2.6.2 Searching for Information in a P2P Community

A critical component of many P2P applications is an information index—a mapping of information to host locations. In such applications, the peers dynamically *update* and *search* the index. Since the notion of "a mapping of information to host locations" may sound a bit abstract, let's take a look at a couple of concrete examples.

- In a P2P file-sharing system, there is typically a large number of participating peers, with each peer having files to share, including MP3s, videos, images, and software. A P2P file-sharing system has an index that dynamically tracks the files that the peers are making available for sharing. For each copy of each file being shared by the community of peers, the index maintains a record that maps information about the copy (for example, if it is an MP3 song, then the title of the song, the artist, and so on) to the IP address of the peer that has the copy. The index is dynamically updated as peers come and go and as peers obtain new copies of files. For example, when a peer joins the system, it notifies the index of the files it has. When a particular user, say Alice, wants to obtain a particular file, she searches the index to locate copies of the desired file. After having located peers that have copies of the file, she can then download the file from those peers. Once she has the entire file, the index gets updated to account for Alice's new copy of the file.
- In an instant messaging application, there is an index that maps usernames to locations (IP addresses). To understand the importance of the index in this application, consider two users, BeautifulAlice and HandsomeBob, both of whom are on each other's buddy list. When HandsomeBob starts his instant messaging client on a host with IP address X, his client will notify the index that HandsomeBob is online at IP address X. Later, when BeautifulAlice starts her instant

messaging client, because HandsomeBob is on her buddy list, her client searches the index for HandsomeBob and discovers that HandsomeBob is online at IP address X. BeautifulAlice can then establish a direct TCP connection to the host at address X and begin to instant message with HandsomeBob. In addition to instant messaging, many other of today's applications use an index for presence tracking, including Internet telephony systems (see Section 2.6.3).

We briefly mention here that the BitTorrent protocol, which is solely a file distribution protocol, does not provide any functionality for indexing and searching for files.

Below we discuss three approaches for organizing and searching an index in a community of peers. For concreteness, we'll do this in the context of searching for a file in a P2P file-sharing system. But we emphasize that this discussion equally applies to searching for any kind of information in a P2P community.

Centralized Index

One of the more straightforward approaches to locating a file is to provide a **centralized index**, as was done by Napster, the first large-scale commercial deployment of a P2P file-sharing application. In this design, the index service is provided by a large server (or server farm). As shown in Figure 2.27, when a user launches the P2P file-sharing application, the application informs the index server of its IP address and of the names of the files that it is making available for sharing (for

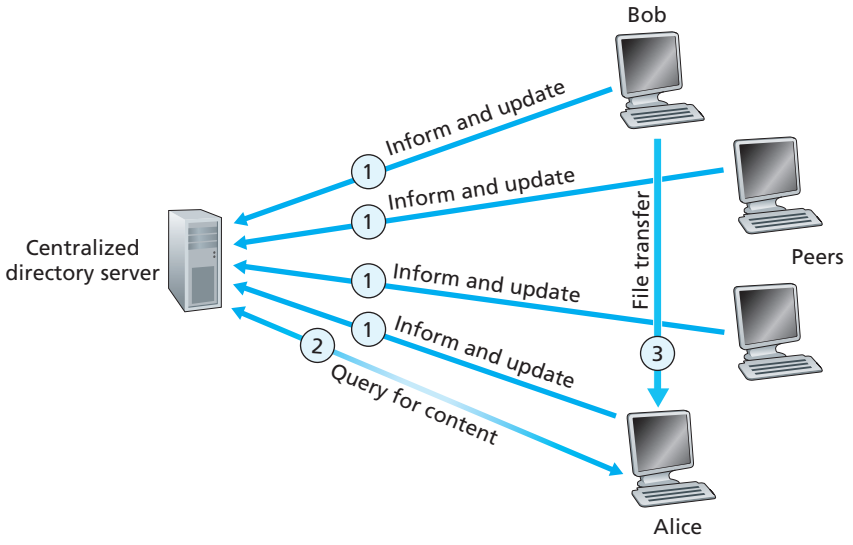

Figure 2.27 ♦ Centralized index

example, the titles of all of its stored MP3s). The index server collects this information from each peer that becomes active, thereby creating a centralized, dynamic index that maps each file copy to a set of IP addresses. Note that a P2P file-sharing system with a centralized index is really a *hybrid of P2P and client-server architectures*. The file distribution is P2P but the search is client-server. Such hybrid architectures can be found in a number of applications today, including many instant messaging applications.

Using a centralized index to locate information is conceptually straightforward, but it does have a number of drawbacks:

- *Single point of failure*. If the index server crashes, the entire application crashes. Even if a server farm with redundant servers is used, Internet connections to the server farm can fail, causing the entire application to crash.
- *Performance bottleneck and infrastructure cost*. In a large P2P system, with hundreds of thousands of connected users, a centralized server must maintain a huge index and must respond to thousands of queries per second. In fact, in 2000, when Napster was the most popular P2P file-sharing application, Napster was plagued by traffic problems at its centralized server.
- *Copyright infringement*. Although this topic is beyond the scope of this book, we briefly mention that the recording industry has been concerned (to say the least!) that P2P file-sharing systems allow users to easily obtain copyrighted content for free. (For an excellent discussion of how copyright laws bear on P2P, see [von Lohmann 2003].) When a P2P file-sharing company has a centralized index server, legal proceedings may result in the company having to shut down the index server. It is more difficult to shut down the more decentralized architectures.

Query Flooding

At the opposite end of the spectrum from centralized directories is the completely decentralized approach of query flooding. Query flooding was employed by the original incarnation of the Gnutella protocol. In query flooding, the index is fully distributed over the community of peers. Each peer indexes the files that it is making available for sharing and no other files.

In query flooding, the peers form an abstract, logical network called an **overlay network**, which is defined in graph-theoretic terms as follows. If peer X maintains a TCP connection with another peer Y, then we say there is an **edge** between X and Y. The graph consisting of all active peers and the connecting edges (ongoing TCP connections) defines the overlay network. Note that an edge is not a physical communication link; instead, an edge is an abstract link, which may consist of many underlying physical links. An edge in the overlay network may represent the TCP connection between a peer in Lithuania and a peer in Brazil, for example.

Although such an overlay network may have hundreds of thousands of participating peers, a given peer will typically be connected to a small number of other nodes (typically, fewer than 10) in the overlay network, as shown in Figure 2.28. Later we'll explain how the overlay network can be built and maintained as peers join and leave the network. For now let's assume that the overlay network is in place and focus on how a peer locates and retrieves content.

In this design, peers send messages to neighboring peers in the overlay network over the pre-existing TCP connections. When Alice wants to locate "Network Love," her client sends a query message, which includes the keywords "Network Love," to all of her neighbors. All of Alice's neighbors forward the query to all of their neighbors, and so on. This **query flooding** process is shown in Figure 2.28. When a peer receives a query, it checks to see whether the keyword matches any of the files it is making available for sharing. If there is a match, the peer sends a query-hit message to Alice containing the matched file name and file size. The query-hit message follows the reverse path of the query message, thereby using pre-existing TCP connections. In this manner, Alice discovers the peers that have a copy of the file she desires.

Although this decentralized design is simple and elegant, it is often criticized for being non-scalable. In particular, with query flooding, whenever a peer initiates a query, the query is flooded to every other peer in the entire overlay network, creating a significant amount of traffic among the peers in the underlying network

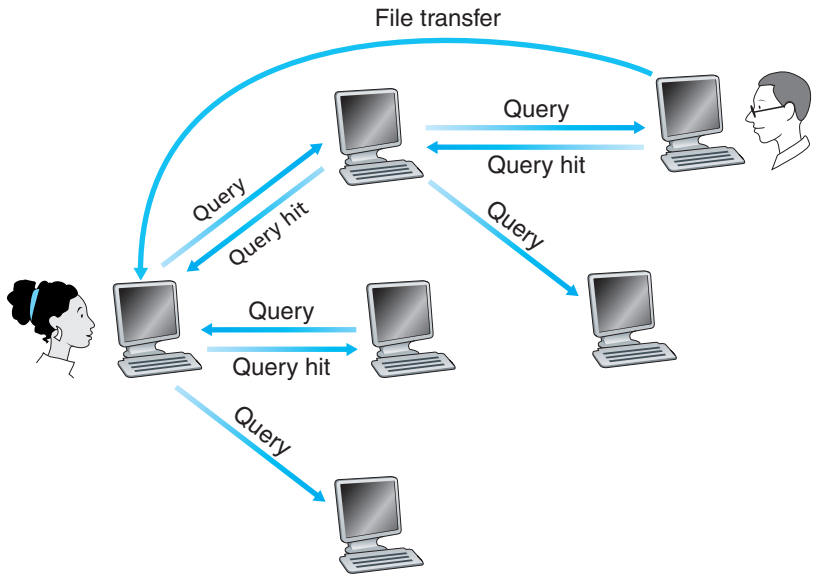

Figure 2.28 ♦ Query flooding

(e.g., the Internet) connecting the peers. The Gnutella designers responded to this problem by using **limited-scope query flooding**. Specifically, when Alice sends out her initial query message, a peer-count field in the message is set to a specific limit (say, 7). Each time the query message reaches a new peer, the peer decrements the peer-count field before forwarding the query to its overlay neighbors. When a peer receives a query with the peer-count field equal to zero, it stops forwarding the query. In this manner, flooding is localized to a region of the overlay network around the peer that initiates the limited-scope query. Clearly, this limited-scope query flooding reduces query traffic. However, it also reduces the number of peers that are queried. Thus, it is possible that a peer seeking content may not be able to locate that content, even though that content is located somewhere in the community of peers.

A fundamental issue in overlays is that of handling peer joins or departures. Using the original Gnutella design as an example, we now describe how an overlay network can be maintained as new peers join. Suppose a new peer X wants to join the overlay network.

1. Peer X must first find some other peer that is already in the overlay network. One approach to solve this **bootstrap problem** is for X to maintain a list of peers (IP addresses) that are often up in the overlay network; alternatively, X can contact a tracker site (as in BitTorrent) that maintains such a list.
2. Once X has access to such a list, X sequentially attempts to set up a TCP connection with peers on the list until one connection is created with some peer Y.
3. After the TCP connection is established between X and Y, peer X can send a "ping" message to Y. The ping message includes a peer-count field. On receiving the ping message, Y forwards it to all its neighbors in the overlay network. The peers continue to forward the ping message until the peer-count field is zero.
4. Whenever a peer Z receives a ping message, it responds by sending a "pong" message through the overlay network back to X. The pong message includes Z's IP address.
5. After X receives the pong messages, it knows the IP addresses of many peers in the overlay network. It can then set up TCP connections with some of these other peers, thereby creating multiple edges from itself into the overlay network.

We explore the actions an overlay can take upon peer departures in the homework problems.

We have now covered the essential features of query flooding and dynamic overlay construction. In summary, query flooding is a simple, distributed P2P scheme that allows a user to query for information that is located at nearby peers (where "nearby" means within a small number of hops in the overlay network). The original Gnutella design implemented query flooding as described above. Over the years, the Gnutella protocol has evolved significantly, and now

exploits the heterogeneity of the peers in a P2P file-sharing system. Gnutella remains very popular today, and is employed by the popular P2P client LimeWire.

Hierarchical Overlay

We have learned that centralized index and query flooding are two diametrically opposed approaches for locating information. We now describe a third approach, which we'll refer to as **hierarchical overlay design**, which combines the best features of these two approaches. The hierarchical overlay design was first pioneered by FastTrack, a P2P file-sharing protocol that was implemented in a number of clients over the years, including Kazaa and Morpheus. Modern Gnutella also uses a hierarchical overlay design, although it is a variant of that described here.

Like the query flooding, the hierarchical overlay design does not use a dedicated server (or server farm) for tracking and indexing files. However, unlike query flooding, not all peers are equal in a hierarchical overlay design. Specifically, peers with high-bandwidth connections into the Internet and high availabilities are designated as super peers and have greater responsibilities. As shown in Figure 2.29, if a peer is not a super peer, then it is an ordinary peer and is assigned as a child to a super peer. A super peer may have a few hundred ordinary peers as children.

A new peer first establishes a TCP connection with one of the super peers. The new peer then informs its super peer of all the files it is making available for sharing. This allows the super peer to maintain an index that includes the identity of all files its children are sharing, meta-data about these files, and the corresponding IP addresses of the children holding these files. In this way, each super peer becomes a "mini" index. But in contrast with the centralized index discussed at the beginning

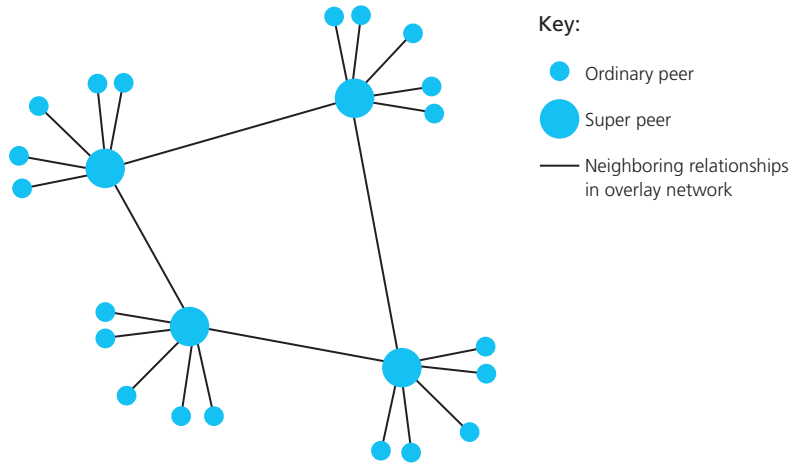

Figure 2.29 ♦ Hierarchical overlay

of this subsection, a super peer is not a dedicated server, but is instead an ordinary peer, typically residing in a residence or a university campus.

If each mini hub and its children were isolated, the amount of content available to any one peer would be severely limited. To address this limitation, super peers interconnect themselves with TCP connections, creating an overlay network among themselves. With this overlay, super peers can forward queries to their neighboring super peers. This approach is similar to query flooding, but with the limited-scope flooding taking place in the overlay network of super peers.

When a peer seeks a match for keywords, it sends a query containing the keywords to its super peer. The super peer responds with the IP addresses of its children peers that have files whose descriptors match the keywords (along with the identifiers of those files). The super peer may also forward the query to one or more other neighboring super peers. If a neighboring peer receives such a query, it also responds with the IP addresses of its children peers that have matching files. Responses from the super peers follow the reverse path in the overlay network.

This hierarchical overlay design exploits the *heterogeneity of peers* by designating a small fraction of the more powerful peers as super peers, which form the top tier of a hierarchical overlay network, as shown in Figure 2.29. As compared to limited-scope query flooding (as in the original Gnutella design), the hierarchical design allows for significantly more peers to be checked for a match, without creating an excessive quantity of query traffic [Liang 2005].

Before ending our discussion of searching for information in a P2P application, we briefly mention another important design approach, referred to as a **Distributed Hash Table (DHT)** [Stoica 2001; Rowstron 2001; Ratnasamy 2001; Zhao 2004; Maymounkov 2002; Garces-Erce 2003]. A full discussion of DHTs is beyond the scope of this book. But we mention here that a DHT (1) creates a fully decentralized index that maps file identifiers to file locations and (2) allows a user to determine *all* the locations of a file (in principle) without generating an excessive amount of search traffic. DHTs have enjoyed tremendous attention in the research community. Overnet, a central component of the popular eMule file-sharing application, employs a DHT [Liang 2006].