Steal This Book!

Yes, you read that right. Steal this book. For free.

Nothing. Zero. Zilch. Nadda. Zip.

Undoubtedly you're asking yourself, "Why would he give away a book he probably spent six grueling months writing? Is he crazy?"

The answer…is yes. Sort of. I know that every day you're faced with hundreds of computer titles, and a lot of them don't deliver the value or the information you need. So here's the deal: I'll give you this book (or this chapter, if you've only downloaded part of the book) for free provided you do me a couple favors:

1. **Send this book to your friends:** No, not your manager. Your "with it" computer friends who are looking for the next Big Thing. JXTA is it. Trust me. They want to know about it.

2. **Send a link to the book's web site:** Maybe the book is too big to send. After all, not everyone can have a fibre optic Internet connection installed in their bedroom. The site, at **www.brendonwilson.com/projects/jxta**, provides chapter-sized PDFs for easy downloading by the bandwidth-challenged.

3. **Visit the book's web site:** Being a professional developer, you probably have Carpal Tunnel Syndrome and shudder at the idea of typing in example source code. Save yourself the trouble. Go to **www.brendonwilson.com/projects/jxta** and download the source code. And while you're there, why not download some of the chapters you're missing?

4. **Buy the book:** You knew there had to be a catch. Sure, the book's PDFs are free, but I'm hoping that enough of you like the book so much that you have to buy a copy. Either that, or none of you can stand to read the book from a screen (or, worse yet, print it all out <shudder>) and resort to paper to save what's left of your eyesight. The book is available at your local bookstore or from **Amazon.com** (at a **handsome discount**, I might add).

I now return to your regularly scheduled program: enjoy the book!

1 Introduction

Peer-to-peer (P2P) technology enables any network-aware device to provide services to another network-aware device. A device in a P2P network can provide access to any type of resource that it has at its disposal, whether documents, storage capacity, computing power, or even its own human operator. Although P2P might sound like a dot-com fad, the technology is a natural extension of the Internet's philosophy of robustness through decentralization. In the same manner that the Internet provides domain name lookup (DNS), World Wide Web, email, and other services by spreading responsibility among millions of servers, P2P has the capacity to power a whole new set of robust applications by leveraging resources spread across all corners of the Internet.

Introduction to Peer-to-Peer

Most Internet services are distributed using the traditional client/server architecture, illustrated in Figure 1.1. In this architecture, clients connect to a server using a specific communications protocol, such as the File Transfer Protocol (FTP), to obtain access to a specific resource. Most of the processing involved in delivering a service usually occurs on the server, leaving the client relatively unburdened. Most popular Internet applications, including the World Wide Web, FTP, telnet, and email, use this service-delivery model.

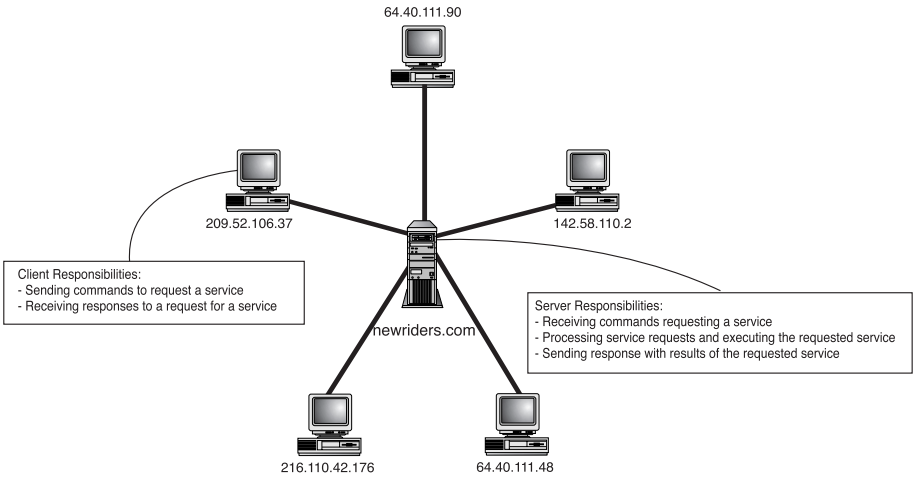

Figure 1.1 Client/server architecture.

Unfortunately, this architecture has a major drawback. As the number of clients increases, the load and bandwidth demands on the server also increase, eventually preventing the server from handling additional clients. The advantage of this architecture is that it requires less computational power on the client side. Ironically, most users have been persuaded to upgrade their computer systems to levels that are ludicrously overpowered for the most popular Internet applications: surfing the web and retrieving email.

The client in the client/server architecture acts in a passive role, capable of demanding services from servers but incapable of providing services to other clients. This model of service delivery was developed at a time when most machines on the Internet had a resolvable static IP address, meaning that all machines on the Internet could find each other easily using a simple name (such as `yourmachine.com`). If all machines on the network ran both a server and a client, they formed the foundation of a rudimentary P2P network.

As the Internet grew, the finite supply of IP addresses prompted service providers to begin dynamically allocating IP addresses to machines each time they connected to the network through dial-up connections. The dynamic nature of these machines' IP addresses effectively prevented users from running useful servers. Although someone could still run a server, that user couldn't access it unless he knew the machine's IP address beforehand. These computers form the "edge" of the Internet: machines that are connected but incapable of easily participating in the exchange of services. For this reason, most useful services are centralized on servers with resolvable IP addresses, where they can be reached by anyone who knows the server's easy-to-remember domain name.

Another reason that most clients' machines can't run servers is that they are a part of a private network, usually run by their own corporation's IT department. This private network is usually isolated from the Internet by a firewall, a device designed to prevent arbitrary connections into and out of the private network. Corporations usually create a private network to secure sensitive corporate information as well as to prevent against network abuse or misuse. The side effect of this technology is that a computer outside the private network can't connect to a computer within the private network to obtain services.

Consider the amount of computing and storage power that these client machines represent! Assume that only 10 million 100MHz machines are connected to the Internet at any one time, each possessing only 100MB of unused storage space, 1000bps of unused bandwidth, and 10% unused processing power. At any one time, these clients represent 10 petabytes (PB) (10^{15} bytes) of available storage space, 10 billion bps of available bandwidth (approximately 1.25GBps), and 10^8 MHz of wasted processing power! These are conservative estimates that only hint at the enormous untapped potential waiting to be unleashed from the "edge" of the Internet.

P2P is the key to realizing this potential, giving individual machines a mechanism for providing services to each other. Unlike the client/server architecture, P2P networks don't rely on a centralized server to provide access to services, and they usually operate outside the domain name system. As shown in Figure 1.2, P2P networks shun the centralized organization of the client/server architecture and instead employ a flat, highly interconnected architecture. By allowing intermittently connected computers to find each other, P2P enables these machines to act as both clients and servers that can determine the services available on the P2P network and engage those services in some application-specific manner.

The main advantage of P2P networks is that they distribute the responsibility of providing services among all peers on the network; this eliminates service outages due to a single point of failure and provides a more scalable solution for offering services. In addition, P2P networks exploit available bandwidth across the entire network by using a variety of communication channels and by filling bandwidth to the "edge" of the Internet. Unlike traditional client/server communications, in which specific routes to popular destinations can become overtaxed (for example, the route to Amazon.com), P2P enables communication via a variety of network routes, thereby reducing network congestion.

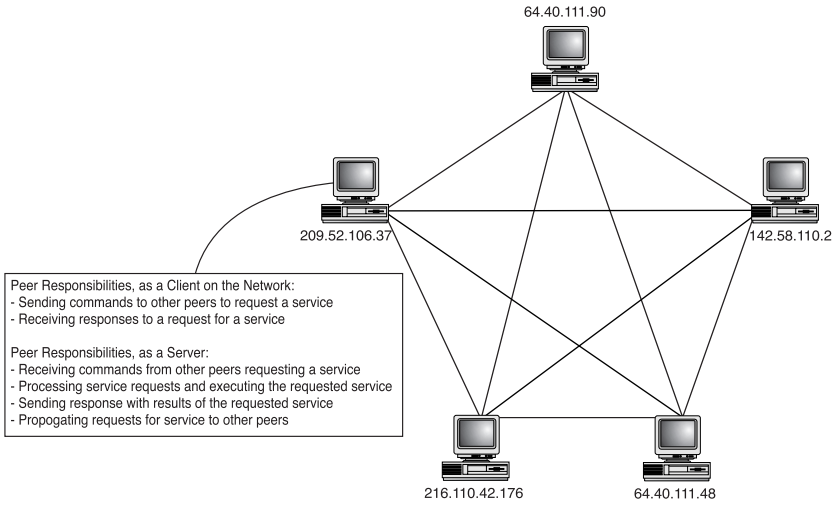

Figure 1.2 Peer-to-peer architecture.

P2P has the capability of serving resources with high availability at a much lower cost while maximizing the use of resources from every peer connected to the P2P network. Whereas client/server solutions rely on the addition of costly bandwidth, equipment, and co-location facilities to maintain a robust solution, P2P can offer a similar level of robustness by spreading network and resource demands across the P2P network. Companies such as Intel are already using P2P to reduce the cost of distributing documents and files across the entire company.

Unfortunately, P2P suffers from some disadvantages due to the redundant nature of a P2P network's structure. The distributed form of communications channels in P2P networks results in service requests that are nondeterministic in nature. For example, clients requesting the exact same resource from the P2P network might connect to entirely different machines via different communication routes, with different results. Requests sent via a P2P network might not result in an immediate response and, in some cases, might not result in *any* response. Resources on a P2P network can disappear at times as the clients that host those resources disconnect from the network; this is different from the services provided by the traditional Internet, which have most resources continuously available.

However, P2P can overcome all these limitations. Although resources might disappear at times, a P2P application might implement functionality to mirror the most popular resources over multiple peers, thereby providing redundant access to a resource. Greater numbers of interconnected peers reduce the likelihood that a request for a service will go unanswered. In short, the very structure of a P2P network that causes problems can be used to solve them.

Why Is Peer-to-Peer Important?

Although P2P gained notoriety as a means for illegally distributing copyrighted intellectual property, P2P has more to offer the computing world than easy access to stolen music or video files. To illustrate the difference between the way things are done now and how P2P could provide more useful and robust solutions, consider the following example.

To find some specific information on the Internet, I usually point my web browser to my favorite search engine, Google, and submit a search query. Most times, I'll receive a list of several thousand results, many of which are unrelated, are out-of-date, or worse yet, point to resources that no longer exist. How frustrating!

One of the problems with the current search engine solution lies in the centralization of knowledge and resources. Google relies on a central database that is updated daily by scouring the Internet for new information. Due to the number of indexed web pages in its database (more than 1.6 billion), not every entry gets updated every day. As a result of this shortcoming, the information in the Google database might not reflect the most up-to-date information available, thus diminishing the usefulness of its results for any given search query.

The search engine technology has a number of other disadvantages:

- It requires a lot of equipment. Google, for example, runs a Linux cluster of 10,000 machines to provide its service.
- If the search engine goes offline (due to, say, a network outage), all the search engine's information is unavailable.
- Due to the size of the Internet, the search engine cannot provide a comprehensive index of the Internet.
- Search engines can't interface with information stored in a corporate web site's database, meaning that the search engine can't "see" some information.

A similar service could be implemented using P2P technology, augmenting the service with additional desirable properties. Imagine if every person could run a personal web server on a desktop computer! Suppose that, in addition to serving content from the user's machine, this server had the capability to process requests for information about the documents managed by the server. A user's server could receive a query, check the documents that it manages for a match, and respond to the query with a list of matching documents.

The user's server would be responsible for indexing the documents that it made available and therefore would be capable of providing more accurate, up-to-date information on the user's documents to anyone submitting a search query. The task of indexing a single user's documents would be much more manageable than the task facing Google (a couple dozen web pages versus billions of pages). Corporations could provide gateways to connect their own web sites' databases of information to the P2P network, providing searchable access to information that the search engines currently can't reach.

The system would have this added advantage: If the user's server disconnected from the network, the search service would also become unavailable; users searching the network wouldn't receive results for resources that were unavailable. As someone searching for information, I would be almost guaranteed that any result I found using the system would be available, reducing wasted search time. I could even sort search results from the entire network to determine which information might suit my needs better based on various characteristics (such as the responsiveness of the server hosting a resource or the number of servers hosting a copy of the same resource).

This example application of P2P technology isn't perfect. For one thing, anyone wanting to drive traffic to a site could return that site as a match to any search query. However, the example illustrates the underlying principle of P2P: to enable anyone to offer services over a network. Until now, the traditional Internet experience has been mostly passive. Like the desktop publishing revolution of the mid-1980s, P2P promises to revolutionize the exchange of information.

A Brief History of P2P

Peer-to-peer has always existed, but it hasn't always been recognized as such; servers with fixed or resolvable IP addresses have always had the capability to communicate with other servers to access services. A number of pre-P2P applications, such as email and the domain name system, built on these capabilities to provide distributed networks, but one such application, Usenet, stands out from the others.

Usenet was created in 1979 by two North Carolina grad students, Tom Truscott and Jim Ellis, to provide a way for two computers to exchange information in the early days before ubiquitous Internet connectivity. Their first iteration allowed a computer to dial another computer, check for new files, and download those files; this was done at night to save on long-distance telephone charges. The system evolved into the massive newsgroup system that it is today. However, as large as Usenet is, it has a few properties that help distinguish it as probably the first P2P application. Usenet has no central managing authority—the distribution of content is managed by each node, and the content of the Usenet network is replicated (in whole or in part) across its nodes.

One of the most interesting things about Usenet is what it is: nothing! Usenet isn't a piece of software or a network of servers; although it requires software and servers to operate, these things don't truly define Usenet. At its core, Usenet is simply a way for machines to talk to each other to allow news messages to be posted and disseminated over a network. By providing a well-defined protocol, the Network News Transport Protocol (Internet Engineering Task Force RFC 977), the widest possible number of machines can participate independently to provide services. This distribution of responsibility is what distinguishes Usenet, making it recognizable as the first true, though rudimentary, application of P2P technology.

Since Usenet, the most popular P2P applications have fallen into one of three major categories: instant messaging, file sharing, and distributed computing.

Instant Messaging (IM)

When Mirabilis released ICQ (`www.icq.com`) in November 1996, it gave its users a faster way to communicate with friends than traditional email. ICQ allows users to be notified when their friends come online and to send instant messages to their friends. In addition to its main capability of instant messaging, ICQ allows users to exchange files. Though classified as a P2P application, ICQ relies on a hybrid of the P2P and client/server architectures to provide its service, as shown in Figure 1.3. ICQ uses a central server to monitor which users are currently online and to notify interested parties when new users connect to the network. All other communication between users is conducted in a P2P fashion, with messages flowing directly from one user's machine to another's with no server intermediary.

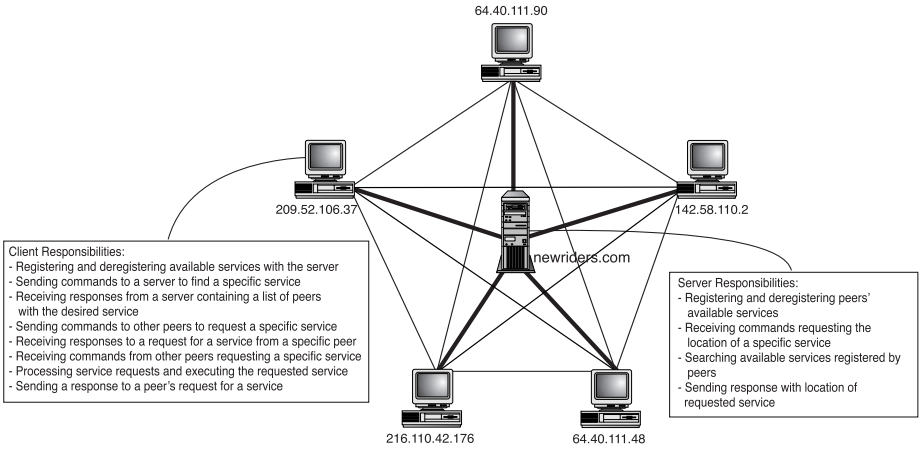

Figure 1.3 Hybrid P2P architecture.

Since its unveiling, ICQ has had many imitators, including MSN Messenger (`www.messenger.msn.com`), AOL Internet Messenger (`www.aol.com/aim`), and Yahoo! Messenger (`www.messenger.yahoo.com`). Sadly, these applications are not compatible; each relies on its own proprietary communication protocol. As a result of this incompatibility, users must download different client software and go through a separate registration process for each network. Because most users choose to avoid this inconvenience, these networks have grown into completely separate user communities that cannot interact.

More recently, various software developers have tried to bridge these separate communities by reverse-engineering the IM protocols and making new client software. One such application, Jabber (`www.jabber.com`), provides gateways to all major IM services, allowing users to interact with each other across the various IM networks. This attempt has met with resistance from service providers, prompting AOL to change its communication protocol in an attempt to block Jabber clients.

File Sharing

Napster (`www.napster.com`) burst onto the Internet stage in 1999, providing users with the capability to swap MP3 files. Napster employs a hybrid P2P solution similar to ICQ, relying on a central server to store a list of MP3 files on each user's machine. This server is also responsible for allowing users to search that list of available files to find a specific song file and its host. File transfer functionality is coordinated directly between peers without a server intermediary. In addition to its main file-sharing functionality, Napster provides a chat function to allow users to send text messages to each other.

Taking its cue from Napster, but noting the legal implications of enabling copyright infringement, the Gnutella project (`www.gnutelliums.com`) took the file-sharing concept pioneered by Napster one step further and eliminated the need for a central server to provide search functionality. The Gnutella network's server independence, combined with its capability to share any type of file, makes it one of the most powerful demonstrations of P2P technology.

Peers on the Gnutella network are responsible not only for serving files, but also for responding to queries and routing messages to other peers. Note that although Gnutella doesn't require a central server to provide search and IP address resolution functionality, connecting to the Gnutella network still requires that a peer know the IP address of a peer already connected to the P2P network. For this reason, a number of peers with static or resolvable IP addresses have been established to provide new peers with a starting point for discovering other peers on the network.

Eliminating the reliance on a central server has raised a number of new issues:

- How do peers distribute messages to each other without flooding the network?
- How do peers provide content securely and anonymously?
- How can the network encourage resource sharing?

Other file-sharing P2P variants, including Freenet (`freenet.sourceforge.net`), Morpheus (`www.musiccity.com`), and MojoNation (`www.mojonation.net`), have stepped into the arena to address these issues. Each of these applications addresses a specific issue. Freenet provides decentralized anonymous content storage protected by strong cryptography against tampering. Morpheus provides improved search capabilities based on metadata embedded in common media formats. MojoNation uses an artificial currency, called Mojo, to enforce resource sharing.

> **The Tragedy of the Commons**
>
> In many communities that share resources, there is a risk of suffering from the "Tragedy of the Commons": the overuse of a shared resource to the point of its destruction. The Tragedy of the Commons originally referred to the problem of overgrazing on public lands, but the term can apply to any public resource that can be used without restriction.
>
> In some P2P systems, peers can use the resources (bandwidth and storage space) of others on the network without making resources of their own available to the network, thereby reducing the value of the network. As more users choose not to share their resources, those peers that do share resources come under increased load and, in many ways, the network begins to revert to the classic client/server architecture. Taken to its logical conclusion, the network eventually collapses, benefiting no one.

> Newer P2P solutions have tried to prevent the Tragedy of the Commons by incorporating checks to ensure that users share resources. Lime Wire (`www.limewire.com`), for example, allows users to restrict downloads based on the number of files that a requesting client is sharing with the network. MojoNation (`www.mojonation.net`) takes this model one step further and incorporates a system of currency that users earn by sharing resources and then spend to access resources.

Distributed Computing

Distributed computing is a way of solving difficult problems by splitting the problem into subproblems that can be solved independently by a large number of computers. Although the most popular applications of distributed computing have not been P2P solutions, it is important to note the breakthrough work that has been accomplished by projects such as SETI@Home (`setiathome.berkeley.edu`) and Distributed.net (`distributed.net`) and companies such as United Devices (`www.ud.com`).

In 1996, SETI@Home began distributing a screen saver–based application to users, to allow them to process radio-telescope data and contribute to the search for extraterrestrial life. Since then, it has signed up more than 3 million users (of which more than a half million are active contributors). In a similar project started in 1997, Distributed.net used the computing power of its users to crack previously unbreakable encrypted messages. In both cases, the client software contacts a server to download its portion of the problem being solved; until the problem is solved, no further communication with the server is required.

In the future, it is expected that distributed computing will evolve to take full advantage of P2P technology to create a marketplace for spare computing power.

Introducing Project JXTA

As you probably noticed, most of the P2P solutions overlap in some shape or form: ICQ provides instant messaging plus a bit of file sharing. Napster provides file sharing plus a bit of instant messaging. You could even say that Gnutella provides file sharing, plus a bit of distributed computing, due to the way that peers take on the task of routing messages across the network.

Regrettably, the current applications of P2P tend to use protocols that are proprietary and incompatible in nature, reducing the advantage offered by gathering devices into P2P networks. Each network forms a closed community, completely independent of the other networks and incapable of leveraging their services.

Until now, the excitement of exploring the possibilities of P2P technology has overshadowed the importance of interoperability and software reuse. To evolve P2P into a mature solution platform, developers need to refocus their efforts from programming P2P network fundamentals to creating P2P applications on a solid, well-defined base. To do this, P2P developers need a common language to allow peers to communicate and perform the fundamentals of P2P networking.

Realizing this need for a common P2P language, Sun Microsystems formed Project JXTA (pronounced *juxtapose* or *juxta*), a small development team under the guidance of Bill Joy and Mike Clary, to design a solution to serve all P2P applications. At its core, JXTA is simply a set of protocol specifications, which is what makes it so powerful. Anyone who wants to produce a new P2P application is spared the difficulty of properly designing protocols to handle the core functions of P2P communication.

> **What Does JXTA Mean?**
> The name JXTA is derived from the word *juxtapose*, meaning to place two entities side by side or in proximity. By choosing this name, the development team at Sun recognized that P2P solutions would always exist alongside the current client/server solutions rather than replacing them completely.

The JXTA v1.0 Protocols Specification defines the basic building blocks and protocols of P2P networking:

- **Peer Discovery Protocol**—Enables peers to discover peer services on the network
- **Peer Resolver Protocol**—Allows peers to send and process generic requests
- **Rendezvous Protocol**—Handles the details of propagating messages between peers
- **Peer Information Protocol**—Provides peers with a way to obtain status information from other peers on the network
- **Pipe Binding Protocol**—Provides a mechanism to bind a virtual communication channel to a peer endpoint
- **Endpoint Routing Protocol**—Provides a set of messages used to enable message routing from a source peer to a destination peer

The JXTA protocols are language-independent, defining a set of XML messages to coordinate some aspect of P2P networking. Although some developers in the P2P community protest the use of such a verbose language, the choice of XML allows implementers of the JXTA protocols to leverage existing

toolsets for XML parsing and formatting. In addition, the simplicity of the JXTA protocols makes it possible to implement P2P solutions on any device with a "digital heartbeat," such as PDAs or cell phones, further expanding the number of potential peers.

In April 2001, Bill Joy placed Project JXTA in the hands of the P2P development community by adopting a license based on the Apache Software License Version 1.1. In addition to maintaining the JXTA v1.0 Protocols Specification, Project JXTA is responsible for the development of reference implementations of the JXTA platform and source code control for a variety of JXTA Community projects. Currently, Project JXTA has a reference implementation available in Java, with implementations in C, Objective-C, Ruby, and Perl 5.0 under way. At this time, Project JXTA houses a variety of JXTA Community projects that are applying JXTA technology in diverse fields such as content management, artificial intelligence, and secure anonymous payment systems.

Summary

This chapter provided an introduction to P2P and outlined the problems of the traditional client/server architecture that P2P can be used to solve. The advantages and shortcomings of current P2P solutions were presented, and the JXTA solution was briefly introduced.

The next chapter examines the common problems that face P2P implementations and how they can be solved. These solutions are presented independently of the JXTA technology but use the JXTA terminology. This allows the chapter to provide a high-level overview of P2P that doesn't overwhelm the reader with JXTA-specific details.