Chapter 2
Architecture of Peer-to-Peer Systems

Peer-to-peer (P2P) computing has been hailed as a promising technology that will reconstruct the architecture of distributed computing (or even that of the Internet). This is because it can harness various resources (including computation, storage and bandwidth) at the edge of the Internet, with lower cost of ownership, and at the same time enjoy many desirable features (e.g., scalability, autonomy, etc.). Since mid-2000, P2P computing technology has spurred increasing interests in both industrial and academic communities. As such, there are increasingly more applications being developed based on this paradigm. For example, digital content sharing (e.g., Naspter [226], Gnutella [133], and Shareaza [289]), scientific computation (e.g., BOINC [63] and Folding@home [2]), collaborative groupware (e.g., Groove [139]), instant messages (e.g., ICQ [9]) and so on. Furthermore, many research topics related to P2P computing have also been studied extensively—overlay network, routing strategies, resource location and allocation, query processing, replication, and caching. However, there has not been much effort to study the architecture of P2P systems. As the architecture of a system is the cornerstone of high-level applications that are implemented upon it, an understanding of P2P architecture is crucial to realizing its full potential. Such a study is important because: (a) It helps researchers, developers, and users to better appreciate the relationships and differences between P2P and other distributed computing paradigms (e.g., client-server and grid computing). (b) It allows us to be conscious of the potential merits of P2P computing for newly emerging application demands, and to determine the most suitable architecture for them. (c) It enables us to determine the architectural factors that are critical to a P2P system's performance, scalability, reliability, and other features. Therefore, we dedicate this chapter to summarize and examine the architecture of P2P systems and some related issues.

We first present a taxonomy of P2P architectures based on existing systems that have been developed. On one extreme, some P2P systems are supported by centralized servers. On the other extreme, pure P2P systems are completely decentralized. Between these two extremes are hybrid systems where nodes are organized into two layers: the upper tier servers and the lower tier common nodes. Second, we will conduct an extensive comparison among these three types of architectures. Third, we

Q.H. Vu et al., *Peer-to-Peer Computing*,
DOI 10.1007/978-3-642-03514-2_2, © Springer-Verlag Berlin Heidelberg 2010

will check how peers in different architectures define their neighbors (those that are directly connected)—statically or dynamically, and figure out the supporting techniques for dynamic reorganization of peers that allow communities to be formed based on some common interests among the nodes. We will also examine how nodes that are relatively powerful can be exploited to shoulder more responsibilities.

2.1 A Taxonomy

We begin by looking at a taxonomy of P2P systems. This taxonomy is derived from examining existing P2P systems. Figure 2.1 shows the taxonomy. In general, we can categorize the systems into two broad categories, *centralized* vs. *decentralized*, based on the availability of one or more servers, and to what extent the peers depend on the services provided by those servers. As expected, most of the research focuses on decentralized systems. There are essentially two main design issues to consider in decentralized systems: (a) the structure—flat (single tier) vs. hierarchical (multi-tier); and (b) the overlay topology—unstructured vs. structured. Besides these two main categories, there are also *hybrid* P2P systems that combine both centralized and decentralized architectures to leverage the advantages of both architectures. We shall examine each of these issues here.

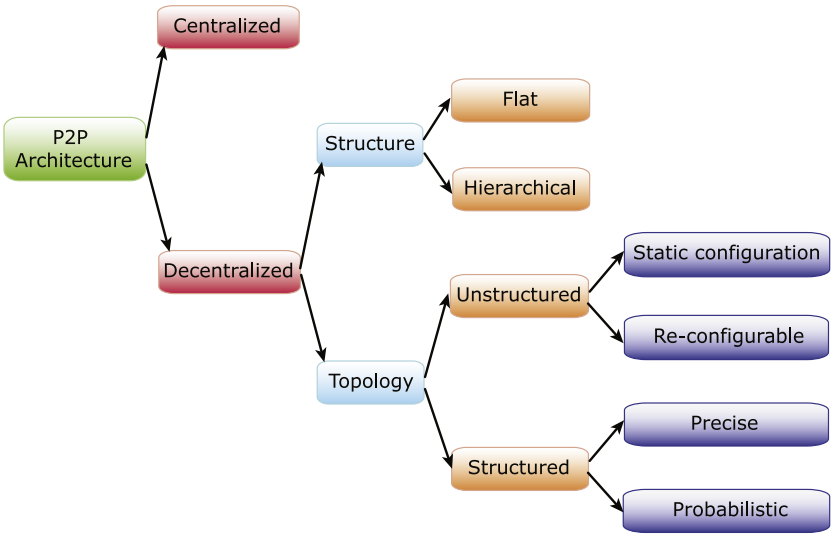


Fig. 2.1 A taxonomy of P2P systems

2.1.1 Centralized P2P Systems

Centralized P2P systems beautifully mix the features of both centralized (e.g., client-server) and decentralized architectures. Like a client-server system, there are one or more central servers, which help peers to locate their desired resources or act as task scheduler to coordinate actions among them. To locate resources, a peer sends messages to the central server to determine the addresses of peers that contain the desired resources (e.g., Napster [226]), or to fetch *work units* from the central server directly (e.g., BOINC [63]). However, like a decentralized system, once a peer has its information/data, it can communicate directly with other peers (without going through the server anymore). As in all centralized systems, this category of P2P systems are susceptible to malicious attacks and single point of failure. Moreover, the centralized server will become a bottleneck for a large number of peers, potentially degrading performance dramatically. Finally, this type of system lacks scalability and robustness. Some examples of this architecture include Napster [226] and BOINC [63].

2.1.2 Decentralized P2P Systems

In a *decentralized P2P system*, peers have equal rights and responsibilities. Each peer has only a partial view of the P2P network and offers data/services that may be relevant to only some queries/peers. As such, locating peers offering services/data quickly is a critical and challenging issue. The advantages of these systems are obvious: (a) they are immune to single point of failure, and (b) possibly enjoy high performance, scalability, robustness, and other desirable features.

As shown in Fig. 2.1, there are two dimensions in the design of decentralized P2P systems. First, the network structure can be *flat* (single-tier) or *hierarchical* (multi-tier). In a flat structure, the functionality and load are uniformly distributed among the participating nodes. It turns out that most of the existing decentralized systems are nonhierarchical. On the other hand, as noted in [126], hierarchical design naturally offers certain advantages including fault isolation and security, effective caching and bandwidth utilization, hierarchical storage and so on. In a hierarchical structure, there are essentially multiple layers of routing structures. For example, at a national level, there may be a routing structure to interconnect states; within each state, there may be another routing structure for universities within the state; and within each university, there may be yet another level that connects departments, and so on. Representatives of this category are the super-peer architecture [341] and the Crescendo system [126].

The second dimension concerns the logical network topology (the overlay network), whether it is structured or unstructured. The difference between these two designs lies in how queries are being forwarded to other nodes. In an unstructured P2P system, each peer is responsible for its own data, and keeps track of a set of neighbors that it may forward queries to. There is no strict mapping between the

identifiers of objects and those of peers. This means (a) locating data in such a system is challenging since it is difficult to precisely predict which peers maintain the queried data; (b) there is no guarantee on the completeness of answers (unless the entire network is searched), and (c) there is no guarantee on response time (except for the worst case where the entire network is searched). The famous forerunners of unstructured P2P systems are FreeNet [3] and the original Guntella [133]. The former applies unicast-based lookup mechanisms to locate expected resources, which is inefficient in terms of response time, but efficient with respect to the bandwidth consumption and the number of messages used; the latter adopts flooding-based routing strategy, which is efficient in terms of response time but inefficient in bandwidth consumption and the number of messages used (since the network is flooded with exponential number of messages). A key issue in unstructured P2P systems is the determination of the neighbors. These neighbors can be (pre-)determined statically and fixed. However, more often, neighbors are determined based on a peer's (or rather the user's) interests. Thus, as the user interests (reflected by the queries) change, the set of neighbors may change. This is based on the inherent assumption that a peer is likely to be issuing similar queries during a period of time, and nodes that have previously provided answers are likely to be contributing answers as well. Thus, keeping these nodes as neighbors can reduce the querying time (in the immediate future). We refer to the latter approach as *reconfigurable* systems, and one such system is the BestPeer system [234].

On the contrary, in a structured P2P system, data placement is under the control of certain predefined strategies (generally a distributed hash table, or simply DHT). In other words, there is a mapping between data and peers. (Very often, for security/privacy reasons, the owners have full control over their own data. Instead, it is the metadata that is being "inserted" into the P2P network, e.g., (id, ptr) pairs that indicate that object with identifier id is located at peer pointed to by ptr. However, we shall use the term *data* in our discussion to refer to both.) More importantly, these systems provide a guarantee (precise or probabilistic) on search cost. This, however, is typically at the expense of maintaining certain additional information. Employing the principle of the mapping, most of the structured P2P systems, including CAN [266], Chord [173], and Pastry [275], adopt the key-based routing (KBR) strategy to locate the desired resource. As a result, a request can be routed to the peer who maintains the desired data quickly and accurately. However, since the placement of data is tightly controlled, the cost of maintaining the structured topology is high, especially in a dynamic network environment, where peers may join and leave the network at will.

Note that there are some systems such as [199] whose overlay network is a mix between unstructured topology and structured topology. The purpose of these systems is to leverage the advantage of search in structured topology while still allowing a good degree of autonomy, and hence keeping an inexpensive maintenance cost as in unstructured topology. The basic idea of a system based on a mixed topology is that the system employs search techniques of unstructured P2P systems such as flooding for locating popular items and search techniques of structured P2P systems (*structured search*) for locating rare items. To serve the purpose of structured

search, a small part of nodes are selected to form a structured network for keeping rare items. Since only a small part of nodes forms the structured network, the maintenance cost is not high. In this system, search is executed in two steps. At first, the system performs unstructured search by broadcasting the query to neighbor nodes. If the search item is a popular item, it should be found in the first few steps, and hence the search cost is not expensive. On the other hand, if the search item is a rare item, i.e., there are not enough results returned within a predefined time or a predefined number of search steps, the system performs structured search, which should locate the item if it exists in the system. As a result, the system can alleviate the problems of search in conventional unstructured P2P systems. The problem of this system, however, is that without global knowledge, it is not easy to identify if a data item is a popular item or a rare item for indexing.

2.1.3 Hybrid P2P Systems

The main advantage of centralized P2P systems is that they are able to provide a quick and reliable resource locating. Their limitation, however, is that the scalability of the systems is affected by the use of servers. While decentralized P2P systems are better than centralized P2P systems in this aspect, they require a longer time in resource locating. As a result, hybrid P2P systems have been introduced to take advantages of both centralized and decentralized architectures. Basically, to maintain the scalability, similar to decentralized P2P systems, there are no servers in hybrid P2P systems. However, peer nodes that are more powerful than others can be selected to act as servers to serve others. These nodes are often called *super peers*. In this way, resource locating can be done by both decentralized search techniques and centralized search techniques (asking super peers), and hence the systems benefit from the search techniques of centralized P2P systems.

While it is clearly that different P2P systems belonging to different categories have different advantages and disadvantages, P2P systems in the same category also have different strengths and weaknesses depending on the specific design of the systems. This leads to the fact that different P2P systems are different in the system performance, resource location, scalability, load-balancing, autonomy, and anonymity. In the following sections, we will examine the architectural features of outstanding P2P systems from different categories in detail. Throughout most of our discussion, we shall deal with data sharing applications. However, it should be clear that the systems can also be used for other applications, e.g., sharing of storage, processing cycles and so on.

2.2 Centralized P2P Systems

Napster and SETI@home are two of the earliest and yet very popular P2P centralized-based systems. The success of Napster in digital content sharing and

SETI@home in scientific computation contributes to a proliferation of applications of centralized P2P systems in various other domains: Folding@home [2], Genome@home [130], and BOINC [63] for scientific computation; Jabber [164] in instant message; Openext [246] in digital content sharing; Net-Z [230] and Star Craft [52] in entertainment.

In general, P2P computing by definition emphasizes the equality of functions and responsibilities of all participants, which play the roles of both resource providers and resource requestors. Thus, a node can issue queries (as a client) and answers queries (as a server). Somewhat different from the equality concept, centralized P2P systems inherit some centralized features from traditional client-server architecture. Figure 2.2 illustrates the typical network structure of a centralized P2P system and how it supports data sharing applications. There is one central server in the network. (In general, there may be more than one servers. For simplicity, we restrict our discussion to just one single server.) The central server maintains metadata of files/objects shared by peers in the network. This metadata can be viewed as (objectID, peerID) pairs where objectID and peerID denote the object identifier and peer identifier, respectively. Any query is first directed to the central server that returns a list of nodes containing the desired objects. Then the query initiator communicates directly with these nodes to obtain the objects. At this phase, the central server is no longer needed.

Briefly speaking, a centralized P2P system enjoys two merits: (a) It speeds up the process of resource location, and guarantees finding all possible nodes that maintain the desired files; (b) It is easy to maintain, organize, and administer the whole system through the central server. However, the central server may become the bottleneck of the system's scalability. Even worse, it may result in a single point of failure. Therefore, improving the scalability, robustness, and security of these systems is an important research issue. In the next two subsections, we present Napster and

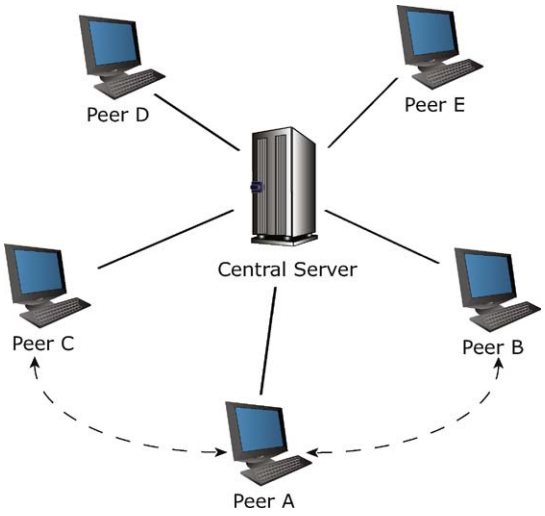


Fig. 2.2 The centralized P2P system. Peer A submits a request to the central server to acquire a list of nodes that satisfy the request. Once Peer A obtains the list (which contains Peer B and Peer C), it communicates directly with the nodes

SETI@home as representative systems to further clarify the advantages and limitations of centralized P2P systems.

2.2.1 Napster: Sharing of Digital Content

Music file sharing is perhaps one of the fastest growing applications in the Internet, and Napster [226] certainly played a critical role in facilitating music file exchange over the net. In Napster, each user (computer) acted as a producer of content. Thus, one can view the system as a collection of MP3 files that are distributed over the personal computers of Napster users. To enable users to locate music files, Napster employed a centralized server that stored the locations of the nodes that own the files. Thus, a request will be channeled to the central server to obtain the list of owner nodes. Actual data exchange between two peers can proceed without any further intervention from the central server.

Generally, Napster provided three basic functions [226]: *search engine*, *file sharing*, and *Internet relay chat*. The search engine is a dedicated server, which realizes the function of resource location. File sharing provides a mechanism to trade MP3 files among peers, without using the storage space of the central server. Internet relay chat provides a way to find and chat with other online peers. A simple MP3 trading procedure in Napster can be divided into three phases: *joining Napster*, *resource discovery*, and *downloading files*. First, through various connections (e.g., dial-up or LAN), a user was able to *join* the Napster network by connecting to the central server and completing a registration process on this central server. Second, a peer queries the central server by sending out a lookup message. After receiving the message, the central server first looked up against the index in its local repository and then returned a list of nodes that contained the desired files. Thus, *resource discovery* was accomplished with the help of the central server. Note that the central server maintained an index of metadata of shared files on all online peers and corresponding information (e.g., IP address), but it did not store MP3 files itself. Finally, the query peer established direct connections with the desired peers, and *downloaded files* without the involvement of the central server. The advantages and limitations of Napster architecture can be briefly described as follows.

- *Fault resilience, privacy, anonymity, and security.* Since Napster relies on the central server to record the information about online users and shared files, it is vulnerable to malicious attacks and a single point of failure. A query peer can easily obtain IP addresses of other peers from the central server, which may destroy the anonymity and privacy of peers. Knowing the IP addresses, a malicious user (e.g., a hacker) can attack other peers directly, or steal valuable information from them (e.g., data loss or data leakage). Therefore, potential danger exists in Napster network, and security cannot be guaranteed efficiently. How to improve fault resilience, privacy, anonymous, and security is a critical issues to Napster.
- *Scalability.* In Napster, all peers must connect to the central server and all queries must be processed by the central server firstly. Considering the server has limited capability, no matter how powerful it is. If the number of connections and

queries at one time exceed the permitted capability of the server, the response time may be beyond users' patience or denial-of-service (DOS). Therefore, Napster is weak in scalability and robustness; that is, it can only serve a limited number of connections and queries at one time.

- *Availability.* In P2P environment, we define availability as to what degree or likelihood a query peer can find the desired data on other peers. After a peer downloads MP3 files from other peers, it will keep a copy of these files within its local storage. Assume that the original peers that maintain the desired files all retreat from Napster network in a extreme condition, and these files are still available on the query peers that downloaded the files previously. Thus, the exchange of music files among peers improves data availability of Napster.
- *Decentralization, self-organization and resource location.* The degree of decentralization of Napster is low since a central server is employed to manage the operation of the system. But the searching process is quite efficient, for the mapping between resources and peers can be found directly from the central server. Self-organization refers to the ability of peers to cluster dynamically according to their own interests or optimization objectives. In Napster, peers connect first to the central server, and then interact with each other. So, it is unnecessary for Napster to do self-organization.
- *Cost of ownership.* One attractive feature of P2P system is of lower ownership cost, including the cost to maintain various resources in P2P network. In the case of client-server architecture, a powerful server is used to store sharing resources for clients to download, and provide other services to the clients. It is very expensive to maintain such a powerful server. There is also a central server in Napster. But different from client-server architecture, Napster stores resources on individual peers other than the server. Free from the burden of providing a large amount of storage spaces for resources, maintaining the central server of Napster is much cheaper than that of traditional client-server applications.
- *Efficiency and effectiveness.* The success of Napster has proved that a central control can accelerate resource location with cheaper cost and high efficiency. Also, in terms of effectiveness, peers can obtain expected files by directly exchange among themselves at will.

2.2.2 About SETI@home

Many scientific research projects require tremendous computational power. Following the conventional methodology, they are typically satisfied with a massive number of supercomputers, which of course results in substantive cost of infrastructure, administration, and maintenance. Inspiringly, armed with P2P technology, we can exploit the idle computing resources of the numerous computers (e.g., PCs) at the edge of the Internet, which is not only more economical but also more powerful than the conventional method. The most notable project in this sense is Search for Extraterrestrial Intelligence (SETI) at home (SETI@home), that aims to detect

aliens and intelligent life outside the earth. Instead of accomplishing all of tasks with high-end computers, SETI@home splits each computational task into manageable *work units* and incites each peer (i.e., a PC at the edge of the Internet who also is a participant of SETI@home) to process one work unit at one time. After finishing one unit, peers can pick up another. In this way, SETI@home has evolved into the "world's most powerful computer". For example, as reported on their website [288], SETI@home is faster than ASCI White even though the cost of building SETI@home is less than 1% of the cost of building ASCI White. Every day, peers joining SETI@home process an average of 700,000 work units, which works out to over 20 TFLOPS. This success has triggered the expansion of the project into many other areas of scientific computation. The University of California at Berkeley has then developed a general purpose distributed scientific computation project, BOINC [63] of which SETI@home is now part of.

SETI@home consists of four components: *data collector*, *data distribution server*, *screensaver (SETI@home client)*, and *user database* [247]. Data collector is an antenna that is responsible for gathering radio signals from outer space, and recording these radio signals in digital linear tapes (DLT). Data distribution server receives data from data collector, and divides data into *work unit* from two dimensions: time and frequency. Then the work units will be dispensed to personal computers that have installed screensaver. Screensaver can be downloaded freely and conveniently installed on personal computer, where the software runs in the idle CPU cycles. User database is a relational database that keeps track of SETI@home users information, for instance, tapes, work units, results, user name, and so on. Some important features of SETI@home are as follows:

- *Fault resilience.* SETI@home is also vulnerable to a single point of failure. If servers (e.g., data distribution server or data collector) are down or suffer from malicious attacks, the whole system will break down. While with respect to the validation of computation result of a work unit, a threshold is predefined, and if the computational time span exceeds the threshold, then the result of the work unit is expired. The work unit will be redispatched to another peer.
- *Privacy, anonymity and security.* Peers of SETI@home only fetch work units from data distribution server or send computation results back to user database server. They never need to know about or exchange information with other peers. Thus, SETI@home can assure the privacy and anonymity among peers.
- *Decentralization and scalability.* Similar to Napster, SETI@home has a server for dispatching work units. The bottleneck of SETI@home lies in the capability provided for maximum connections, storage spaces for work units and results. Due to limited capability of the server, SETI@home cannot obtain high decentralization and scalability.
- *Availability.* SETI@home assumes that a work unit can be completed within a given time span. If the time period for calculating a work unit exceeds the given threshold, the result of the work unit is expired, and the work unit will be reassigned to another peer. Therefore, no matter whether a peer can complete a task on time or not, SETI@home always can obtain the corresponding results of the assigned work units.

- *Cost of ownership.* SETI@home pools idle computing power of PCs together, and provides a parallel and distributed computing environment with high performance. As shown by a statistics coming from SETI@home [288], the total number of users in the system are about four millions, and the TFLOPS is about 2.8×10^{21}. By far, SETI@home arms with the most powerful computing capability in the world, but pays for a very cheap cost of ownership.
- *Efficiency and effectiveness.* As reported in [288], SETI@home is faster than ASCI White even though the cost of building SETI@home is much lower than the cost of building ASCI White. Furthermore, peers participating SETI@home can process a huge amount of work each day. Therefore, SETI@home has satisfying efficiency and effectiveness.

2.3 Fully Decentralized P2P Systems

In a P2P system of fully decentralized architecture, each peer is of equal responsibilities and rights, so that none is superior to the other. Furthermore, there are neither centralized servers nor other auxiliary mechanisms, e.g., "supernodes" to coordinate the operations among peers, including resource location, replication and caching, etc. The system can run smoothly while nodes joining or leaving the network at any time.

Several existing P2P systems belong to this category, such as the original Gnutella [133], FreeNet [3], FreeHaven [155], Chord [173], PAST [114], OceanStore [242], etc. As mentioned before, they can be further classified into two subcategories, i.e., unstructured P2P systems and structured P2P systems, based on the criterion of "the structure of overlay network".

- In *unstructured P2P systems*, the content resided on each peer has no relationship to the "structure" of the underlying overlay network. That is to say, each peer chooses the content to store at will. So given a query, it is hard to know precisely the location where the results are. The solution is to search all or a subset of the peers in the network. Usually, a lot of nodes need to be checked before the desired files are found. Most of unstructured P2P systems adopt broadcast-based query routing strategies to discover expected resources, for example, BFS-based (breadth-first search) broadcast in Gnutella. The advantage is that it can easily accommodate a highly transient node population, since the query can be spread to a large number of peers within a short time. The disadvantage is that it widely floods the query to many peers in the network no matter whether they can answer the query or not, which causes heavy network traffic of exponential query messages. Therefore, this routing strategy is efficient as far as network bandwidth consuming is concerned. Consequently, the scalability of an unstructured system is problematic. Examples of the unstructured P2P systems are the original Gnutella, FreeNet, FreeHaven.
- On the contrary, in a *structured P2P system*, there is a certain mechanism to determine the location of files in the network; that is to say, files are placed at

precisely specified locations. For example, by applying a distributed hash function (e.g., SHA-1) on both files and peers' name, files are placed on the peers whose hash values are numerically close to that of the files. Thus, a mapping is built up between files and peers. Given a query, the location of desired files can be decided quickly and deterministically, so it is unnecessary to aimlessly visit unrelated nodes to find the answers to the query. As a result, the efficiency of searching and routing can be improved greatly. Usually, peers need to maintain some data structures (e.g., distributed hash table) to guarantee the correctness and efficiency of query routing. When nodes join and leave the network very frequently, the cost to maintain the routing information is quite high. Systems such as Chord, PAST, and OceanStore belong to this category.

2.3.1 Properties

Fully decentralized P2P systems have no centralized mechanism, such as central servers and "super" nodes, to provide services to others or coordinate the operations of the systems. All participants have the same rights and obligations, and any peer can depart the network without significantly impacting the normal running of the system. Obviously, a single point of failure is avoided, since all tasks and services are distributed throughout the network and no peer is indispensable to the system. Thus, the network has strong immunity to censorship, technical failures of partial network and malicious attacks.

As there exists no central index, routing is also done in a distributed manner. Usually, peers send messages to their neighbors to indicate their requests, decide the path to use, and return feedbacks. In unstructured P2P systems, query routing is often achieved via message-based broadcasting, which involves a large number of peers in the searching process during a short time. To avoid messages flooding in the network, a Time-To-Live (TTL) counter is attached to each message. The TTL value decreases as the message is forwarded among nodes. When it turns zero, the message reaches the end of its lifetime and is no longer forwarded.

Though the broadcast strategy is easy to employ, it is inefficient with regard to the bandwidth consumption and response time. To overcome these limitations, a number of structured P2P systems have been proposed to improve the efficiency of query routing. Since there are precise mappings between the identifiers of the files and those of the peers (or locations), desired objects can be located precisely and efficiently. The cost to pay is some storage space at each peer for the routing table that contains routing information. Also it is the peer's duty to keep the routing information fresh with the change of the network. The difficulty lies in the efficient maintenance of distributed routing tables when nodes join and leave the network at a high rate.

As peers in a fully decentralized P2P system interact with each other without any central coordination, the network is entirely decentralized, self-organizing, and symmetric. In the following subsections, we study two typical fully decentralized P2P systems, Gnutella and Pastry, on various aspects of the design and functionalities.

2.3.2 Gnutella: The First "Pure" P2P System

Gnutella is a purely decentralized P2P system. No central authority is in charge of the network's organization, and there is no discrimination between the client and the server. Nodes in the system connect to each other directly through a specific software application. The Gnutella network expands as new nodes join the network and collapses as all nodes leave the network. In this sense, it is a software-based network infrastructure. Routers, switches, and hubs are not necessary to enable communication at this level.

Briefly speaking, the basic operations of Gnutella include *joining or leaving network*, *searching and downloading files*:

- *Joining or leaving network.* When a node joins the Gnutella network, it sends a "PING" message to indicate its presence. The "PING" message is forwarded to other nodes by broadcast strategy. When nodes receive the "PING" message, they feed back "PONG" messages as replying, which means they are now aware of the existence of the newcomer. From the "PONG" messages, the newcomer can get the information about those nodes and establish its own neighborhood with some of them. When a node leaves the network, it is not necessary for the node to notify its neighbors. On the contrary, each node needs to probe its neighbors with "PING" messages at regular intervals to confirm whether its neighbors are still online. If no response is returned, the node will take it for granted that these nodes have left the network, and then update the list of its neighbors.
- *Searching and downloading files.* When a node wants to find certain files, it asks its neighbors by issuing a "lookup" message, and its neighbors in turn relay the message to all of their own neighbors in the same manner. Those who have the desired files reply the query initiator with "hit" messages, which are reversely routed back along the same path as the "lookup" message has routed. Through relaying the "lookup" message via nodes' neighbors, a good recall of the searching can be achieved. The broadcasting process goes on until the entire network is covered or the TTL (time-to-live) value of the lookup message reaches zero. Now the original node may have many replies at hand, and can choose some nodes to connect and then download the desired files. Moreover, each message is attached to a unique identifier. When a node receives a message with an identifier it has seen, it will drop the message so that message loops can be avoided. Figure 2.3 illustrates the routing process of Gnutella.

2.3.2.1 Properties

Gnutella has the following properties:

- *Scalability.* The broadcasting mechanism of Gnutella is a two-edged sword. On one hand, because each query may be broadcasted to as many nodes as possible in the network, Gnutella is powerful at discovering all potential results. On the

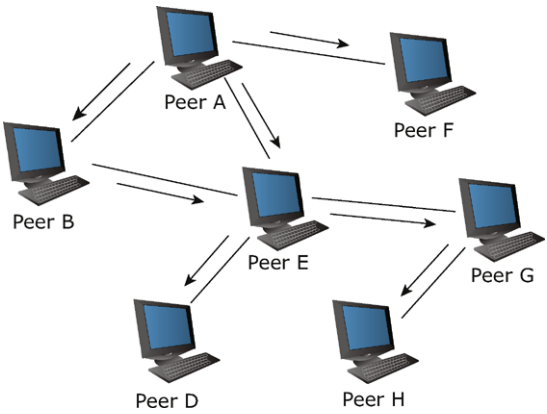


Fig. 2.3 Gnutella's search mechanism. Peer A requests for some data that Peer D and Peer H have. The query will be broadcasted to the neighbors of Peer A, and gradually, to the other peers in the whole network

other hand, as more and more nodes join the Gnutella network and the nodes issue queries continuously, the network may be congested with floods of messages. Thus, the scalability of Gnutella is problematic.

- *Self-organization.* When a node connects to the Gnutella network at the first time (or even rejoins after a departure or failure), it is just like a person entering a totally new environment. It randomly chooses a node as its entry point and stays there. As time goes, it becomes familiar with more nodes and builds connections with them. These connections are not permanent. In order to make sure that queries can be best and quickly satisfied, it is up to the node itself to decide which connections are to be established and which established connections are to be severed. Obviously, the node tends to maintain connections with those nodes who have often answered its queries and have enough bandwidth. From the viewpoint of the whole network, high-speed nodes will gradually be placed in the central part of the topology, while low-speed ones will be pushed to the edge of the topology.
- *Anonymity.* Gnutella is a system with good certain degree of anonymity. It uses a message-based broadcasting mechanism to delivery a query. The broadcast-based routing strategy is influenced by the routing tables of quite a lot of Gnutella nodes, which are dynamic and changing all the time. Therefore, it is almost impossible to figure out from which nodes a query came or to which nodes the query would go. However, the anonymity is broken when the original node chooses one or several nodes to establish direct connection and download files. At this phase, the IP addresses of the providers and the requesters are exposed to each other.
- *Availability.* Since a node can connect to and disconnect from the Gnutella network at any time without warning and because there is no mechanism to control the availability and the stability of the replies from other nodes, the availability is not guaranteed. Studies in [21, 284] have shown that only a small fraction of Gnutella nodes will be online long enough to share files with other nodes. Therefore it cannot be guaranteed that queries can be answered well, and the desired files can be downloaded successfully. When it fails, the only available solution is to retry the query or download from other peers.

2.3.3 PAST: A Structured P2P File Sharing System

PAST is a persistent peer-to-peer archival storage utility that enjoys many desirable advantages, including high performance, scalability, availability, and security. It is built on Pastry [275], a DHT-supported overlay that adopts a prefix-based routing scheme. In the PAST system, each node is assigned a 128-bit node identifier that is obtained by hashing the node's public key using a hash function such as SHA-1 [16]. Similarly, each file stored in the PAST is assigned a 160-bit file identifier that is derived from hashing the file name, the owner's public key, and a randomly chosen salt.

When a file is inserted into PAST, it is put on k nodes whose identifiers are numerically closest to the 128 most significant bits of the file identifier, among all live nodes. Given a lookup for a file, a node will forward the request to a node whose identifier has a longer prefix match with the file identifier than itself. If such a node cannot be found, the message will be forwarded to a node whose identifier shares the same length with the file identifier as the present node, but is numerically closer to the file identifier. To achieve this, each node needs to maintain a leaf set and a routing table, whose entry maps a node identifier to its corresponding IP address. Finally, the request will be reliably routed to one of the k nodes that store the file and near to the node issuing the lookup. A simple routing process is illustrated in Fig. 2.4.

2.3.3.1 Properties

PAST has the following properties:

– *Efficiency and cost of ownership.* Since PAST assigns documents to specific nodes according to some predefined rules, the locations of a file are not totally random. As a result, the routing of a request can be well directed. In a steady PAST network, all lookups can be resolved in a number of hops at most logarithmic to the total number of nodes in the system. The property is valid even when

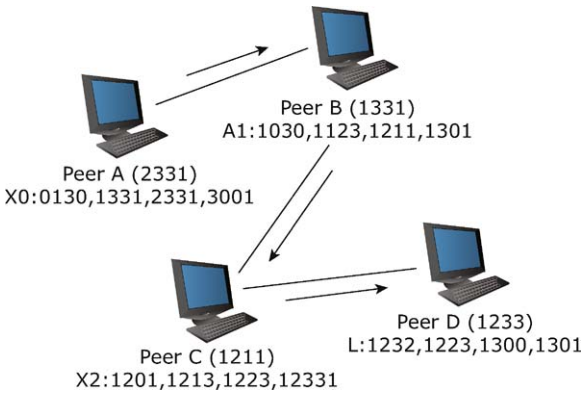


Fig. 2.4 PAST's search mechanism. Peer 2331 issues a query for a file on Peer 1233. Each time, the query is forwarded to a peer closer to the destination. Finally, it will arrive at Peer 1233

the number of nodes in the network expands dramatically. The cost of ownership is that each PAST node needs to maintain a table with $(2^b - 1) * \lceil \log_{2^b} N \rceil + 2l$ entries, where N is the number of nodes in the PAST network, while b and l are configuration parameters with typical value 4 and 32, respectively.

- *Availability and persistence.* PAST is an entirely self-organizing overlay network, and provides high availability and persistence. One key factor that may influence system performance is the dynamic nature of the network. Fortunately, the PAST network can be efficiently maintained as nodes arrive frequently or leave without notifying others. Another situation under which performance should be examined is that of extreme operating states, such as when a large fraction of the aggregate storage capacity of all nodes has been occupied. PAST presents two kinds of storage management methods, replica diversion and file diversion, to deal with this situation. With them, free storage space among nodes in PAST network can be well-balanced, and hence PAST can achieve high global storage utilization and graceful degradation when the system approaches its maximal load. Furthermore, PAST will cache copies of popular files on some nodes to minimize the hops count, maximize the query throughput, and balance the workload in the system.
- *Anonymity.* Recall that when a file is inserted into PAST, a file identifier is computed by applying the SHA-1 hash function to the file name, the owner's public key, and a random number. The public key acts as an *initially unlinkable pseudonym* [255] for users to hide their identities. It is also possible that a user uses several pseudonyms to obscure that certain operations were initiated by the same user. It is hard to break the pseudonym to reveal the identity of a user. In addition, other strong mechanisms can be layered on top of PAST to provide higher levels of anonymity.
- *Fault resilience.* In PAST, each file has several replicas on different nodes, and the probability that those nodes fail simultaneously is quite low. Through which, PAST guarantees the availability of files unless all k nodes have failed at the same time. In the case of node failures, all members in the leaf set of the failed node will be notified and updated by the first one that detects this failure. However, routing table entries that refer to failed nodes are repaired lazily: only when the failed node is on a routing path, it will be detected and be replaced by another appropriate node. Experimental results [275] show that Pastry can recover all missing table entries, and the average number of hops is only slightly higher than that before the failures.
- *Security.* In most conditions, PAST nodes and end users use *smart card* to ensure security and do quota management. The smart cards generate identities for nodes and files, and maintain their integrity. Consequently, an attacker cannot forge identifiers of nodes and files to control the identifier space or mislead file insertions. When a file is inserted into the PAST network, a *file certification* is issued and signed with the owner's private key. The file certification should be verified by the stored receipt, which prevents a malicious node from tampering the stored content or occupying extra storage quotas. Analogously, in order to reclaim a file, the file's legitimate owner must issue a *reclaim certificate*, which

is verified by a node storing the file by comparison with the file certificate stored with the file itself. The PAST routing scheme is actually randomized to prevent from repeating requests along a path being intercepted by a malicious node. Because of all these measures taken, PAST has a high security level and is resistant to attackers.

2.3.4 Canon: Turning Flat DHT into Hierarchical DHT

As stated in Sect. 2.1, the network structure can be *flat* (single-tier) or *hierarchical* (multi-tier). In a flat structure, the functionality and load are uniformly distributed among the participating nodes. It turns out that most existing decentralized systems are nonhierarchical. On the other hand, recently, Ganesan et al [126] put forward an approach to turn flat DHT into hierarchical one, which takes advantages of both flat and hierarchical structures. Concretely, hierarchical design naturally offers fault isolation and security, efficient caching and effective bandwidth utilization, hierarchical storage, proximity of physical network and so on. In what follows, we discuss the hierarchical DHT design, named as Canonical approach.

The key idea behind of the Canonical approach is that it uses recursive routing structure to construct a hierarchical DHT. For example, Fig. 2.5 shows a part of hierarchical structure of National University of Singapore. In the Canonical approach, all nodes in any domain are interconnected with each other to form a flat DHT routing structure. Notice that "nodes in domain X" refers to all nodes in the subtree rooted at X. However, different from any flat DHT structure, the Canonical approach ensures the nodes located in different sub-domains, will be merged into a new, high-level DHT structure in the current domain by adding some links from each node in one domain to some nodes in other domains. For example, there are three research groups in the SoC domain, i.e., DB, IS, and DS, and all computers of each sub-domain forms a Chord ring. Similarly, in SoC domain, all nodes of DB, IS, and DS domains form a new, high-level Chord ring. In this way, all participating nodes will form a hierarchical DHT according to the hierarchical structure of their domains.

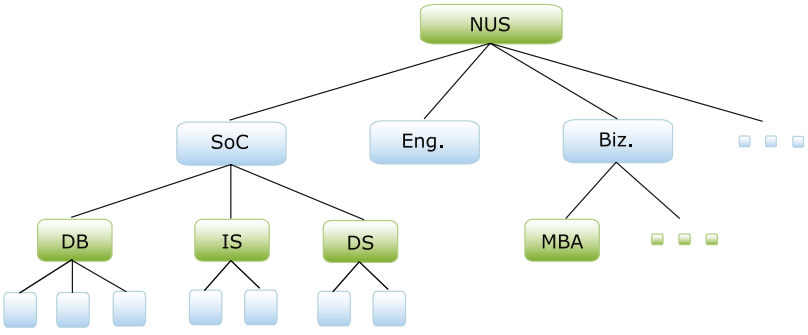


Fig. 2.5 A hierarchical structure of National University of Singapore

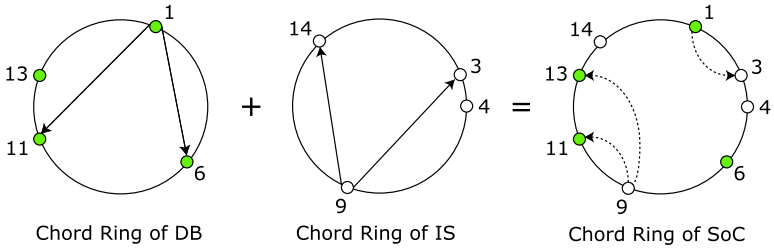


Fig. 2.6 Merging Chord rings of DB and IS into Chord ring of SoC

Obviously, the challenge of the Canonical approach is to design a merging operation in such a manner that the total number of links per node remains the same as in a flat DHT, and that global routing between any two nodes can still be achieved as efficiently as in the flat design. In what follows, we use the Chord protocol [173] as an example to illustrate how to design such a hierarchical DHT (please refer to Sect. 3.3.1 for description of Chord). Specifically, nodes of Chord rings being merged together retain all their original neighborhood. At the same time, they may create some additional links from each node n in its Chord ring to other nodes n' in their Chord rings if and only if (1) for some $0 \leq k < m$ (m is the size of the namespace), node n' is the closest node that is at least distance 2^k away; and (2) n' is closer to n than any other node in n's ring. Condition (1) is indeed the standard Chord rule for establishing neighborhood with other remote nodes, which is used for uniting nodes of different Chord rings. On the other side, condition (2) emphasizes on the fact that each node in one ring need only establish a subset of these links (defined by Chord protocol). That is, only the links to nodes that are closer to it than any other nodes in its own ring.

Figure 2.6 depicts how two Chord rings are merged together into a new Chord ring. Suppose that there are two Chord rings of DB and IS, each with four nodes and its namespace m equals to 4. Let's focus on node 1 of DB ring and node 9 of IS ring. Recall that node 1 builds neighborhood with node 6 and node 11 in DB ring (for each $0 \leq k < 4$, the closest node that is at least distance 2^k away, and hence, node 6 is the closest node corresponding to distance 1, 2, and 4, and node 11 is the closest node corresponding to distance 8). Similarly, in IS ring, node 9's neighbors are node 14 and node 3.

When we merge two Chord rings of DB and IS together, a few new links must be created between nodes of the two Chord rings. Returning to the example above, let's consider the links to be created by node 1 in DB ring. According to condition (1), node 1 will build neighborhood with node 3 (for distance 1 and 2), and with node 9 (for distance 8). However, according to condition (2), node 9 should be ruled out since it is further away than the closest node (node 6) in the DB ring. As such, node 1 only establishes neighboring relationship with node 3. Similarly, node 9 in IS ring should build links with node 11 (for distance 1 and 2) and node 13 (for distance 4), while not with node 1 (for distance 8).

To route messages in such a hierarchical DHT, the Chord protocol is employed. In general, greedy clockwise routing will forward the message to the closest pre-

decessor p of the destination at each level, and p would then be responsible for switching to the next level of Canonical ring and continue routing on that ring. For example, in Fig. 2.6, if node 3 looks for the node with key 13, it first routes the query to node 9 in DB ring. Then node 9 will route the query in the merged ring till node 13 is found. The maintenance of node joining and leaving is similar to the Chord protocol (if interesting, reader can refer to [126] for details).

2.3.4.1 Properties

Hierarchical DHT has the following properties:

- *Efficiency and cost of ownership.* From above, we observe that the cost of ownership of each node in the Canonical ring is $O(\log N)$ irrespective of the structure of the hierarchy. Thus, the efficiency of the Canonical structure is the same as the original one.
- *Fault isolation.* Like DNS system, such a hierarchical DHT can isolate fault efficiently. This is because all nodes in a domain have their own flat DHT. As such, any fault occurring within certain domain would not affect nodes outside the domain.
- *Proximity of physical network.* The likelihood of nodes within a domain being physically close to each other is very high. Hence, such a hierarchical DHT naturally adapts to the proximity of physical network.
- *Hierarchical storage and retrieval.* Compared with the flat DHT, the hierarchical design of a DHT offers more alternatives for content storage. When a node inserts content to the network, it can use domain information to store the content into a specific domain. As a result, the retrieval can be limited within a particular domain and other nodes outside this domain never need to be accessed.

2.3.5 Skip Graph: A Probabilistic-Based Structured Overlay

From the perspective of the *network topology*, there are two main ways in which P2P systems are structured: using DHT and using skip-lists. Distributed Hash Table (DHT) provides a basis for distributing data objects (or just indices) as evenly as possible over nodes in the underlying node space. Well-known systems in this class include Chord [173], CAN [266], Pastry [275], and Tapestry [349]. Though DHT-based systems can guarantee data availability and search efficiency on exact key lookup, they cannot support complex queries (e.g., *nearest neighbor* or *range* queries) as hashing destroys data locality. To solve this problem, Aspnes et al. [31] have proposed a novel structured P2P overlay named *Skip Graph*, which preserves data locality and has the potential capability of supporting complex queries. Typical P2P systems based on Skip Graph are SkipIndex [348] and SkipNet [154].

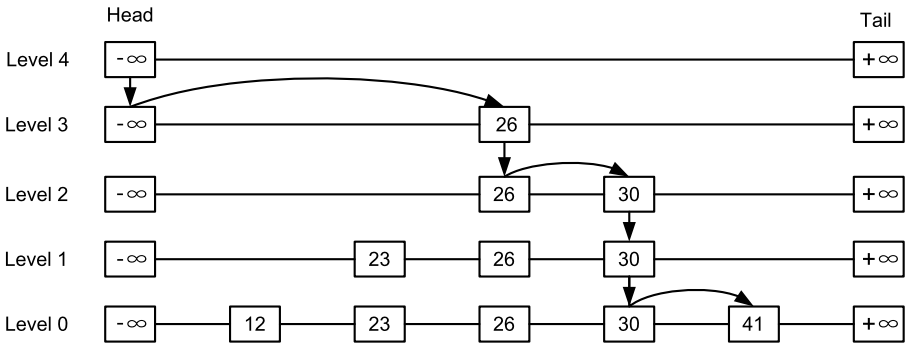


Fig. 2.7 A Skip List with 5 nodes. The *arrow lines* show the process of search node 41 from head, through node 26 and node 30, to node 41

Before presenting the general ideas of Skip Graphs, let us first review the Skip List. A Skip List [260] is a randomized balanced search tree where nodes are organized to a set of sorted linked lists, each of which corresponds to a level of the tree. The list at the lowest level of the tree (level 0) contains all nodes in the Skip List sorted increasingly by the nodes' keys. The list at level $l > 0$ contains a subset of nodes in the list at level $l - 1$ in which each node in the list at level $l - 1$ appears in the list at level l with a fixed probability p and independently on other nodes. In this way, the density of nodes in lists decreases with the increasing of the lists' level, i.e., high level lists are more sparse than low level lists. The purpose of high level links is to provide a jump over a large number of nodes in query processing. In particular, to search the node with a specified key, the nodes in the top level are first traversed, while not overshoot the node compared with the search key. Then the search repeatedly drops down to the lower levels till the desired node is found. Figure 2.7 shows a Skip List with 5 nodes and the process of search node with key 41.

A Skip Graph consists of a set of Skip Lists. Because a fixed probability p is used to determine which level a node will belong to, we thus classify Skip Graphs into *probabilistic-based* structured P2P overlay. Like the Skip List, in the lowest level of a Skip Graph, all nodes are also organized as a linked list in increasing order with respect to the identifier or key of each node. However, unlike the Skip List, in a Skip Graph each node belongs to *several* Skip Lists and the list a node x participates in is determined by its *membership vector* $m(x)$. That is, at each level $i > 0$, there are many linked lists and each node takes part in one of these lists. The highest level of a Skip Graph is the level where each node belongs to a separate list. In other words, the number of lists in the highest level of a Skip Graph is equal to the number of nodes in the Skip Graph.

To build a P2P system based on the skip graph, each resource is assigned to a node and all nodes are ordered according to their resource key. Figure 2.8 shows a skip graph with 7 nodes. Now, we give a formal description of membership vector to determine, for each level $i > 0$, which list a specific node should belong to. Suppose that membership vector $m(x)$ is an infinite random word over some fixed alphabet. Then the membership vector of any list is defined by some finite word w, and any

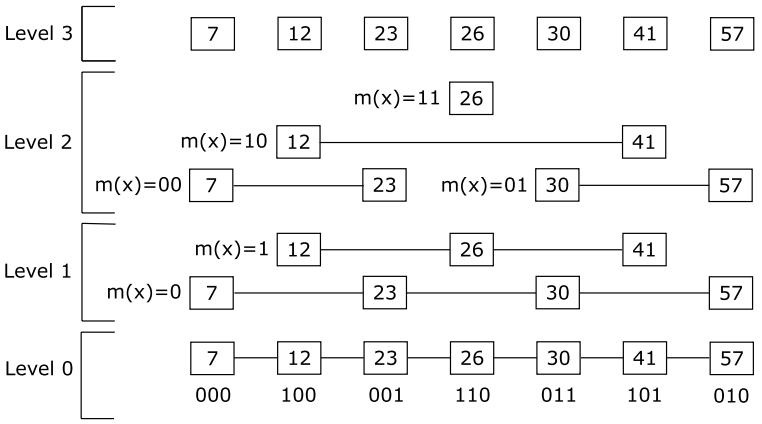


Fig. 2.8 A skip graph with 7 nodes, where level 0, 1, 2, and 3 contain 1 list, 2 lists, 4 lists, and 7 lists, respectively. For example, in level 1, there are 2 lists and their membership vectors are $m(x) = 0$ (node 7, node 23, node 30, and node 57) and $m(x) = 1$ (node 12, node 26, and node 41)

node x in the list labeled by w if and only if w is a prefix of $m(x)$. That is, at i^{th} level, any node in a list has the same prefix of its word w and the length of the prefix equals to i. For example, in Fig. 2.8, level 1 includes two lists $m(x) = 0$ and $m(x) = 1$. In the list $m(x) = 0$, node 7, node 23, node 30, and node 57 have the same prefix "0" whose length is 1, while in the list $m(x) = 1$, node 12, node 26, and node 41 have the same prefix "1".

Since insertions and deletions in a Skip Graph can be supported in the same way as search operations, i.e., locating the node holding a particular key to insert or delete data, we only introduce how to locate the node with a particular key (to see details how insertion and deletions are executed please refer to [31]). Like search in a Skip List, the search begins at the highest level of the node issuing the query. At any step in the search process, it travels along the same level without overshooting the key. In cases the search cannot go further in a level, it travels to the next lower level until it reaches level 0. Since nodes are ordered according to their key, Skip Graphs can support *range queries* that retrieve all nodes whose key is between x and y. To this end, the query node needs only to search the first node whose key is less than or equal to x and then traverses all nodes in level 0 sequentially, till any node whose key is greater than y.

2.3.5.1 Properties

Skip Graph has the following properties:

– *Efficiency and cost of ownership.* As a kind of distributed search tree structure, Skip Graphs guarantees to find all nodes with desired answers in $O(\log N)$ steps using $O(\log N)$ messages, where N is the number of nodes in the system. Similarly, inserting a node in a Skip Graph is expected to incur $O(\log N)$ messages

in $O(\log N)$ time. The cost of ownership of Skip Graphs is measured by the number of neighboring nodes and with high probability, is $\Theta(\log N)$.
- *Locality preservation.* Since no hash function is used, similar resources are stored at adjacent nodes in a Skip Graph. As such, resource locality will be preserved. This property benefits some applications such as pre-fetching of web pages, enhanced browsing and efficient searching.
- *Support of complex queries.* Preserving data locality in a Skip Graph is especially useful for database communities to design the novel P2P system that can support range queries, i.e., locating resources whose keys fall into a specified range of values. Further, based on the range search, nearest neighbor queries can also be implemented. Indeed, many subsequent refinement and proposals have used Skip Graph as part of their design.
- *Fault tolerance.* Since each level of a Skip Graph contains several linked lists, the chance that any individual node takes part in a search is not big. Thus, neither single points of failure nor hot spots should exist. Furthermore, all nodes in a Skip Graph are still interconnected even under the circumstances of removal of a large number of nodes selected at random. In particular, if we randomly select an $O(1/\log N)$ fraction of the nodes in a Skip Graph to remove, most of remaining nodes still interconnect.
- *Scalability.* In DHT-based P2P systems, it is necessary to know the size of the system to determine the namespace of nodes. On the contrary, a Skip Graph does not need to have this constraint. As a result, a Skip Graph can be inflate or deflate at will with respect to the number of nodes in the network.

2.4 Hybrid P2P Systems

Hybrid P2P systems draw advantages from the other types of P2P architectures, i.e., centralized and fully distributed ones, while distinguish themselves from the other two types by their elegant auxiliary mechanisms that facilitate resource location. In some P2P systems of hybrid architecture, there are some peers possessing much more powerful capabilities and having more responsibilities than other peers, which are usually referred to as "super" peers (or supernodes). These supernodes form an "upper level" of a hybrid system, which provides similar services for the ordinary peers as the central server does in a centralized P2P system. The common peers, on the other hand, can enjoy much more services from the supernodes in the "upper" layer, especially in the process of resource location. Though supernodes share some similar features to the central server in centralized P2P systems, it is easy to distinguish one from the other based on the following metrics: (i) A supernode is not as powerful as the central server of a centralized P2P system, and it is only in charge of a subset of peers in the network. (ii) A server as in Naspter, just helps peers to locate desired files without sharing any file by itself; however, a supernode has to not only coordinate the operations among the peers under its supervision, but also perform the same operations by itself and contribute its own resources as the common peers do. Interestingly, the determination of a supernode and its connections to

other peers are very similar to contacts between persons in human society. For example, in human society, some sociable persons always keep more knowledge and connections than the common persons (e.g., professors, mentors). If one person has some problem, he or she can seek the help of these "mentors". The probability for these latter individuals to settle the matter is greater than that of the average persons.

In some other hybrid P2P systems, there exist some components as their "upper" level. For example, BestPeer [234] has a relatively small number of location independent global names lookup servers (LIGLOs), which serve as the "upper" level of the system. Obviously, the LIGLOs are also distinct from the central serves in centralized P2P systems. Since a LIGLO just generates a unique identifier for the peers under its management, it helps common peers to recognize their neighbors in spite of their dynamic IP, and facilitates peers to dynamically reconfigure their neighbors based on certain metrics (e.g., MaxCount and MinHops). However, a LIGLO is never involved in the resource location of a peer. In such a system, when a peer joins the network for the first time, it can randomly choose a set of nodes as its neighbors and issue queries for desirable information or answer queries from other nodes. With the feature of self-reconfiguration, each peer in a hybrid P2P system manages to directly connect to those that can potentially benefit its later queries. Note that any node can also be chosen as neighbor by other peers. As time passes, queries can probably be answered more efficiently and more precisely due to dynamic reconfiguration. In summary, hybrid P2P systems have many advantages, such as optimizing network topology, improving response time and saving system resource consumption, and avoiding a single point of failure as well. Hence, there are plenty of research focusing on hybrid P2P systems and corresponding applications of such P2P systems in real life, including current Gnutella, BestPeer, PeerDB, PeerIS, CQBuddy.

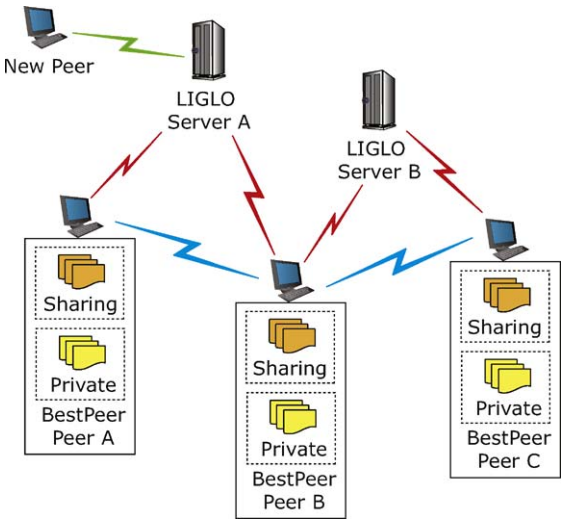
In the following sections, we introduce a self-configurable P2P system—BestPeer, which exhibits essential features of hybrid P2P systems. Furthermore, BestPeer combines mobile agents and P2P technology into a unified framework gracefully. From the discussions on BestPeer, we can obtain a good understanding of the desirable features of hybrid P2P systems.

2.4.1 BestPeer: A Self-Configurable P2P System

BestPeer is designed as a generic platform to develop P2P applications. Compared to other P2P systems, BestPeer has four distinct features:

1. BestPeer employs mobile agent technology. The system uses mobile agents that contain executive instructions to allow peers to execute operations locally. In this way, raw data can be processed directly at its owner node, and hence the system utilizes network bandwidth efficiently. Furthermore, since agents can be customized, new applications can be extended on BestPeer easily.
2. BestPeer allows peers in the system to share not only data but also computational resources. It is because mobile agent technology can allow a peer to process a request on behalf of another peer.

Fig. 2.9 BestPeer network



3. BestPeer uses a dynamic method that allows a peer to keep peers having a high potential of answering its queries nearby, and hence the system can reduce the query response time. This feature is actually similar to human behavior.
4. BestPeer introduces a concept of location independent global names lookup (LIGLO). The system uses LIGLO servers to identify peers independently of their IP address. In this way, even though a peer can change its IP address each time it joins the system, the system still recognizes it as a unique peer.

There are two types of nodes in a BestPeer system: peers and LIGLO servers, and a majority of node in the system are peers. As in Fig. 2.9, which shows an example of a BestPeer system, each peer participating the system can share a portion of its data and a peer can only access the sharable data of other peers. A typical procedure of a node joining BestPeer network, accessing sharable resources of other nodes, and rejoining BestPeer network is as follows:

– Joining BestPeer system. For the first time when a peer joins the system, it needs to register itself with a LIGLO server. When a LIGLO server receives a registration request from a peer, it creates a global and unique identifier BPID (BestPeer ID) for that peer. This BPID is then returned to the new node. Additionally, the LIGLO server also returns to the new node a list of BPID and IP address of current online peers registered to the server. The new peer can communicate with peers in this list directly without going through LIGLO servers. However, since a peer can leave the system anytime without notifying the server, the IP address of a peer may be incorrect. In this case, the peer simply removes the unreachable peer from the list. To avoid this case, LIGLO servers often check IP address of their registered peers and discard offline peers from the list. Note that a peer can register to multiple LIGLO servers.
– Accessing sharable resources. When a peer is in the system, it can query and access shared data from other peers. The basic idea of query processing in BestPeer

is similar to that in Gnutella. In particular, when a node wants to issue a query, it simply broadcasts the query to its neighbor peers, which in turn forward the query to their neighbor peers, and so on. When a peer receives a query request, if it contains the queried data, it returns the result directly to the query initiator.
- Rejoining BestPeer network. When a node rejoins the system, i.e., it is not the first time the node joins the system, the node sends its IP address together with its BPID to the LIGLO server it has been registered before. If this IP address is different from the previous registered IP address, the LIGLO server updates the new IP address for the node.

As discussed above, BestPeer can be distinguished from other P2P systems based on four main features. In the following part, we analyze in detail the effect of these features on the performance aspects of BestPeer such as fault resilience, security, anonymity, scalability and so on.

- *Fault resilience.* The use of LIGLO servers helps BestPeer to avoid a single point of failure phenomenon in centralized P2P systems. The main purpose of LIGLO servers is to provide peer registration and auxiliary mechanism for recognizing rejoining nodes. Thus, if a node finds its registered LIGLO server is down, it can still exchange sharable data with others through his neighbors. Additionally, the failure of a LIGLO server does not affect other LIGLO servers or peers registered to these servers. This is essentially different from the centralized P2P architecture, e.g., in Napster a failure at the central server will disrupt all communication between peers. As a result, BestPeer is immune to a single point of failure and has high fault-resilience.
- *Security and trust.* Thanks to combining agent-based technology with P2P computing technology, BestPeer can transform security and trust problems of information exchanging between peers into a secure agent-based routing issue. Pang et al. [251] have discussed two security agent-based routing approaches. One is a parallel dispatch model, the other is a serial dispatch model. In the former model, the route of a mobile agent is predefined, encrypted, and signed at the first step. After that, the agent is dispatched to each new peer to collect information. While in the later model, the peer with which a mobile agent needs to communicate is determined independently by the agent itself and information is collected dynamically when the agent visits peers. For each model, different attack types are considered and the corresponding solutions are discussed. As far as typical attack is concerned, mobile agents can work on behalf of their owner more autonomously, and system scalability, security and performance as a whole can be improved greatly.
- *Scalability.* By use of mobile agent technology, scalability of BestPeer is better than that of the centralized P2P system. Since mobile agents can execute operations locally at peers, agents can be customized for different purposes. As a result, several applications can be deployed on BestPeer easily. Furthermore, an application can provide different functions. For example, an agent can be customized to search files based on file names, another agent can be customized to search files based on file content. On the other hand, BestPeer can easily add a

new LIGLO server or remove an existing LIGLO server without affecting the existing system environments.

- *Self-reconfiguration.* BestPeer provides a mechanism that lets each peer keep some most promising peers as close as possible without additional information exchange. In detail, BestPeer employs two approaches: MaxCount and MinHops. The former guarantees that a peer with such neighbor deployment strategy can obtain maximum number of objects from its direct neighbors. In other words, a peer tends to choose those peers that may contain maximum number of potential answers of its queries. While the latter, MinHops, connects peers in a way that minimizes the number of hops for query processing. This indicates how far the query answering peers from the request peer. Through the two strategies, any node in BestPeer network can reconfigure his neighbors in a dynamic manner, which will reduce bandwidth cost for broadcasting queries and return desired answers as quickly as possible. Note that the self-reconfiguration mechanism is irrelevant to the presence of LIGLO servers. Thus, in contrast to static peers network that a peer's neighbors will not change automatically during runtime, BestPeer can adjust peers' neighbors automatically to make good use of existing bandwidth efficiently.
- *Novel applications and extensibility.* Like any infrastructure, it is important for a P2P platform to be able to support a variety of new applications effectively and efficiently. By far, five prototype systems have been developed on BestPeer to enhance peer data management (PeerDB [235]), information retrieval (PeerIS [196]), continuous query processing (CQBuddy [236]), Web caching (BuddyWeb [330]) and OLAP application (PeerOLAP [169]). PeerDB provides relational data management in P2P environment, while PeerIS combines information retrieval technology with P2P framework to offer high-efficient message routing and resource location. In order to process online analysis processing queries, PeerOLAP, a distributed cache system, is established to facilitate similar queries from different nodes nearby each other. BuddyWeb aims at improving the effectiveness and efficiency of Web searching through applying data caching technology in P2P network, and CQBuddy copes with distributed continuous queries processing in P2P environment. Moreover, to further improve security of BestPeer platform and guarantee privacy and anonymity during data exchange between peers, high security routing issue is studied in agent-based P2P system.
- Efficiency and effectiveness. The former refers to the system performance (e.g., response time), while the latter deals with the quality of the answers(e.g., relevant degree). For example, a request peer can receive answers from other peers quickly after initiating query message, which means a good efficiency. On the other hand, the number of answers may be very few and some of them are irrelevant to query, which means poor effectiveness. Because BestPeer employs MaxCount and MinHops strategies to choose peers' neighbors, Ng et al. [234] have proved that any peer in BestPeer network can obtain a better quantity and quality of answers than Gnutella and traditional client-server architecture.

Notwithstanding, being a system that supports agents, BestPeer has to provide the environment for agents to operate on. While agents bring with them their own

definitions and actions such as a new query processing strategy, they also bring problems to the operating environment. As a result, agents increase the complexity of the system, and hence they are not supported in BestPeer 2.0.[1]

2.5 Summary

From the birth of Napster to the current prolific deployment of P2P-based applications, great efforts have been made to address various specific issues of P2P computing. In this chapter, we presented a summary of architectural issues of P2P systems, such that researchers, developers, and users are able to see clearly the potential merits of different P2P systems, identify the key architectural factors that decide the system performance, and make appropriate implementation decisions.

In terms of the degree of decentralization, the architectures of P2P systems can be generally classified into three categories: *centralized P2P systems*, *decentralized P2P systems*, and *hybrid P2P systems*. On one extreme, the centralized P2P systems are supported by one or more centralized servers, where key operations are managed by the servers. On the other extreme, fully distributed P2P systems are completely decentralized. Between these two extremes are hybrid systems where nodes are organized into two layers. The upper layer, such as "super" nodes or other distributed mechanisms (e.g., LIGLO), provide services for the lower layer nodes. The features that distinguish one category from the others are summarized as follows:

- *Centralized P2P systems* inherit centralized features from the client-server architecture, which are composed of one or more central servers and a great number of "clients" (peers). The major point distinguishing it from the client-server model lies in the fact that these P2P servers do not perform sharing operations by themselves or even contribute to the sharing resources. Specifically, in the application of data sharing, the servers never store sharable files and are not involved in the file trading between peers. They only manage resource location by building up metadata index of sharable files. When a peer searches a file, its query is first sent to the central server, which in turn replies to the query initiator with the location of peers that contain desired files. At last, the requestor directly interacts with the answer contributor, without the involvement of the central server. Due to the limited capability of the central server, this type of P2P systems lacks scalability and is vulnerable to malicious attacks and a single point of failure. The most important contribution of centralized P2P systems, such as Napster and SETI@home, is that they arouse great interests from wide application areas, and further incur a wave of research and deployment of the brand new Internet-based applications to make the P2P-based applications successful in real life.
- Even though *fully decentralized P2P systems* can be further divided into subclasses according to either their *structure*: *flat* and *hierarchical* or their *topology*:

[1] http://www.bestpeer.com.

structured and *unstructured*, the latter classification is often used. The difference between structured P2P systems and unstructured P2P systems is whether the sharable objects are precisely mapped to their locations or not. In an unstructured P2P system, each peer stores files at will and utilizes heuristic routing strategies, such as routing indices, to facilitate looking up expected files from others. However, these systems might be weak in scalability, since their network might be flooded with query messages if they do not adopt effective searching schemes. In structured P2P systems, such as Chord, CAN, Tapestry, Pastry and Viceroy, peers only contain the sharable objects related to their identifiers, which can be efficiently located through key-based routing (KBR) strategies. Indeed, thanks to the features of the distributed hash tables employed in structured P2P system, queries can reach the locations of the desired objects within $O(\log n)$ or $O(dn^{1/d})$ hops. In addition, it is also advantageous in scalability, fault resilience, anonymity and security over either centralized P2P systems or unstructured P2P systems. However, in structured P2P systems, since the data placement are tightly controlled, their cost to maintain the structured topology is very high, especially in a dynamic environment.

– Different from the other two P2P architectures, *a hybrid P2P system* consists of two kind of nodes and forms a hierarchy of two tiers, where the upper tier serves the processing of the lower one. Actually, hybrid P2P systems can also be considered as a special type of hierarchical unstructured P2P systems. In the current Gnutella, its supernodes form an upper layer, which provides similar services to the ordinary peers as the central server in a centralized P2P system. The common peers, on the other hand, can enjoy much more services from the supernodes, especially in the process of resource location. As aforementioned, the supernodes are different from the servers in client-server systems. In BestPeer and various applications implemented upon it, the upper tier is made up of LIGLOs, which generate an unique identifier for their peers, facilitate peers to recognize and further dynamically reconfigure their neighbors. These desirable functionalities are helpful for peers to naturally evolve into interest communities. In short, hybrid P2P systems combine advantages of both fully distributed and client-server systems.

Besides outlining the categories of P2P system architectures, analyzing their features and comparing their advantages and disadvantages, we have studied how peers in different architectures define their neighbors, i.e., statically or dynamically, and figured out the mechanism supporting dynamic self-reorganization and peers evolving into interest communities.

http://www.springer.com/978-3-642-03513-5

Peer-to-Peer Computing
Principles and Applications
Vu, Q.H.; Lupu, M.; Ooi, B.C.
2010, XVI, 317 p., Hardcover
ISBN: 978-3-642-03513-5