port is to add a colon after the domain, then specify an integer port number. Thus for instance, to connect to our server on port 888 we would specify the URL as **http://funwebdev.com:888/**.

1.6.4 Path

The path is a familiar concept to anyone who has ever used a computer file system. The root of a web server corresponds to a folder somewhere on that server. On many Linux servers that path is **/var/www/html/** or something similar (for Windows IIS machines it is often **/inetpub/wwwroot/**). The path is case sensitive, though on Windows servers it can be case insensitive.

The path is optional. However, when requesting a folder or the top-level page of a domain, the web server will decide which file to send you. On Apache servers it is generally **index.html** or **index.php**. Windows servers sometimes use **Default.html** or **Default.aspx**. The default names can always be configured and changed.

1.6.5 Query String

Query strings will be covered in depth when we learn more about HTML forms and server-side programming. They are the way of passing information such as user form input from the client to the server. In URLs, they are encoded as key-value pairs delimited by "**&**" symbols and preceded by the "**?**" symbol. The components for a query string encoding a username and password are illustrated in Figure 1.30.

1.6.6 Fragment

The last part of a URL is the optional fragment. This is used as a way of requesting a portion of a page. Browsers will see the fragment in the URL, seek out the

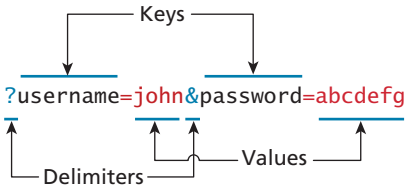


FIGURE 1.30 Query string components

fragment tag anchor in the HTML, and scroll the website down to it. Many early websites would have one page with links to content within that page using fragments and "back to top" links in each section.

1.7 Hypertext Transfer Protocol

There are several layers of protocols in the TCP/IP model, each one building on the lower ones until we reach the highest level, the application layer, which allows for many different types of services, like Secure Shell (SSH), File Transfer Protocol (FTP), and the World Wide Web's protocol, i.e., the **Hypertext Transfer Protocol (HTTP)**.

While the details of many of the application layer protocols are beyond the scope of this text, some, like HTTP, are an essential part of the web and hence require a deep understanding for a developer to build atop them successfully. We will come back to the HTTP protocol at various times in this book; each time we will focus on a different aspect of it. However, here we will just try to provide an overview of its main points.

HANDS-ON EXERCISES

LAB 1 EXERCISE
Seeing HTTP Headers

The HTTP establishes a TCP connection on port 80 (by default). The server waits for the request, and then responds with a response code, headers, and an optional message (which can include files) as shown in Figure 1.31.

The user experience for a website is unlike a user experience for traditional desktop software. Users do not download software; they visit a URL. While we as web users might be tempted to think of an entire page being returned in a single HTTP response, this is not in fact what happens.

In reality the experience of seeing a single web page is facilitated by the client's browser, which requests the initial HTML page, then parses the returned HTML to find all the resources referenced from within it, like images, style sheets, and scripts. Only when all the files have been retrieved is the page fully loaded for the user, as shown in Figure 1.32. A single web page can reference dozens of files and requires many HTTP requests and responses.

The fact that a single web page requires multiple resources, possibly from different domains, is the reality we must work with and be aware of. Modern browsers provide the developer with tools that can help us understand the HTTP traffic for a

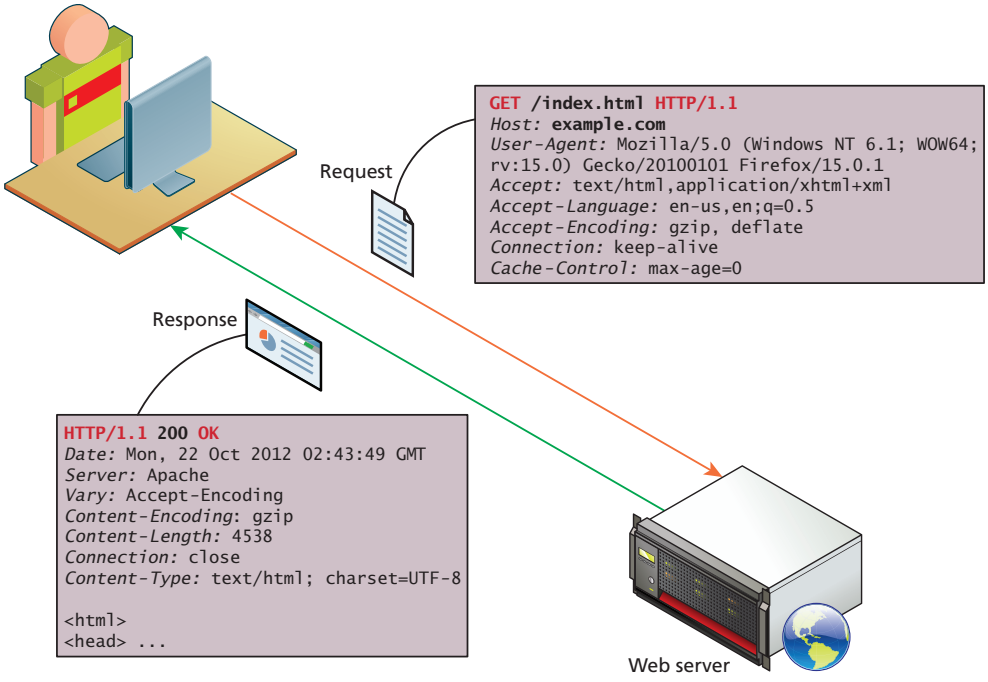


FIGURE 1.31 HTTP illustrated

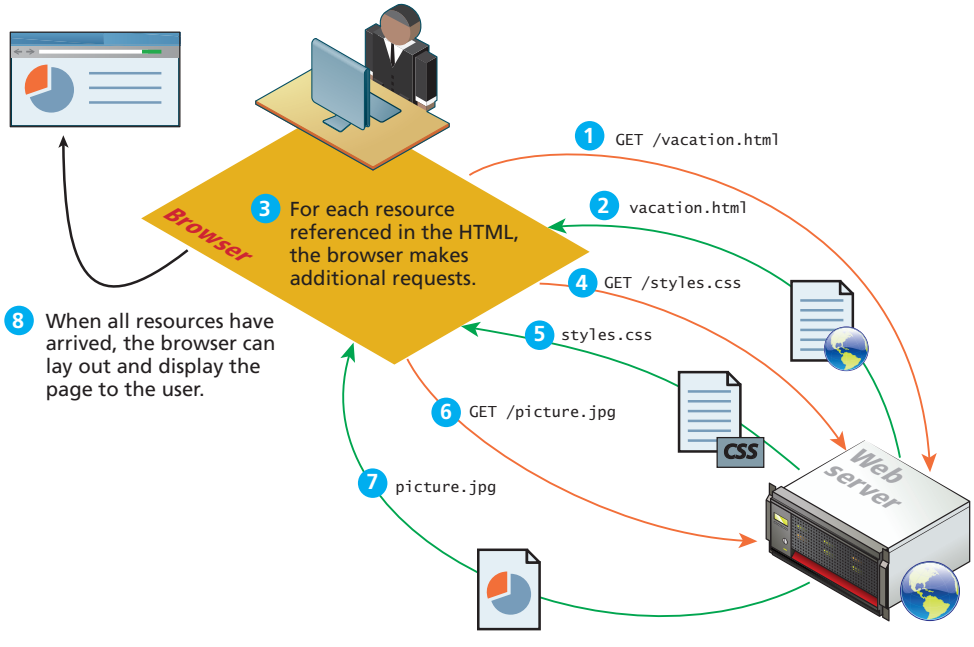


FIGURE 1.32 Browser parsing HTML and making subsequent requests

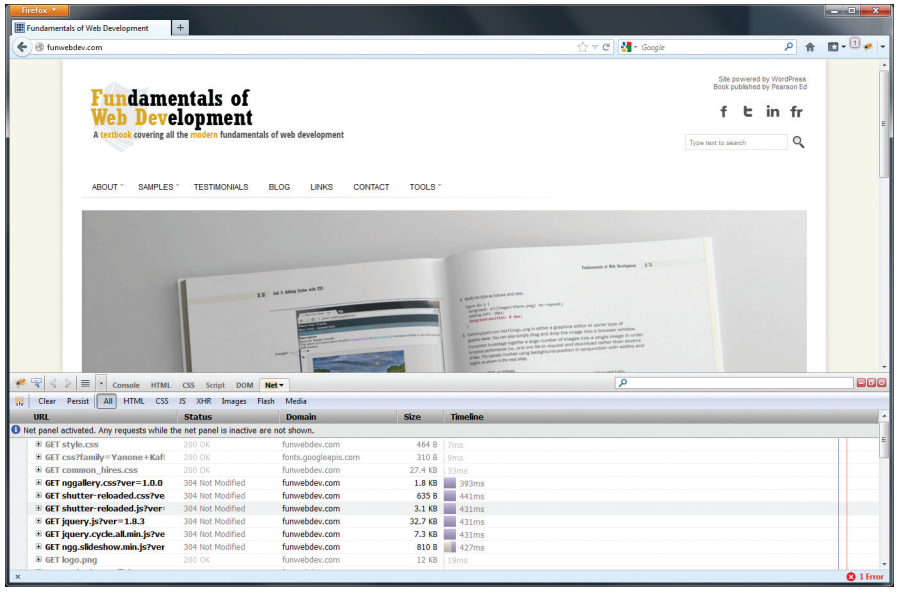


FIGURE 1.33 Distribution of load times

given page. Figure 1.33 shows a screen from the Firefox plugin FireBug (an HTML/JavaScript debugger), which lists the resources requested for a current page and the breakdown of the load times for each component.

1.7.1 Headers

Headers are sent in the request from the client and received in the response from the server. These encode the parameters for the HTTP transaction, meaning they define what kind of response the server will send. Headers are one of the most powerful aspects of HTTP and unfortunately few developers spend any time learning about them. Although there are dozens of headers,[15] we will cover a few of the essential ones to give you a sense of what type of information is sent with each and every request.

Request headers include data about the client machine (as in your personal computer). Web developers can use this information for analytic reasons and for site customization. Some of these include:

- **Host.** The `host` header was introduced in HTTP 1.1, and it allows multiple websites to be hosted off the same IP address. Since requests for different domains can arrive at the same IP, the `host` header tells the server which domain at this IP address we are interested in.
- **User-Agent.** The User-Agent string is the most referenced header in modern web development. It tells us what kind of operating system and browser

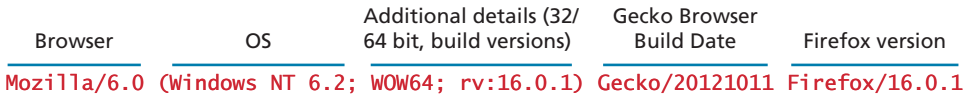


FIGURE 1.34 User-Agent components

the user is running. Figure 1.34 shows a sample string and the components encoded within. These strings can be used to switch between different style sheets and to record statistical data about the site's visitors.

- **Accept.** The `Accept` header tells the server what kind of media types the client can receive in the response. The server must adhere to these constraints and not transmit data types that are not acceptable to the client. A text browser, for example, may not accept attachment binaries, whereas a graphical browser can do so.
- **Accept-Encoding.** The `Accept-Encoding` headers specify what types of modifications can be done to the data before transmission. This is where a browser can specify that it can unzip or "deflate" files compressed with certain algorithms. Compressed transmission reduces bandwidth usage, but is only useful if the client can actually deflate and see the content.
- **Connection.** This header specifies whether the server should keep the connection open, or close it after response. Although the server can abide by the request, a response Connection header can terminate a session, even if the client requested it stay open.
- **Cache-Control.** The `Cache` header allows the client to control caching mechanisms. This header can specify, for example, to only download the data if it is newer than a certain age, never redownload if cached, or always redownload. Proper use of the `Cache-Control` header can greatly reduce bandwidth.

Response headers have information about the server answering the request and the data being sent. Some of these include:

- **Server.** The `Server` header tells the client about the server. It can include what type of operating system the server is running as well as the web server software that it is using.

NOTE

The `Server` header can provide additional information to hackers about your infrastructure. If, for example, you are running a vulnerable version of a plugin, and your Server header declares that information to any client that asks, you could be scanned, and subsequently attacked based on that header alone. For this reason, many administrators limit this field to as little info as possible.

- **Last-Modified**. `Last-Modified` contains information about when the requested resource last changed. A static file that does not change will always transmit the same last modified timestamp associated with the file. This allows cache mechanisms (like the `Cache-Control` request header) to decide whether to download a fresh copy of the file or use a locally cached copy.
- **Content-Length.** `Content-Length` specifies how large the response body (message) will be. The requesting browser can then allocate an appropriate amount of memory to receive the data. On dynamic websites where the `Last-Modified` header changes each request, this field can also be used to determine the "freshness" of a cached copy.
- **Content-Type**. To accompany the request header `Accept`, the response header `Content-Type` tells the browser what type of data is attached in the body of the message. Some media-type values are `text/html`, `image/jpeg`, `image/png`, `application/xml`, and others. Since the body data could be binary, specifying what type of file is attached is essential.
- **Content-Encoding.** Even though a client may be able to gzip decompress files and specified so in the `Accept-Encoding` header, the server may or may not choose to encode the file. In any case, the server must specify to the client how the content was encoded so that it can be decompressed if need be.

NOTE

Although compressing pages before transmission reduces bandwidth, it requires CPU cycles and memory to do so. On busy servers, sometimes it can be more efficient to transmit dynamic content uncompressed, saving those CPU cycles to respond to requests.

1.7.2 Request Methods

The HTTP protocol defines several different types of requests, each with a different intent and characteristics. The most common requests are the `GET` and `POST` request, along with the `HEAD` request. Other requests, such as `PUT`, `DELETE`, `CONNECT`, `TRACE`, and `OPTIONS` are seldom used, and are not covered here.

The most common type of HTTP request is the **GET request**. In this request one is asking for a resource located at a specified URL to be retrieved. Whenever you click on a link, type in a URL in your browser, or click on a book mark, you are usually making a `GET` request.

Data can also be transmitted through a `GET` request, something you will be learning about more in Chapter 4.

The other common request method is the **POST request**. This method is normally used to transmit data to the server using an HTML form (though as we will learn in

Chapter 4, a data entry form could use the GET method instead). In a POST request, data is transmitted through the header of the request, and as such is not subject to length limitations like with GET. Additionally, since the data is not transmitted in the URL, it is seen to be a safer way of transmitting data (although in practice all post data is transmitted unencrypted, and can be read nearly as easily as GET data). Figure 1.35 illustrates a GET and a POST request in action.

A **HEAD request** is similar to a GET except that the response includes only the header information, and not the body that would be retrieved in a full GET. Search engines, for example, use this request to determine if a page needs to be reindexed without making unneeded requests for the body of the resource, saving bandwidth.

1.7.3 Response Codes

Response codes are integer values returned by the server as part of the response header. These codes describe the state of the request, including whether it was successful, had errors, requires permission, and more. For a complete listing, please refer to the HTTP specification. Some commonly encountered codes are listed on the following page to provide a taste of what kind of response codes exist.

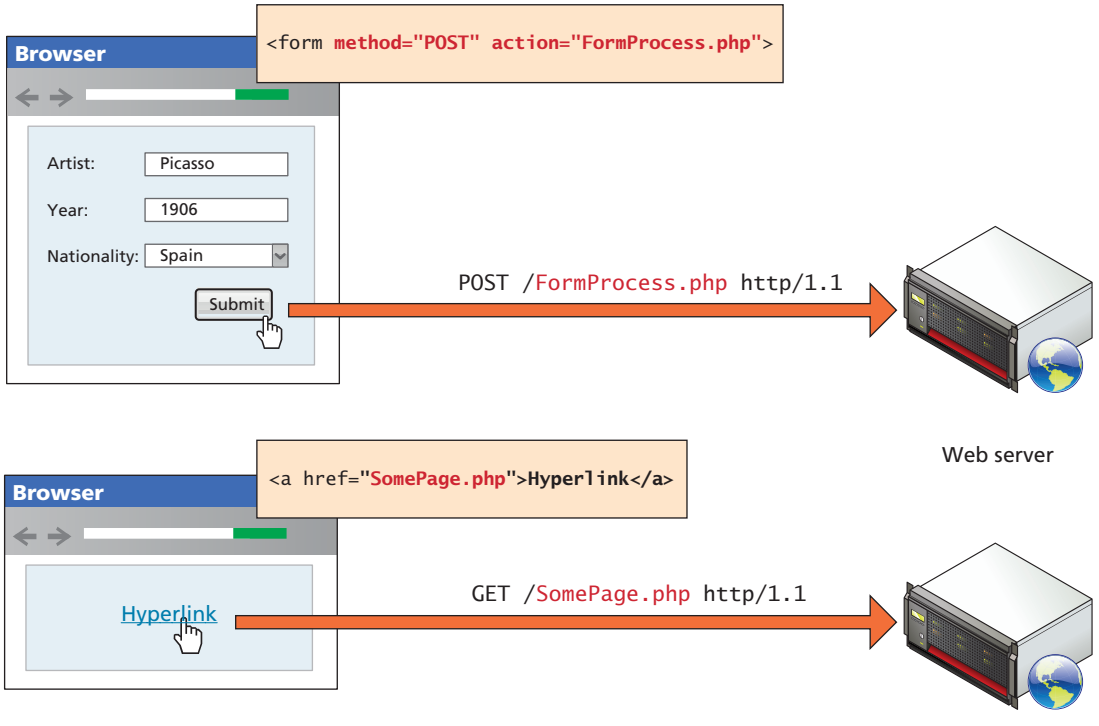


FIGURE 1.35 GET versus POST requests

Table 1.1 lists the most common response codes. The codes use the first digit to indicate the category of response. `2##` codes are for successful responses, `3##` are for redirection-related responses, `4##` codes are client errors, while `5##` codes are server errors.

1.8 Web Servers

A web server is, at a fundamental level, nothing more than a computer that responds to HTTP requests. The first web server was hosted on Tim Berners-Lee's desktop

Code	Description
`200: OK`	The 200 response code means that the request was successful.
`301: Moved Permanently`	Tells the client that the requested resource has permanently moved. Codes like this allow search engines to update their databases to reflect the new location of the resource. Normally the new location for that resource is returned in the response.
`304: Not Modified`	If the client so requested a resource with appropriate `Cache-Control` headers, the response might say that the resource on the server is no newer than the one in the client cache. A response like this is just a header, since we expect the client to use a cached copy of the resource.
`307: Temporary redirect`	This code is similar to 301, except the redirection should be considered temporary.
`400: Bad Request`	If something about the headers or HTTP request in general is not correctly adhering to HTTP protocol, the 400 response code will inform the client.
`401: Unauthorized`	Some web resources are protected and require the user to provide credentials to access the resource. If the client gets a 401 code, the request will have to be resent, and the user will need to provide those credentials.
`404: Not found`	404 codes are one of the only ones known to web users. Many browsers will display an HTML page with the 404 code to them when the requested resource was not found.
`414: Request URI too long`	URLs have a length limitation, which varies depending on the server software in place. A 414 response code likely means too much data is likely trying to be submitted via the URL.
`500: Internal server error`	This error provides almost no information to the client except to say the server has encountered an error.

TABLE 1.1 HTTP Response Codes

computer; later when you begin PHP development in Chapter 8, you may find yourself turning your own computer into a web server.

Real-world web servers are often more powerful than your own desktop computer, and typically come with additional software to make them more reliable and replaceable. And as we saw in Section 1.3.6, real-world websites typically have many web servers configured together in web farms.

Regardless of the physical characteristics of the server, one must choose an application stack to run a website. This stack will include an operating system, web server software, a database, and a scripting language to process dynamic requests.

Web practitioners often develop an affinity for a particular stack (often without rationale). Throughout this textbook we will rely on the **LAMP software stack**, which refers to the **L**inux operating system, **A**pache web server, **M**ySQL database, and **P**HP scripting language. Since Apache and MySQL also run on Windows and Mac operating systems, variations of the LAMP stack can run on nearly any computer (which is great for students). The Apple OSX MAMP software stack is nearly identical to LAMP, since OSX is a Unix implementation, and includes all the tools available in Linux. The WAMP software stack is another popular variation where Windows operating system is used.

Despite the wide adoption of the LAMP stack, web developers need to be aware of alternate software that could be used to support their websites. Many corporations, for instance, make use of the Microsoft **WISA software stack**, which refers to **W**indows operating system, **I**IS web server, **S**QL Server database, and the **A**SP.NET server-side development technologies.

1.8.1 Operating Systems

The choice of operating system will constrain what other software can be installed and used on the server. The most common choice for a web server is a Linux-based OS, although there is a large business-focused market that uses Microsoft Windows IIS.

Linux is the preferred choice for technical reasons like the higher average uptime, lower memory requirements, and the easier ability to remotely administer the machine from the command line, if required. The free cost also makes it an excellent tool for students and professionals alike looking to save on licensing costs.

Organizations that have already adopted Microsoft solutions across the organization are more likely to use a Windows server OS to host their websites, since they will have in-house Windows administrators familiar with the Microsoft suite of tools.

1.8.2 Web Server Software

If running Linux, the most likely server software is **Apache**, which has been ported to run on Windows, Linux, and Mac, making it platform agnostic. Apache is also

well suited to textbook discussion since all of its configuration options can be set through text files (although graphical interfaces exist).

IIS, the Windows server software, is preferred largely by those using Windows in their enterprises already or who prefer the .NET development framework. The most compelling reason to choose an IIS server is to get access to other Microsoft tools and products, including ASP.NET and SQL Server.

1.8.3 Database Software

The moment you decide your website will be dynamic, and not just static HTML pages, you will likely need to make use of relational database software capable of running SQL queries.

The open-source DBMS of choice is usually MySQL (though some prefer PostgreSQL or SQLite), whereas the proprietary choice for web DBMS includes Oracle, IBM DB2, and Microsoft SQL Server. All of these database servers are capable of managing large amounts of data, maintaining integrity, responding to many queries, creating indexes, creating triggers, and more. The differences between these servers are real, but are not relevant to the scope of projects we will be developing in this text.

In this book you will be using the MySQL Server, meaning if you are developing on another platform, some queries may have to be altered.

1.8.4 Scripting Software

Finally (or perhaps firstly if you are starting a project from scratch) is the choice of server-side development language or platform. This development platform will be used to write software that responds to HTTP requests. The choice for a LAMP stack is usually PHP or Python or Ruby on Rails. We have chosen PHP due to its access to low-level HTTP features, object-oriented support, C-like syntax, and its wide proliferation on the web.

Other technologies like ASP.NET are available to those interested in working entirely inside the Microsoft platform. Each technology does have real advantages and disadvantages, but we will not be addressing them here.

1.9 Chapter Summary

This long chapter has been broad in its coverage of how the Internet and the web work. It began with a short history of the Internet and how those early choices are still affecting the web today. From the design of the Internet suite of protocols you saw how IP addresses, and a multilayer stack of protocols guaranteed transmission and receipt of data. The chapter also tried to provide a picture of the hardware component of the web and the Internet, from your home router, to gigantic web farms, to the many tentacles of undersea and overland fiber optic cable. The chapter then covered some of the key protocols that make the web work: the DNS, URLs, and the HTTP protocol.

1.9.1 Key Terms

address resolution
Apache
application layer
application server
authentication server
bandwidth
broadband modem
cable modem termination system
circuit switching
client
client-server model
country code top-level domain (ccTLD)
data center
database server
DNS resolver
DNS server
domain names
domain name registrars
Domain Name System (DNS)
dynamic website
failover redundancy
fiber optic cable
four-layer network model
generic top-level domain (gTLD)
GET request
head-end
HTTP
Internet Assigned Numbers Authority (IANA)
Internet Corporation for Assigned Names and Numbers (ICANN)
internationalized top-level domain name (IDN)
intranet
Internet exchange point (IX or IXP)
Internet layer
Internet Protocol (IP)
IP address
IPv4
IPv6
LAMP software stack
link layer
load balancers
MAC addresses
mail server
media server
Mosaic
Netscape Navigator
Network Access Points (NAP)
next-hop routing
packet
packet switching
peer
peer-to-peer model
port
POST request
protocol
request
semantic web
Request for Comments (RFC)
request headers
request-response loop
response
response codes
response headers
reverse DNS lookups
root name server
router
routing table
second-level domain
server
server farm
server racks
static website
subdomain
TCP/IP (Transmission Control Protocol/ Internet Protocol)
top-level domain (TLD)
TLD name server
transport layer
Transmission Control Protocol (TCP)
User Datagram Protocol (UDP)
Uniform Resource Locator (URL)
virtual server
webmaster
Web 2.0
web server
WISA software stack
World Wide Web Consortium (W3C)

1.9.2 Review Questions

1. What is bandwidth? What is its standard unit of measurement?
2. What are the five essential elements of the early web that are still the core features of the modern web?

3. Describe the relative advantages and disadvantages of web-based applications in comparison to traditional desktop applications.
4. What is Request for Comments?
5. What is semantic web? What are the semi-standardized approaches for adding semantic qualifiers to HTML?
6. What is a virtual server?
7. What is a broadband modem?
8. What is the Internet Protocol (IP)? Why is it important for web developers?
9. What is the client-server model of communications? How does it differ from peer-to-peer?
10. Discuss the relationship between server farms, data centers, and Internet exchange points. Be sure to provide a definition for each.
11. Describe the function of a Uniform Resource Locator (URL).
12. What are the two main benefits of DNS?
13. How many levels can a domain name have? What are generic top-level domains?
14. Describe the main steps in the domain name address resolution process.
15. How many requests are involved in displaying a single web page?
16. How many distinct domains can be hosted at a single IP address?
17. What is the LAMP stack? What are some of its common variants?

1.9.3 References

1. J. Postel, "Internet Protocol," September 1981. [Online]. http://www.rfc-editor.org/rfc/rfc791.txt.
2. J. Postel, "Transmission Control Protocol," September 1981. [Online]. http://www.rfc-editor.org/rfc/rfc793.txt.
3. R. Hauben, "From the ARPANET to the Internet," 2001. [Online]. http://www.columbia.edu/~rh120/other/tcpdigest_paper.txt.
4. T. Berners-Lee, "The World Wide Web Project," December 1992. [Online]. http://www.w3.org/History/19921103-hypertext/hypertext/WWW/TheProject.html.
5. Internet Systems Consortium, "Internet host count history," July 2012. [Online]. http://www.isc.org/solutions/survey/history.
6. E. R. Braden, "Requirements for Internet Hosts—Application and Support," October 1989. [Online]. http://www.rfc-editor.org/rfc/rfc1123.txt.
7. E. R. Braden, "Requirements for Internet Hosts—Communication Layers," October 1989. [Online]. http://www.rfc-editor.org/rfc/rfc1122.txt.
8. A. S. Tanenbaum, *Computer Networks*, Prentice Hall-PTR, 2002.
9. http://huanliu.wordpress.com/2012/03/13/amazon-data-center-size/.
10. http://perspectives.mvdirona.com/2012/08/13/FunWithEnergyConsumptionData.aspx.

11. P. S. Ryan and G. Jason, "A Primer on Internet Exchange Points for Policymakers and Non-Engineers," August 2012. http://ssrn.com/abstract=2128103 or http://dx.doi.org/10.2139/ssrn.2128103.
12. P. V. Mockapetris and K. J. Dunlap, "Development of the domain name system," 123–133, in Symposium proceedings on communications architectures and protocols (SIGCOMM '88), New York, NY, 1988.
13. ICANN, "Reveal Day 13 June 2012—New gTLD Applied-For Strings," June 2012. [Online]. http://newgtlds.icann.org/en/program-status/application-results/strings-1200utc-13jun12-en.
14. World Intellectual Property Association. [Online]. http://www.wipo.int/amc/en/domains/cctld_db/index.html.
15. T. Berners-Lee et al., "Hypertext Transfer Protocol—HTTP/1.1," June 1999. [Online]. http://www.rfc-editor.org/rfc/rfc2616.txt.

2 Introduction to HTML

CHAPTER OBJECTIVES

In this chapter you will learn . . .

- A very brief history of HTML
- The syntax of HTML
- Why semantic structure is so important for HTML
- How HTML documents are structured
- A tour of the main elements in HTML
- The semantic structure elements in HTML5

This chapter provides an overview of HTML, the building block of all web pages. The massive success and growth of the web has in large part been due to the simplicity of this language. There are many books devoted just to HTML; this book covers HTML in just two chapters. As a consequence, this chapter skips over some details and instead focuses on the key parts of HTML.

2.1 What Is HTML and Where Did It Come from?

Dedicated HTML books invariably begin with a brief history of HTML. Such a history might begin with the ARPANET of the late 1960s, jump quickly to the first public specification of the HTML by Tim Berners-Lee in 1991, and then to HTML's codification by the **World Wide Web Consortium** (better known as the **W3C**) in 1997. Some histories of HTML might also tell stories about the Netscape Navigator and Microsoft Internet Explorer of the early and mid-1990s, a time when intrepid developers working for the two browser manufacturers ignored the W3C and brought forward a variety of essential new tags (such as, for instance, the `<table>` tag), and features such as CSS and JavaScript, all of which have been essential to the growth and popularization of the web.

Perhaps in reaction to these manufacturer innovations, in 1998 the W3C froze the HTML specification at version 4.01. This specification begins by stating:

> *To publish information for global distribution, one needs a universally understood language, a kind of publishing mother tongue that all computers may potentially understand. The publishing language used by the World Wide Web is HTML (from HyperText Markup Language).*

As one can see from the W3C quote, HTML is defined as a **markup language**. A markup language is simply a way of annotating a document in such a way as to make the annotations distinct from the text being annotated. Markup languages such as HTML, Tex, XML, and XHTML allow users to control how text and visual elements will be laid out and displayed. The term comes from the days of print, when editors would write instructions on manuscript pages that might be revision instructions to the author or copy editor. You may very well have been the recipient of markup from caring parents or concerned teachers at various points in your past, as shown in Figure 2.1.

At its simplest, **markup** is a way to indicate *information about the content* that is distinct from the content. This "information about content" in HTML is implemented via **tags** (or more formally, HTML elements, but more on that later). The markup in Figure 2.1 consists of the red text and the various circles and arrows and the little yellow sticky notes. HTML does the same thing but uses textual tags.

In addition to specifying "information about content" many markup languages are able to encode information how to display the content for the end user. These presentation semantics can be as simple as specifying a bold weight font for certain words, and were a part of the earliest HTML specification. Although combining semantic markup with presentation markup is no longer permitted in HTML5, "formatting the content" for display remains a key reason why HTML was widely adopted.

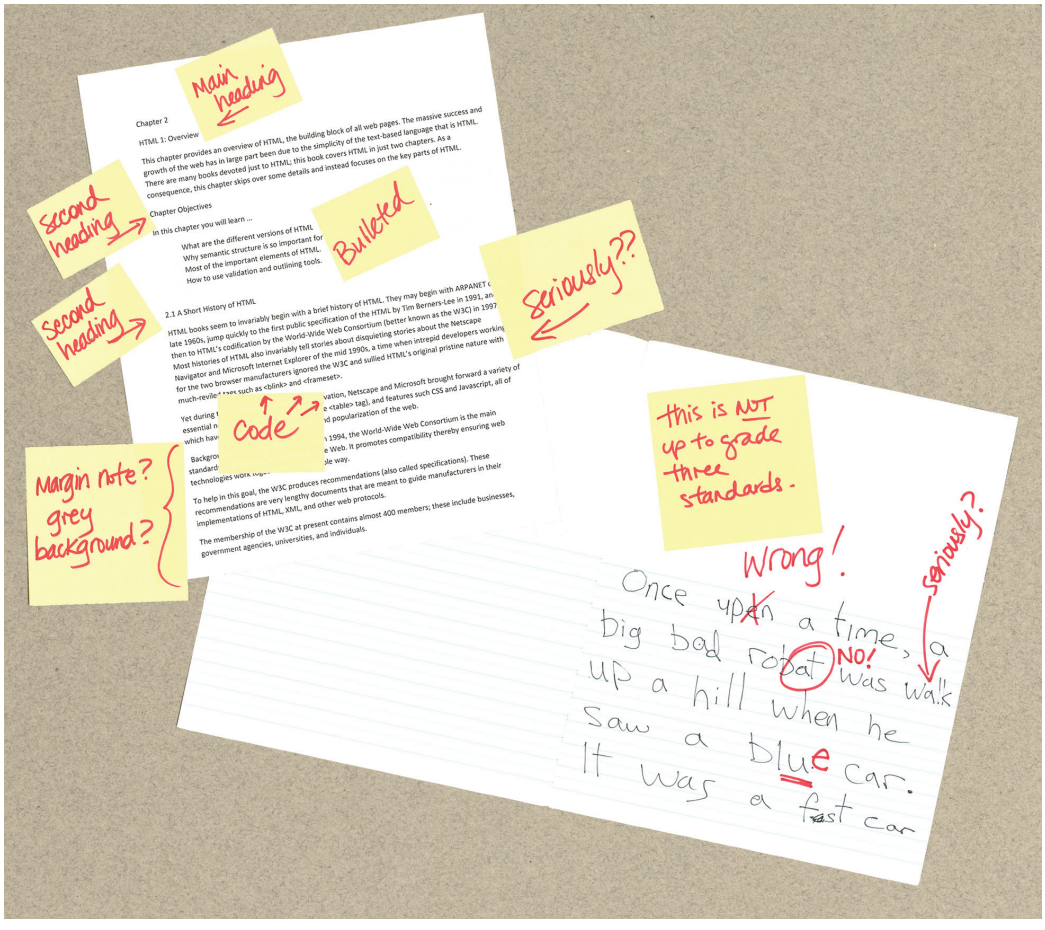


FIGURE 2.1 Sample ad-hoc markup languages

BACKGROUND

Created in 1994, the World Wide Web Consortium (W3C) is the main standards organization for the World Wide Web (WWW). It promotes compatibility, thereby ensuring web technologies work together in a predictable way.

To help in this goal, the W3C produces **Recommendations** (also called **specifications**). These Recommendations are very lengthy documents that are meant to guide manufacturers in their implementations of HTML, XML, and other web protocols.

The membership of the W3C at present consists of almost 400 members; these include businesses, government agencies, universities, and individuals.

2.1.1 XHTML

Instead of growing HTML, the W3C turned its attention in the late 1990s to a new specification called XHTML 1.0, which was a version of HTML that used stricter XML (extensible markup language) syntax rules (see Background next).

But why was "stricter" considered a good thing? Perhaps the best analogy might be that of a strict teacher. When one is prone to bad habits and is learning something difficult in school, sometimes a teacher who is more scrupulous about the need to finish daily homework may actually in the long run be more beneficial than a more permissive and lenient teacher.

As the web evolved in the 1990s, web browsers evolved into quite permissive and lenient programs. They could handle sloppy HTML, missing or malformed tags, and other syntax errors. However, it was somewhat unpredictable how each browser would handle such errors. The goal of XHTML with its strict rules was to make page rendering more predictable by forcing web authors to create web pages without syntax errors.

To help web authors, two versions of XHTML were created: XHTML 1.0 Strict and XHTML 1.0 Transitional. The strict version was meant to be rendered

BACKGROUND

Like HTML, XML is a textual markup language. Also like HTML, the formal rules for XML were set by the W3C.

XML is a more general markup language than HTML. It is (and has been) used to mark up any type of data. XML-based data formats (called schemas in XML) are almost everywhere. For instance, Microsoft Office products now use compressed XML as the default file format for the documents it creates. RSS data feeds use XML and Web 2.0 sites often use XML data formats to move data back and forth asynchronously between the browser and the server. The following is an example of a simple XML document:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<art>
  <painting id="290">
    <title>Balcony</title>
    <artist>
      <name>Manet</name>
      <nationality>France</nationality>
    </artist>
    <year>1868</year>
    <medium>Oil on canvas</medium>
  </painting>
</art>
```

By and large, the XML-based syntax rules (called "well-formed" in XML lingo) for XHTML are pretty easy to follow. The main rules are:

(*continued*)

- There must be a single root element.
- Element names are composed of any of the valid characters (most punctuation symbols and spaces are not allowed) in XML.
- Element names can't start with a number.
- Element and attribute names are case sensitive.
- Attributes must always be within quotes.
- All elements must have a closing element (or be self-closing).

XML also provides a mechanism for validating its content. It can check, for instance, whether an element name is valid, or elements are in the correct order, or that the elements follow a proper nesting hierarchy. It can also perform data type checks on the text within an element: for instance, whether the text inside an element called `<date>` is actually a valid date, or the text within an element called `<year>` is a valid integer and falls between, say, the numbers 1950 and 2010. Chapter 17 covers XML in more detail.

by a browser using the strict syntax rules and tag support described by the W3C XHTML 1.0 Strict specification; the transitional recommendation is a more forgiving flavor of XHTML, and was meant to act as a temporary transition to the eventual global adoption of XHTML Strict.

The payoff of XHTML Strict was to be predictable and standardized web documents. Indeed, during much of the 2000s, the focus in the professional web development community was on standards: that is, on limiting oneself to the W3C specification for XHTML.

A key part of the standards movement in the web development community of the 2000s was the use of **HTML validators** (see Figure 2.2) as a means of verifying that a web page's markup followed the rules for XHTML Transitional or Strict. Web developers often placed proud images on their sites to tell the world at large that their site followed XHTML rules (and also to communicate their support for web standards).

Yet despite the presence of XHTML validators and the peer pressure from book authors, actual web browsers tried to be forgiving when encountering badly formed HTML so that pages worked more or less how the authors intended regardless of whether a document was XHTML valid or not.

In the mid-2000s, the W3C presented a draft of the XHTML 2.0 specification. It proposed a revolutionary and substantial change to HTML. The most important was that backwards compatibility with HTML and XHTML 1.0 was dropped. Browsers would become significantly less forgiving of invalid markup. The XHTML 2.0 specification also dropped familiar tags such as `<img>`, `<a>`, `<br>`, and numbered headings such as `<h1>`. Development on the XHTML 2.0 specification dragged on

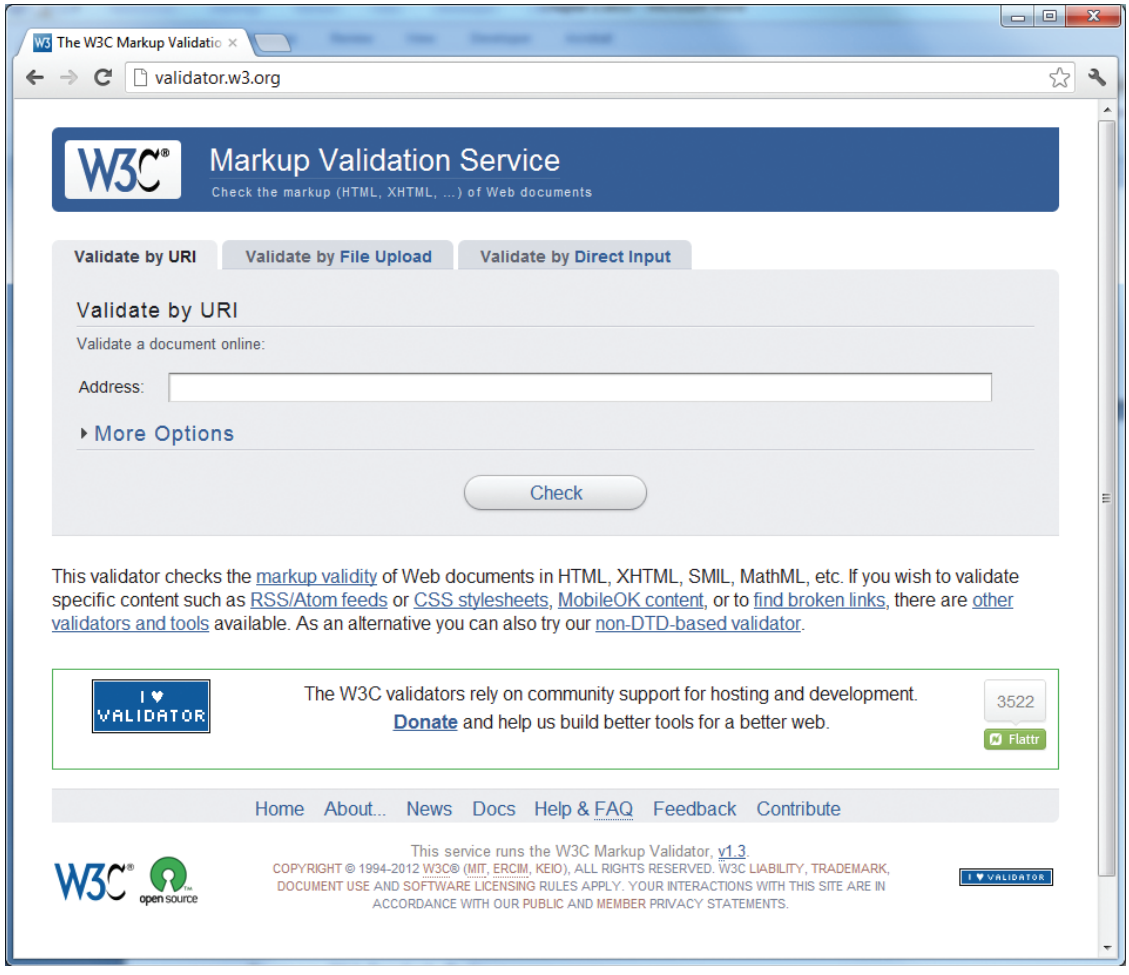


FIGURE 2.2 W3C XHTML validation service

for many years, a result not only of the large W3C committee in charge of the specification, but also of gradual discomfort on the part of the browser manufacturers and the web development community at large, who were faced with making substantial changes to all existing web pages.

2.1.2 HTML5

At around the same time the XHTML 2.0 specification was being developed, a group of developers at Opera and Mozilla formed the **WHATWG** (Web Hypertext Application Technology Working Group) group within the W3C. This group was not convinced