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!

2

P2P Concepts

IT'S NECESSARY TO INTRODUCE THE TERMINOLOGY and concepts of JXTA and place them in the general framework that's common to all P2P networks. This chapter introduces the terminology used to describe aspects of P2P networks, the components common to all P2P solutions (including those not built using JXTA technology), and the problems and solutions inherent in P2P networks.

Elements of P2P Networks

P2P is the solution to a straightforward question: How can you connect a set of devices in such a way that they can share information, resources, and services? On the surface, it seems a simple question, but to answer it properly requires answering several implied questions:

- How does one device learn of another device's presence?
- How do devices organize to address common interests?
- How does a device make its capabilities known?
- What information is required to uniquely identify a device?
- How do devices exchange data?

All P2P networks build on fundamental elements to provide the answers to these questions and others. Unfortunately, many of these elements are assumed or implied by proprietary P2P networks and are hard-coded into many P2P applications' implementations, resulting in inflexibility. For example, the majority of current P2P solutions assumes the use of TCP as a network transport mechanism and cannot operate in any other network environment. Flexible P2P solutions need a language that explicitly declares all of the variables in any P2P solution.

The following sections define the basic terminology of P2P networking. I've tried to provide definitions at this point that use the JXTA terminology while omitting the JXTA-specific implementation details. This will help you learn the language of JXTA and P2P without being overwhelmed by JXTA-specific details.

Peers

A *peer* is a node on a P2P network that forms the fundamental processing unit of any P2P solution. Until now, you might have described a peer as an application running on a single computer connected to a network such as the Internet, but that limited definition wouldn't capture the true function of a peer and all its possible incarnations. This limited definition discounts the possibility that a peer might be an application distributed over several machines or that a peer might be a smaller device, such as a PDA, that connects to a network indirectly, such as via a synching cradle. A single machine might even be responsible for running multiple peer instances.

To encompass all these facets, this book defines a peer as follows:

> Any entity capable of performing some useful work and communicating the results of that work to another entity over a network, either directly or indirectly.

The definition of *useful work* depends on the type of peer. Three possible types of peers exist in any P2P network:

- Simple peers
- Rendezvous peers
- Router peers

Each peer on the network can act as one or more types of peer, with each type defining a different set of responsibilities for the peer to the P2P network as a whole.

Simple Peers

A simple peer is designed to serve a single end user, allowing that user to provide services from his device and consuming services provided by other peers on the network. In all likelihood, a simple peer on a network will be located behind a firewall, separated from the network at large; peers outside the firewall will probably not be capable of directly communicating with the simple peer located inside the firewall.

Because of their limited network accessibility, simple peers have the least amount of responsibility in any P2P network. Unlike other peer types, they are not responsible for handling communication on behalf of other peers or serving third-party information for consumption by other peers.

Rendezvous Peers

Taken literally, a rendezvous is a gathering or meeting place; in P2P, a rendezvous peer provides peers with a network location to use to discover other peers and peer resources. Peers issue discovery queries to a rendezvous peer, and the rendezvous provides information on the peers it is aware of on the network. How a rendezvous peer discovers other peers on its local network will be discussed in the section "P2P Communication," later in this chapter.

A rendezvous peer can augment its capabilities by caching information on peers for future use or by forwarding discovery requests to other rendezvous peers. These schemes have the potential to improve responsiveness, reduce network traffic, and provide better service to simple peers.

A rendezvous peer will usually exist outside a private internal network's firewall. A rendezvous could exist behind the firewall, but it would need to be capable of traversing the firewall using either a protocol authorized by the firewall or a router peer outside the firewall.

Router Peers

A router peer provides a mechanism for peers to communicate with other peers separated from the network by firewall or Network Address Translation (NAT) equipment. A router peer provides a go-between that peers outside the firewall can use to communicate with a peer behind the firewall, and vice versa. This technique of firewall and NAT traversal is discussed in detail in the upcoming section "Challenges to Direct Communication."

To send a message to a peer via a router, the peer sending the message must first determine which router peer to use to communicate with the destination peer. This routing information provides a mechanism in P2P to replace traditional DNS, enabling an intermittently connected device with a dynamic IP

address to be found on the network. In a similar manner to the way that DNS translates a simple name to an IP address, routing information provides a mapping between a unique identifier specifying a remote peer on the network and a representation that can be used to contact the remote peer via a router peer.

In simple systems, routing information might consist solely of resolving an IP address and a TCP port for a given unique identifier. A more complex system might provide routing information consisting of an ordered list of router peers to use to properly route a message to a peer. Routing a message through multiple router peers might be necessary to allow two peers to communicate by using a router peer to translate between two different and incompatible network transports.

Peer Groups

Before JXTA, the proprietary and specialized nature of P2P solutions and their associated protocols divided the usage of the network space according to the application. If you wanted to perform file sharing, you probably used the Gnutella protocol and could communicate only with other peers using the Gnutella protocol; similarly, if you wanted to perform instant messaging, you used ICQ and could communicate only with other peers also using ICQ.

The protocols' incompatibilities effectively divided the network space based on the application being used by the peers involved. If you consider a P2P system in which all clients can speak the same set of protocols, as they can in JXTA, the concept of a peer group is necessary to subdivide the network space. As you would probably expect, a *peer group* is defined as follows:

> A set of peers formed to serve a common interest or goal dictated by the peers involved. Peer groups can provide services to their member peers that aren't accessible by other peers in the P2P network.

Peer groups divide the P2P network into groups of peers with common goals based on the following:

- **The application they want to collaborate on as a group.** A peer group is formed to exchange service that the members do not want to have available to the entire population of the P2P network. One reason for doing this could be the private nature of the data used by the application.
- **The security requirements of the peers involved.** A peer group can employ authentication services to restrict who can join the group and access the services offered by the group.

- **The need for status information on members of the group**. Members of a peer group can monitor other members. Status information might be used to maintain a minimum level of service for the peer group's application.

Peer group members can provide redundant access to a service, ensuring that a service is always available to a peer group as long as at least one member is providing the service.

Network Transport

To exchange data, peers must employ some type of mechanism to handle the transmission of data over the network. This layer, called the *network transport*, is responsible for all aspects of data transmission, including breaking the data into manageable packets, adding appropriate headers to a packet to control its destination, and in some cases, ensuring that a packet arrives at its destination. A network transport could be a low-level transport, such as UDP or TCP, or a high-level transport, such as HTTP or SMTP.

The concept of a network transport in P2P can be broken into three constituent parts:

- **Endpoints**—The initial source or final destination of any piece of data being transmitted over the network. An endpoint corresponds to the network interfaces used to send and receive data.
- **Pipes**—Unidirectional, asynchronous, virtual communications channels connecting two or more endpoints.
- **Messages**—Containers for data being transmitted over a pipe from one endpoint to another.

To communicate using a pipe, a peer first needs to find the endpoints, one for the source of the message and one for each destination of the message, and connect them by binding a pipe to each of the endpoints. When bound this way, the endpoint acting as a data source is called an *output pipe* and the endpoint acting as a data sink is called an *input pipe*. The pipe itself isn't responsible for actually carrying data between the endpoints; it's merely an abstraction used to represent the fact that two endpoints are connected. The endpoints themselves provide the access to the underlying network interface used for transmitting and receiving data.

To send data from one peer to another, a peer packages the data to be transmitted into a message and sends the message using an output pipe; on the opposite end, a peer receives a message from an input pipe and extracts the transmitted data.

Notice that a pipe provides communication in only one direction, thus requiring two pipes to achieve two-way communication between two peers. The definition of a pipe is structured this way to capture the lowest common denominator possible in network communications, to avoid excluding any possible network transports. Although bidirectional communication is the norm in modern networks, there's no reason to exclude the possibility of a unidirectional communications channel in the definition because any bidirectional network transport can easily be modeled using two unidirectional pipes.

Services

Services provide functionality that peers can engage to perform "useful work" on a remote peer. This work might include transferring a file, providing status information, performing a calculation, or basically doing anything that you might want a peer in a P2P network to be capable of doing. Services are the motivation for gathering devices into a P2P network; without services, you don't a have a P2P network—you have just a set of devices incapable of leveraging each other's resources.

Services can be divided into two categories:

- **Peer services**—Functionality offered by a particular peer on the network to other peers. The capabilities of this service will be unique to the peer and will be available only when the peer is connected to the network. When the peer disconnects from the network, the service is no longer available.
- **Peer group services**—Functionality offered by a peer group to members of the peer group. This functionality could be provided by several members of the peer group, thereby providing redundant access to the service. As long as one member of the peer group is connected to the network and is providing the service, the service is available to the peer group.

Most of the functionality required to create and maintain a P2P network, such as the underlying protocols required to find peers and resources, could also be considered services. These *core services* provide the basic P2P foundation used to build other, more complex services.

Advertisements

Until now, P2P applications have used an informal form of advertisements. In Gnutella, the results returned by a search query could be considered an

advertisement that specifies the location of a specific song file on the Gnutella network. These primitive advertisements are extremely limited in their purpose and application. At its core, an *advertisement* is defined as follows:

> A structured representation of an entity, service, or resource made available by a peer or peer group as a part of a P2P network.

All the building blocks discussed up to this point in the chapter can be described by advertisements, including peers, peer groups, pipes, endpoints, services, and content. When you start looking at advertisements in JXTA, you'll see the power of describing resources as advertisements and learn how advertisements simplify the task of organizing P2P networks.

Protocols

Every data exchange relies on a protocol to dictate what data gets sent and in what order it gets sent. Even the simplest human gesture, the handshake, is built on a protocol that defines when it's appropriate to shake hands, which hand to use, and how long to shake. A *protocol* is simply this:

> A way of structuring the exchange of information between two or more parties using rules that have previously been agreed upon by all parties.

In P2P, protocols are needed to define every type of interaction that a peer can perform as part of the P2P network:

- Finding peers on the network
- Finding what services a peer provides
- Obtaining status information from a peer
- Invoking a service on a peer
- Creating, joining, and leaving peer groups
- Creating data connections to peers
- Routing messages for other peers

The organization of information into advertisements simplifies the protocols required to make P2P work. The advertisements themselves dictate the structure and representation of the data, simplifying the definition of a protocol. Rather than passing back and forth raw data, protocols simply organize the exchange of advertisements containing the required information to perform some arbitrary functionality.

Entity Naming

Most items on a P2P network need some piece of information that uniquely identifies them on the network:

- **Peers**—A peer needs an identifier that other peers can use to locate or specify it on the network. Identifying a particular peer could be necessary to allow a message to be routed through a third party to the correct peer.
- **Peer groups**—A peer needs some way to identify which peer group it would like to use to perform some action. Actions could include joining, querying, or leaving a peer group.
- **Pipes**—To permit communication, a peer needs some way of identifying a pipe that connects endpoints on the network.
- **Contents**—A piece of content needs to be uniquely identifiable to enable peers to mirror content across the network, thereby providing redundant access. Peers can then use this unique identifier to find the content on any peer.

In traditional P2P networks, some of these identifiers might have used network transport-specific details; for example, a peer could be identified by its IP address. However, using system-dependent representations is inflexible and can't provide a system of identification that is independent of the operating system or network transport. In the ideal P2P network, any device should be capable of participating, regardless of its operating system or network transport. A system-independent entity naming scheme is a requirement for a flexible P2P network.

P2P Communication

The fundamental problem in P2P is how to enable the exchange of services between networked devices. Solving this problem requires first finding answers to two important questions:

- How does a device find peers and services on a P2P network?
- How does a device in a private network participate in P2P?

The first question is important because, without the knowledge of the existence of a peer or a service on the network, there's no possibility for a device to engage that service. The second question is important to answer because many devices in a P2P network will be separated from the network at large by networking equipment designed to prevent or restrict direct connections between two devices in different internal private networks.

Finding Advertisements

Any of the basic building blocks discussed in the last section can be represented as an advertisement, and that characteristic considerably simplifies the problem of finding peers, peer groups, services, pipes, and endpoints. Instead of worrying about the specific case, such as finding a peer, you need to consider only the general problem of finding advertisements on the network.

A peer can discover an advertisement in three ways:

- No discovery
- Direct discovery
- Indirect discovery

The first technique involves no network connectivity and can be considered a passive discovery technique. The other two techniques involve connecting to the network to perform discovery and are considered active discovery techniques.

No Discovery

The easiest way for a peer to discover advertisements is to eliminate the process of discovery entirely. Instead of actively searching for advertisements on the network, a peer can rely on a cache of previously discovered advertisements to provide information on peer resources, as shown in Figure 2.1. Although this method might sound trivial, it can effectively reduce the amount of network traffic generated by the peer and allow a peer to obtain nearly instantaneous results, unlike active discovery methods.

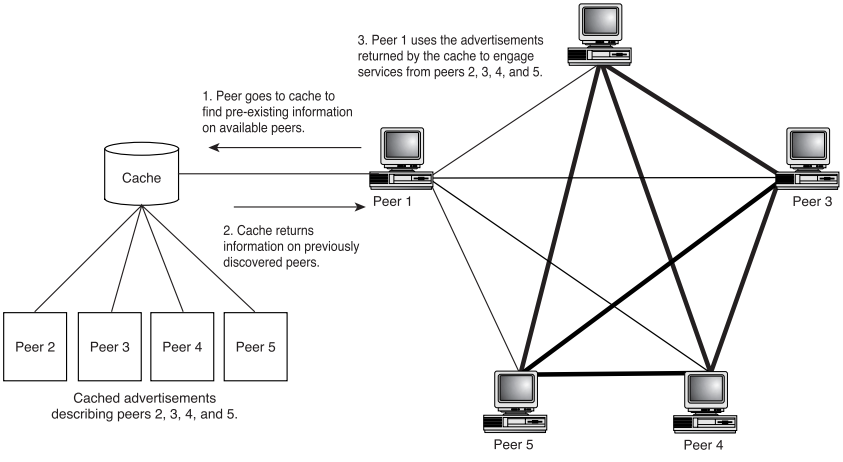

Figure 2.1 Peer discovery using cached advertisements.

In its simplest form, the local cache might consist only of a text file that lists the IP addresses and ports of previously discovered rendezvous peers, thereby providing a starting point for active peer discovery. At the other extreme, a cache might be as comprehensive as a database of every advertisement discovered by the peer in the past. The cache of advertisements might even be hard-coded into the P2P application itself, although this would limit the flexibility of the application somewhat.

A drawback of using a cache of known advertisements is the potential for advertisements in the cache to grow *stale* and describe resources that are no longer available on the network. This presents a problem when a peer attempts to engage a resource described by a stale advertisement and fails to engage the service. Although the cache has the potential to reduce network traffic, in this case, stale advertisements in the cache increase network traffic. When a peer attempts to engage a resource over the network and discovers that the resource is no longer available, the peer will probably have to resort to an active discovery method. Thus, the peer engages the network twice in this case instead of once, which would have been the case if it had used only active discovery.

To reduce the possibility that a given advertisement is stale, a cache can expire advertisements, thereby removing them from the cache based on the probability that a given advertisement is still valid.

One way to expire advertisements is to store a *best before* timestamp in the cache with each advertisement. When an advertisement is discovered, a timestamp is stored in the cache, setting the maximum lifespan of the advertisement. Before using an advertisement, the cache checks the advertisement's best before timestamp and discards the advertisement if it's no longer considered valid. Instead of waiting for an advertisement to be used, the cache might also periodically cull the store of expired advertisements to reduce storage requirements and improve responsiveness.

Another expiration technique that a cache might use is a first-in, first-out stack of advertisements with a fixed maximum size for the stack. When the cache is full, adding a new advertisement to the stack pushes out the oldest advertisement first.

Using a cache to discover advertisements is simple to implement, especially when built in conjunction with active discovery methods. In most modern programming languages, it's trivial to create code that processes an advertisement from an abstract source and to create wrappers for file and network sources. When done this way, the code is independent of the source and will operate the same regardless of whether the advertisement originated from a file cache or from another peer on the network.

Direct Discovery

Peers that exist on the same LAN might be capable of discovering each other directly without relying on an intermediate rendezvous peer to aid the discovery process. Direct discovery requires peers to use the broadcast or multicasting capabilities of their native network transport, as shown in Figure 2.2.

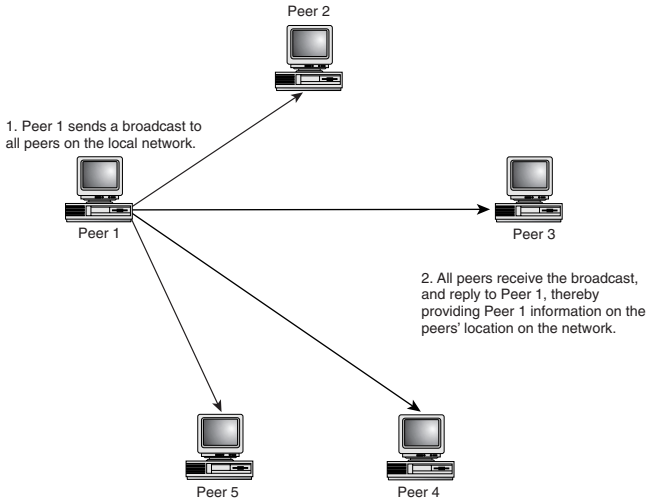

Figure 2.2 Direct peer discovery.

When other peers have been discovered using this mechanism, the peer can discover other advertisements by communicating directly with the peers, without using broadcast or multicast capabilities.

Unfortunately, this discovery technique is limited to peers located on the same local LAN segment and usually can't be used to discover peers outside the local network. Discovering peers and advertisements outside the private network requires indirect discovery conducted via a rendezvous peer.

Indirect Discovery

Indirect discovery requires using a rendezvous peer to act as a source of known peers and advertisements, and to perform discovery on a peer's behalf. This technique can be used by peers on a local LAN to find other peers without using broadcast or multicast capabilities, or by peers in a private internal network to find peers outside the internal network.

Rendezvous peers provide peers with two possible ways of locating peers and other advertisements:

- **Propagation**—A rendezvous peer passes the discovery request to other peers on the network that it knows about, including other rendezvous peers that also propagate the request to other peers.
- **Cached advertisements**—In the same manner that simple peers can use cached advertisements to reduce network traffic, a rendezvous can use cached advertisements to respond to a peer's discovery queries.

When used together as shown in Figure 2.3, propagation and caching provide an effective solution for rendezvous peers to cache a large number of advertisements and serve a large number of simple peers. As each simple or rendezvous peer responds to the discovery request, the rendezvous peer can cache the response for future use, further reducing network traffic and increasing network performance.

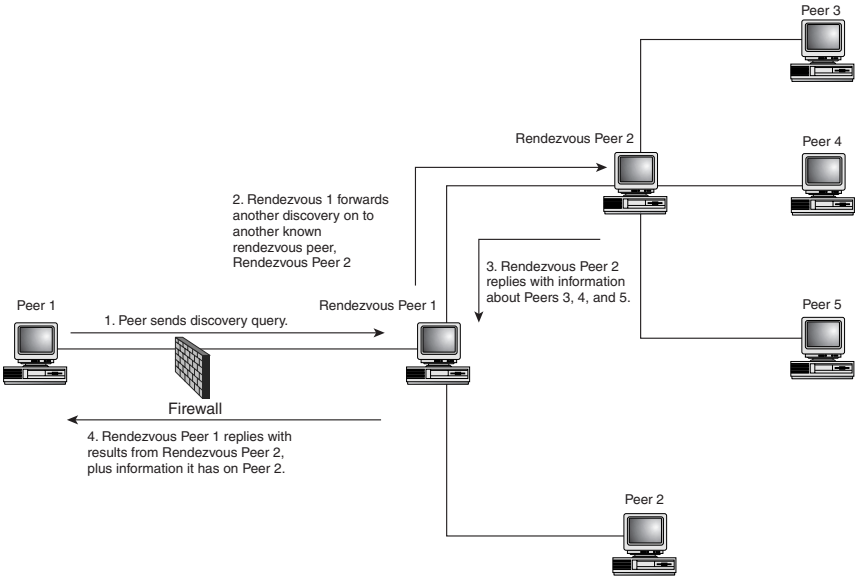

Figure 2.3 Indirect discovery via a rendezvous peer.

Although caching reduces network traffic required to discover resources, propagating discovery queries to other rendezvous peers without restriction can lead to severe network congestion on a P2P network, as shown in Figure 2.4. When one rendezvous receives a discovery query, it forwards the request to all the rendezvous peers that it knows; one query comes in, and many queries go out.

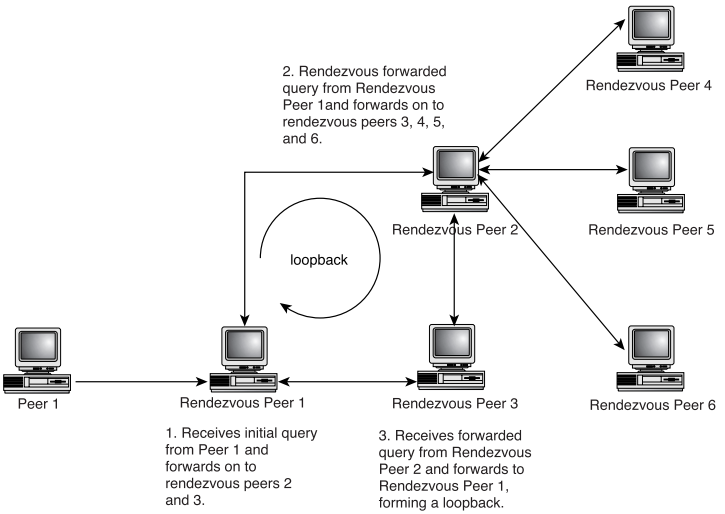

Figure 2.4 Discovery propagation chaos.

This retransmission *amplifies* the discovery query. When the query is propagated to other rendezvous peers, it is amplified again, dramatically increasing the load on the network. Adding to the problem of unchecked propagation, a discovery query's path could double back on itself, creating a feedback loop or *loopback* in the network.

To prevent excessive propagation of requests, messages usually incorporate a *Time To Live* (TTL) attribute. TTL is expressed as the maximum number of times a query should be propagated between peers on the network. As shown in Figure 2.5, when a rendezvous peer receives a message containing a discovery query, it decrements the message's TTL by 1 and discards the query if the resulting TTL value is 0. Otherwise, the query message is propagated to other peers using the new TTL value.

As a result, each message has a maximum *radius* on the network that it can travel. Of course, for this technique to work, all rendezvous peers must properly decrement the TTL field.

To address the problem of loopback, propagated messages can include path information along with the request. Rendezvous peers along the way can use this path information to prevent propagating a message to a rendezvous that has already received the message. Although this technique eliminates loopback, it doesn't prevent a rendezvous peer from getting the same message multiple times through indirect paths.

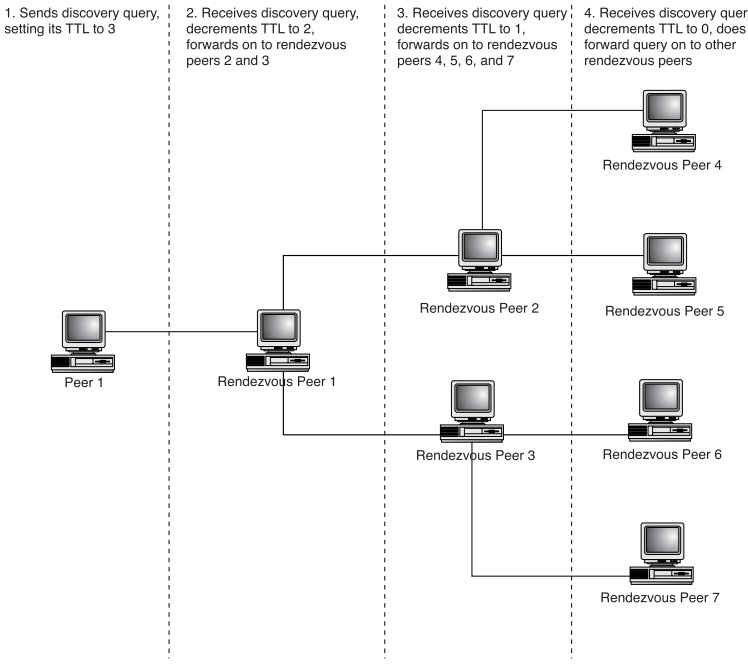

Figure 2.5 Illustration of TTL in discovery propagation.

Discovering Rendezvous and Routing Peers

For most peers existing on a private internal network, finding rendezvous and router peers is critical to participating in the P2P network. Because of the restrictions of a private network's firewall, a peer on an internal network has no capability to use direct discovery to perform discovery outside the internal network. However, a peer might still be capable of performing indirect discovery using rendezvous and router peers on the internal network.

In most P2P applications, the easiest way to ensure that a simple peer can find rendezvous and router peers is to seed the peer with a hard-coded set of rendezvous and router peers. These rendezvous and router peers usually exist at static, resolvable IP addresses and are used by a peer as an entrance point to the P2P network. A peer located behind a firewall can use these static rendezvous peers as a starting point for discovering other peers and services and can connect to other peers using the static set of router peers to traverse firewalls.

Challenges to Direct Communication

The use of firewalls and NAT by corporate private networks poses a serious obstacle to P2P networking. NAT and firewalls are usually used together to secure a corporate network against unauthorized network activity originating from either inside or outside the network and to provide a private internal networking environment.

Firewalls

Firewalls are used to protect corporate networks from unauthorized network connections, either incoming from the outside network or outgoing from the internal network, as shown in Figure 2.6. Typically firewalls use IP filtering to regulate which protocols may be used to connect from outside the firewall to the internal network or vice versa. A firewall might also regulate the ports used by outside clients to initiate inbound connections to the internal network or by internal clients to initiate outbound connections from the internal network.

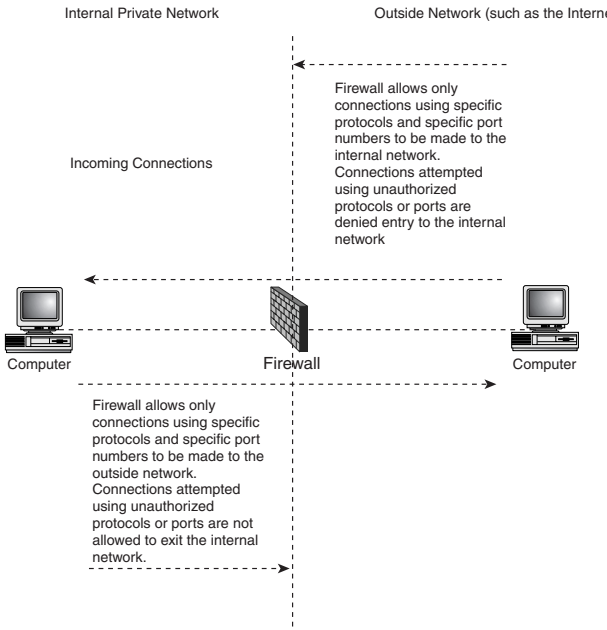

Figure 2.6 A network topology using a firewall.

Because a firewall might block incoming connections, a peer outside the firewall will most likely not be capable of connecting directly to a peer inside the firewall. A peer within the network might also be restricted to using only certain protocols (such as HTTP) to connect to locations outside the firewall, further limiting the types of P2P communication possible.

Network Address Translation (NAT)

NAT is a technique used to map a set of private IP addresses within an internal network to another set of external IP addresses on a public network. NAT comes in two varieties:

- **Static NAT**—In static NAT, the mapping relationship between internal and external IP addresses is one-to-one. Every internal IP address is mapped to one and only one external IP address.
- **Dynamic NAT**—Dynamic NAT maps the set of internal IP addresses to a smaller set of external IP addresses.

A private network employing NAT usually assigns internal IP addresses from one of the ranges of IP addresses defined specifically for private networks:

- Class A private addresses: 10.0.0.0 through 10.255.255.255
- Class B private addresses: 172.16.0.0 through 172.31.255.255
- Class C private addresses: 192.168.0.0 through 192.168.255.255

A machine using an IP address within this range is most likely behind NAT equipment.

NAT is used for a variety of reasons, the most popular reason being that it eliminates the need for global unique IP addresses for every workstation within a corporation, thereby reducing the cost of a corporate network. NAT also enables system administrators to protect a network by providing only a single point of entry into the internal network. NAT accomplishes this by allowing only incoming connections to internal machines that originally initiated a connection to the outside network. Rather than attempting to protect each machine using a firewall to filter incoming connections, a system administrator can use NAT to ensure that the only connections allowed back into the network are those that originated within the network.

NAT is usually implemented by a router or a firewall acting as a gateway to the Internet for the private internal network. To map a packet from an internal IP address to an external IP address, the router does the following:

1. Stores the source IP address and port number of the packet in the router's translation table
2. Replaces the source IP address for the packet with one of the IP addresses from the router's pool of public IP addresses, storing the mapping of the original IP address to the public IP address in the translation table in the process
3. Replaces the source port number with a new port number that it assigns and stores the mapping in the translation table

After each step has been performed, the packet is forwarded to the external network. Data packets arriving at one of the router's external public IP addresses go through an inverse mapping process that uses the router's translation table to map the external port number and IP address to an internal IP address and port number. If no matching entry for a given public IP address and port number is found in the translation table, the router blocks the data from entering the internal private network. The flow of data across a NAT router is illustrated in Figure 2.7.

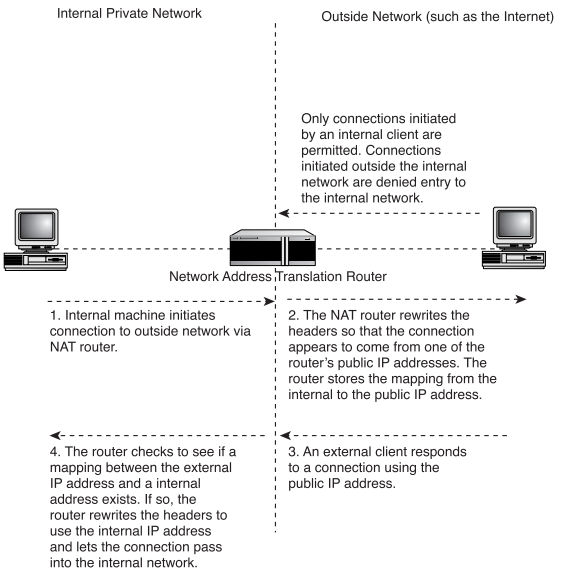

Figure 2.7 A network topology using Network Address Translation.

NAT protects networks by allowing only connections to the internal network that originated within the internal network. A machine outside the network can't connect to a machine in the internal network unless the internal machine initiated the connection to the external machine. As a result, an external peer in a P2P network has no mechanism to spontaneously connect to a peer located behind a NAT gateway. From the outside peer's point of view, the peer doesn't exist because no mapping between external and internal IP addresses and port numbers exists in the router's translation table.

Traversing the NAT/Firewall Boundary

The combined use of NAT and firewalls results in an especially difficult set of circumstances for peer communication: Peers can't connect to machines behind NAT unless the internal peer initiates communication, and connections can be blocked at the firewall based on the connection's protocol or destination IP address and port number.

The only tool that a peer has at its disposal to solve this problem is its capability to create outgoing network connections to hosts outside the firewall/NAT gateway. Peers can use protocols permitted by the firewall to tunnel connections through the firewall to the outside network. By initiating the connection within the internal network, the necessary mapping in the NAT router translation tables is set up, allowing an external machine to send data back into the internal network. However, if a firewall is configured to deny all outgoing connections, peer communication is impossible.

In most corporate networks, HTTP is the protocol most likely to be enabled by a firewall for outgoing connections. Unfortunately, HTTP is a request-response protocol: Each HTTP connection sends a request and then expects a response. The connection must remain open after the initial request to receive the response. Although HTTP provides a peer with a mechanism to send requests out of the internal network, it doesn't provide the capability for external peers to spontaneously cross the firewall boundary to connect to peers inside the internal network.

To address this problem, a peer inside a firewall uses a router peer either located outside the firewall or visible outside the firewall to traverse the firewall, as shown in Figure 2.8. Peers attempting to contact a peer behind a firewall connect to the router peer, and the peer behind the firewall periodically connects to a router peer. When the internal peer connects to the router, any incoming messages get *pushed* down to the peer in the HTTP response.

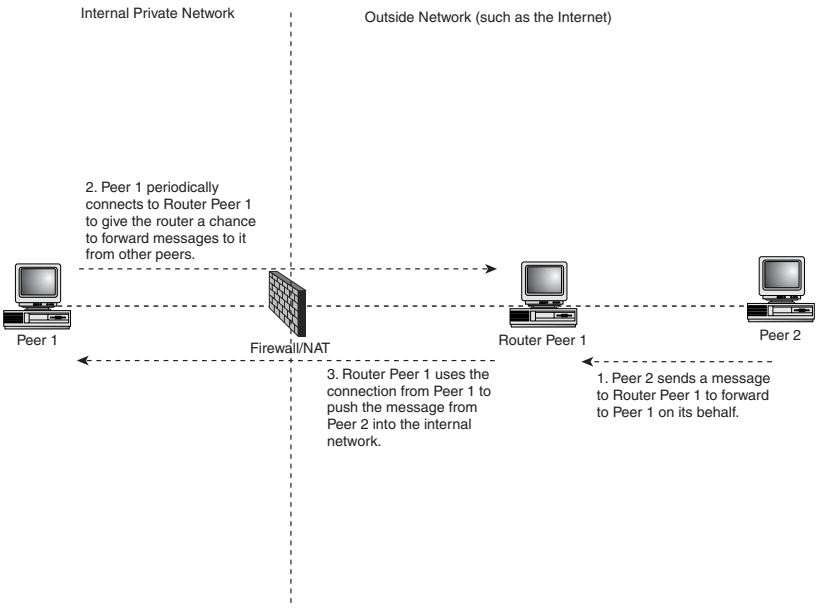

Figure 2.8 Traversing a firewall/NAT.

This technique can be used with any protocol permitted by the firewall and understood by the router peer. The router peer effectively translates between the network transport used for P2P communication and the transport used to tunnel through the firewall.

Routing Messages Between Peers

In cases when a firewall or NAT is located between two peers, a router peer must be used to proxy a connection between the public network and the peer located inside the firewall. In the simple case, only a single firewall separates the source and destination peers, thus requiring only a single router peer. In more complex cases, a firewall or NAT can protect each of the peers and require the use of multiple router peers to traverse each firewall/NAT boundary.

Single Firewall/NAT Traversal

Figure 2.9 shows the process for sending messages outside a single firewall/NAT.

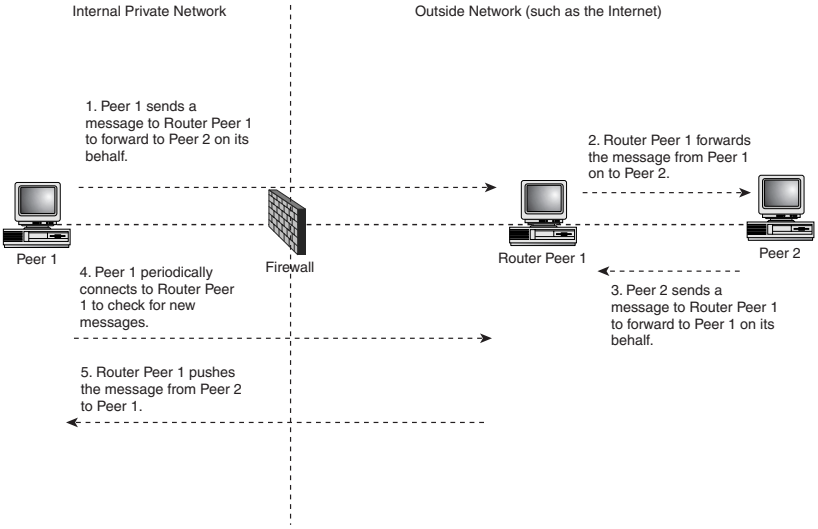

Figure 2.9 Outgoing single firewall/NAT traversal.

To allow a peer located inside a firewall/NAT to send a message to another peer located on the public network, three steps are required:

1. The peer behind the firewall/NAT connects to the router peer using a protocol capable of traversing the firewall, such as HTTP, and requests that the router peer forward a message to a destination peer.
2. The router accepts the connection from the peer behind the firewall and initiates a connection to the requested destination on the peer's behalf. This connection uses whatever network transport both the router peer and the destination peer have in common.
3. The message is sent from the source to the destination peer by the router peer, acting as a proxy for the source peer.

After the message from the source peer has been sent to the destination peer, the connection closes. Further messages can be sent by repeating the procedure, but the message might use a different router peer and, therefore, might follow a different route to the destination peer.

To allow a public peer to send a message to a peer located behind a firewall/NAT, the source peer must know routing information that describes a router peer capable of routing the message to the destination peer. Route information might have been obtained previously during discovery or might

require an additional discovery request to the P2P network. When the source peer has obtained routing information, sending the message involves three steps:

1. The source peer opens a connection to the router peer, asking it to forward the message on to the destination peer.
2. The router peer waits until the destination peer connects to it using a protocol capable of traversing the firewall, such as HTTP.
3. The destination peer connects to the router peer periodically, at which point the message is pushed down to the destination peer.

Again, when the message reaches the destination peer, the connection between the router peer and the other two peers is closed. Sending another message from the source peer requires repeating the procedure and might use a different router peer to provide connectivity to the destination peer.

Double Firewall/NAT Traversal

Most simple peers located at the *edge* of the Internet are likely to be protected by a firewall/NAT, so any message being sent from a source peer to a destination peer will need to traverse two firewall/NAT boundaries. The procedure for traversing two firewalls is similar to the single firewall traversal case and basically combines both the incoming and the outgoing cases of the single firewall traversal scenario. Figure 2.10 illustrates a double firewall/NAT traversal.

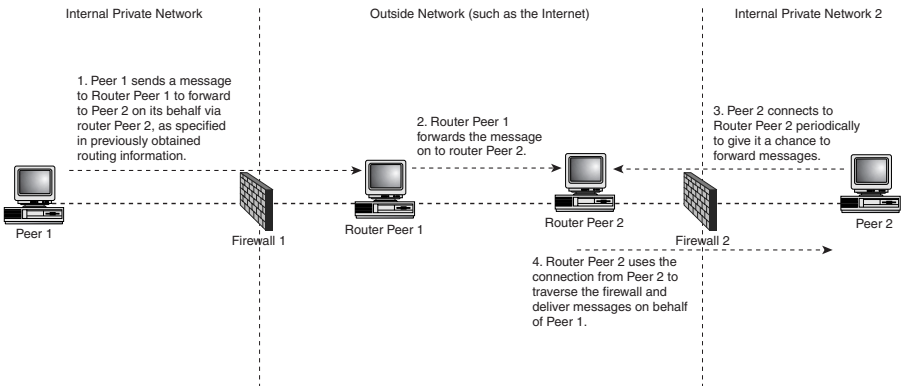

Figure 2.10 Double firewall traversal.

Before a source peer can send the message, it needs to locate routing information for the peer that describes a set of router peers capable of proxying messages to the destination peer. In this case, more than one router peer might be involved; one router peer is required to allow the source peer to traverse its firewall, and another is required to traverse the firewall providing access to the destination peer. When the source peer has this routing information, sending the message involves four steps:

1. The source peer opens a connection to the source router peer, asking it to forward the message on to the destination peer by way of the destination router peer provided.
2. The source router peer opens a connection to the destination router peer. This connection uses whatever network transport both router peers have in common.
3. The destination router peer waits until the destination peer connects to it using a protocol capable of traversing the firewall, such as HTTP.
4. The destination peer connects to the router peer periodically, and the message is pushed down to the destination peer.

Traversing both firewalls might involve only one router peer if both the source and the destination peers have a router peer in common; however, traversing firewall boundaries isn't the only reason to use a router peer. Multiple router peers can be used by a peer to circumnavigate network bottlenecks and achieve greater performance, or to provide translation between two incompatible networks transports. When the peer connects to the source router peer in this case, it provides an ordered list of router peers to use to send the message to the peer on its behalf.

Comparisons to Existing P2P Solutions

Using the building blocks of P2P networks defined in this chapter, it's possible to interpret existing proprietary P2P solutions, such as Napster and Gnutella, or even non-P2P applications, such as the client/server architecture.

Napster

Napster's hybrid P2P network, consisting of a centralized server for performing search functionality, could be modeled as a single rendezvous peer and multiple simple peers, all using TCP as a network transport. The rendezvous peer provides simple peers with the capability to locate an MP3 file advertisement consisting of filename, IP address, and port information. Simple peers use this information to connect directly and download the file from its host peer.

Napster doesn't provide a complete solution for bypassing firewalls, and it is capable of traversing only a single firewall. Each peer acts as a simple router, capable of sending content to a firewalled peer when a request is made via HTTP. Napster provides no message-routing capabilities, meaning that simple peers on the network can't act as router peers to enable other peers to perform double firewall traversal.

Gnutella

In the Gnutella network, each peer acts as a simple peer, a rendezvous peer, and a router peer, using TCP for message transport and HTTP for file transfer. Searches on the network are propagated by a peer to all its known peer neighbors, which then propagate the query to other peers. Advertisements for content on the Gnutella network consist of an IP address, a port number, an index number identifying the file on the host peer, and file details such as name and size. Gnutella peers don't provide full router peer capabilities, which means that, as with Napster, Gnutella peers are capable of traversing only a single firewall.

Client/Server

Even traditional client/server architecture can be interpreted in terms of the P2P building blocks. The client acts as a simple peer, and the server acts as a rendezvous peer capable of providing advertisements that vary according to the application. No capabilities for traversing firewalls or NAT are provided, and the network transport used varies by application.

The definitions of these basic P2P building blocks will be expanded in the coming chapters to incorporate the implementation-specific details defined by JXTA and the Java reference implementation of JXTA.

Summary

This chapter presented the basic building blocks of P2P networking and explained some of the obstacles that a P2P network must overcome. Specifically, this chapter explained the barrier to P2P communication presented by firewall/NAT routers, provided background information on how they work, and explained how P2P manages to provide connectivity to private networks protected by firewall/NAT routers.

The next chapter builds on this chapter and reveals the JXTA realization of the building blocks defined in this chapter. Using the JXTA Shell, you'll see how to experiment with these primitives directly, to better understand them before you explore the JXTA protocols.