Contents

1 Scalable and Secure P2P Overlay Networks **1**

1.1 Introduction 1

1.2 P2P Overlay Network Characteristics 3

1.3 Scaling techniques 8

1.3.1 Classification of P2P systems 9

1.3.2 Unstructured P2P 10

1.3.3 Structured P2P 15

1.3.4 Hybrid P2P 20

1.3.5 Cycloid: An Example of Structured P2P DHTs 22

1.4 Security Concerns 25

1.4.1 Authenticity 25

1.4.2 Anonymity 27

1.4.3 Attacks on the network 28

1.4.4 Secure routing 30

1.5 Concluding Remarks 32

Chapter 1

Scalable and Secure P2P Overlay Networks

Haiying Shen, Aharon S. Brodie, Cheng-Zhong Xu, and Weisong Shi
Wayne State University, Detroit, MI 48202

Abstract

Peer-to-peer(P2P) overlay network is a logical network in which peers are organized without any centralized coordination. Each peer has equivalent responsibilities, and offers both client and server functionalities to the network for resource sharing. In this article, we present an overview of distinguishing characteristics of the network, including decentralization with ad-hoc connectivity and self-organization, dynamics/churn, scalability, security, performance, fault resilience, and interoperability. Focus is on the scalability and security issues. Highly scalable systems expand in size without performance degradation; a secure P2P system provides a safe environment for participants by resisting different kinds of attacks and avoiding malicious nodes.

P2P systems can be classified into structured, unstructured, and hybrid categories. Representatives of the systems and related scaling techniques in each class are reviewed. P2P systems provide security challenges in four aspects: authenticity, anonymity, attacks, and secure routing. Their requirements and security measures are discussed.

1.1 Introduction

Since the inception of Napster in the late 1990s, peer-to-peer (P2P) computing model has grown dramatically. In literature, however, there is little consensus about its definition. An Intel P2P

working group defines it as "the sharing of computer resources and services by direct exchange between systems" [85]. Veytsel defines P2P as "the use of devices on the internet periphery in a nonclient capacity" [110]. Shirky [103] presents a more detailed definition as "a class of applications that takes advantage of resources — storage, cycles, content, human presence — available at the edges of the Internet. Because accessing these decentralized resources means operating in an environment of unstable connectivity and unpredictable IP addresses, P2P nodes must operate outside the DNS system and have significant or total autonomy from central servers". The author summarizes two criteria to judge if a system is P2P or not: (1) Does it treat variable connectivity and temporary network addresses as the norm? (2) Does it give the nodes at the edges of the network significant autonomy?

Essentially, P2P system is a logical network on top of physical networks in which peers are organized without any centralized coordination. A peer (or node) is an abstract notion of participating entities. It can be a computer process, a computer, an electronic device, or a group of them. Each peer has equivalent responsibilities, and offers both client and server functionalities to the network for resource sharing.

P2P systems are a complement to client/server computing model. Although it is popular in distributed computing, the client/server model has weak scalability and availability because it is prone to bottleneck traffic and failures at the servers. On the other hand, client machines on the network edge have seen a steady increase in processing networking and storage capacities. P2P computing exploits the potential of distributed resources by shifting the service away from central servers, outwards to the edge and allowing clients to share data and other resources such as spare cycles, disk spaces, and network bandwidth directly.

P2P systems bring about many benefits, such as aggregating resource, cost sharing/reduction, utilizing spare resource, enhancing scalability/reliability, assuring anonymity/privacy in resource sharing, and being adaptive to dynamic/ad-hoc environments [77]. These benefits are demonstrated in a wide range of applications, including distributed computing, file sharing, multicasting, collaboration, platforms, search engines, agent based systems, awareness systems, mirror systems, naming systems, etc [77, 51].

P2P systems can be traced back to ARPANET in the late 1960s. ARPANET connected hosts as equal computing peers, which had equal rights in sending and receiving packets. The decentralized control model of early Usenet bears much resemblance to today's P2P applications; Domain Name System (DNS) combines the P2P concepts with a hierarchy of information ownership. The P2P model gets momentum with the popularity of Napster for music sharing on the Web [81]. An important novel feature introduced by Napster is that the concepts of ownership and distribution of information are differentiated so that people can distribute information, which they do not own, in a free and pseudo-anonymous manner.

Although the Napster web site was forced down due to a lawsuit for copyright infringement, research on the P2P computing model has never stopped. Many Napster offsprings have been proposed over the past few years. Representatives include Gnutella [38], Kazaa [54] and Freenet [17].

Gnutella inspired Sun's Infrasearch for P2P information retrieval. Freenet provides an anonymous method for storing and retrieving information. That is, a user can make requests for data without anyone being able to determine who is making these requests. Likewise, if a user stores a file in the system, it should be impossible to determine who placed the file into the system. Finally, the operator of a Freenet node should have no knowledge of what data is stored on the local disk. In Napter and its offsprings, peers are connected in a simple unstructured network, but interacted in a fairly complicated protocol. Their complex interaction directly affects the overall system performance and network scalability and limits the usefulness of the networks for new applications beyond traditional file sharing.

On another track altogether, there are studies on distributed hash tables (DHTs) that organize P2P systems in a structured network. A DHT maps document keys to network peers based on a consistent hashing function [53]. Representatives of the DHTs include CAN [92], Chord [107], Pastry [96], Tapestry [124], Kademlia [75], and Cycloid [101]. They organize the peers in various ways for efficient management of distributed data.

Two primary concerns in the development of P2P systems are scalability and security. Scalability refers to the capability of a system to maintain consistently acceptable levels of performance, while peers are added. A highly scalable P2P system can grow to several million concurrent participants without significant performance degradation. Security represents the system ability to resist different kinds of attacks and avoid malicious peers so as to provide a safe distributed processing environment for participants. Making a P2P system secure is a challenge because the system is open and each participant is potentially unreliable. With the P2P model quickly emerging as a computing paradigm of the future, there is an ever-increasing need for distributed algorithm that would allow P2P applications to scale to a large community of users, as well as security methods to make P2P systems secure.

The rest of this chapter is structured as follows. Section 1.2 presents an overview of the characteristics of P2P overlay systems. Section 1.3 and Section 1.4 review scaling techniques and security measures in both structured and unstructured networks. Section 1.5 concludes this chapter with remarks on open issues.

1.2 P2P Overlay Network Characteristics

This section presents an overview of distinguishing characteristics of P2P systems: decentralization with ad-hoc connectivity and self-organization, dynamics/churn, scalability, security, performance, fault resilience, and interoperability. They have a major impact on the effectiveness and deployment of P2P systems.

Decentralization and self-organization. As its name implies, P2P upends the traditional client/server model by decentralization. In a P2P system, each peer has control over its data and resources independently. P2P computing makes new services available to end users in a novel way by making use of their machines as active participants in computing processes rather than just resource requestors. This decentralization characteristic offers each node in P2P systems a self-organization function. Self-organization is required for reliability and for the support of ad-hoc connectivity of the peers. By ad-hoc connectivity, we mean that the connections between peers are improvised instead of preexisting, due to the continuous and fast peer joining, leaving and failure. The objective of reliability requires P2P systems to adapt to such system membership changes rapidly and keep system consistency at a low cost. Self-organization has a two-fold objective: self-maintenance and self-repair. For self-maintenance, node joining and leaving is handled in a distributed manner, without requiring the member change information to be propagated through the entire network for re-organization. When a node joins system, it builds connections with other nodes to become an active participant. When a node leaves, it is removed from the whole system without leaving any trace of its existence. For example, in Pastry [96, 97], self-organization handles the arrival and departure of nodes by information exchange within the overlay network.

When a node fails without warning, its connected nodes still regard it as alive. In this case, a self-repair function helps them to update their routing and neighboring information. In Chord [107], stabilization implements self-organization for node failures to keep the Chord ring intact.

Dynamics/Churn. Churn represents a situation where a great number of nodes join, leave and fail continually and rapidly. Previous work shows that an ideal overlay can still locate an object efficiently even after some fraction of the nodes are failed. Because churn has the potential to increase P2P system workload, reduce the object location efficiency, an ideal overlay should perform well under churn. Liben-Nowell *et al.* [65] indicate that P2P systems' "maintenance bandwidth" depends on the rate at which nodes tend to enter and leave the system. They prove that stabilization protocol of Chord is near optimal for churn. Saia *et al.* propose a virtual CAN butterfly content addressable network that routes efficiently in the face of churn [98]. Pandurangan *et al.* present a centralized algorithm to ensure connectivity when many nodes fail [87]. Ledlie *et al.* [63] propose a self-organizing hierarchically-based P2P system. Mahajan *et al.* [71] describe a new technique to reduce maintainance cost while providing high reliability and performance. Prakash *et al.* [66] regard churn as a security threat and propose a churn-resistant P2P web caching system based on Kelips system [44] to ensure high cache hit rate without stabilization protocol. Li *et al.* [64] provide a unified framework for evaluating cost and performance under churn in different DHT protocols. Rhea *et al.* [94] propose Bamboo, a DHT protocol for dealing with network high churn efficiently and gracefully. Castro *et al.* [9] present an optimized Pastry, MSPastry, to deal with consistent routing under churn with low overhead. Based on Tapestry, Hildrum *et al.* [48] propose algorithms for adapting to arriving and departing nodes and provide necessary failure recovery mechanisms.

Table 1.1: Types of Anonymity.

Types of Anonymity	Difficult for Adversary to Determine:
Author	Which users created which documents?
Server	Which nodes store a given document?
Reader	Which users access which documents?
Document	Which documents are stored at a given node?

Scalability. P2P computing is an alternative to the client/server computing for greatly improved scalability. However, the problems, such as low bandwidth and high latency on peer connections combined with large amounts of traffic, pose serious barriers to the scalability of P2P networks. Napster, as a centralized network, attacked the scalability problem by having the peers directly download music files from the peers that possess the requested document. However, its scalability depends entirely on the scalability of the central server. In unstructured P2P systems, such as Kazaa and Gnutella, a huge amount of bandwidth is required for routing control messages like "I'm alive" and other network chatter before any actual data messages are transferred. Many architectures and scaling techniques, approaches such as increasing ring search and random walks, are proposed for reducing congestion and enhancing the scalability of Gnutella. In addition, structured P2P overlay networks such as Chord, CAN and Past appeared recently to be able to let the P2P system scales well to potentially billions of keys, stored on hundreds or millions of nodes. They use consistent mapping between the key and the hosting node to ensure the availability of the object when the hosing node is alive. A comprehensive review of unstructured and structured P2P systems will be presented in Section 1.3.

Security. P2P systems provide security challenges since they are not structured in the traditional client/server model. The client/server model allows the server to authenticate users, provide measures to validate data, and poses smaller security risks because the communication has no intermediaries. P2P systems, on the other hand, especially today's popular decentralized models, lack an authoritative entity that can authenticate and verify users and data. Moreover, communication passes through foreign nodes on the network which can be malicious. P2P systems raise security issues in four aspects:

- *Authenticity*. In order to authenticate data on today's P2P networks, an alternative to centralized verification systems must be implemented. Since data is replicated on the network, it is further necessary to verify existing copies as well. Different aspects of authentication will be discussed, from originality of data to ensuring copies have been modified. Different methods proposed to solve these problems will be introduced, such as reputation systems and file hashes.
- *Anonymity*.An important feature in P2P networks is user anonymity. Table 1.2 presents a summary for the anonymity discussed in [78]. Users do not need to log in, or identify them-

selves, upon entering the network. In certain P2P implementations, Anonymity is achieved by indirect communication, the client and server communicate through other nodes. While this is an attractive feature for users, it complicates other aspects of the P2P system. Since a node does not know whom it is communicating with, great measures must be taken to protect itself, both in terms of security and data validity. Moreover, in certain implementations, network usability degrades. Anonymous implementations will be discussed, along with their advantages and disadvantages.

- *Attacks*. The insecurity of P2P networks have brought forth numerous attacks. These attacks include denial of service, node masquerading and data tampering [25]. Attacks discussed in this chapter are denial of service, which virtually disconnects a node from the network, and byzantine and sybil attacks, which can lead to malicious nodes masquerading as others as well as communication and data tampering. Daswani *et al.* [24] and Lynch [70] provide ways to minimize the effect of such attacks. Examples of these attacks and their attempted solutions will be further discussed.

- *Secure routing*. One of the basic requirements of a P2P network is to provide reliable and efficient routing of communication between nodes. Reliability focuses on ensuring messages arrive at their destination despite legitimate, although sometimes unforseen, circumstances. Secure routing, however, deals with malicious nodes actively attempting to disrupt network communication. This is a crucial and difficult task, although nodes will actively misroute messages or segregate nodes from the network, the network would still attempt for the messages to reach their destination. The three crucial tasks a P2P network needs to implement securely exist in different aspects of the protocol. Securely assigning node IDs, updating the routing table [36], and ensuring successful message propagation. These will be further discussed, along with possible attacks and their solutions.

More details about the security requirements and current security measures will be discussed in Section 1.4.

Performance. Recall that P2P overlay network is a logical structure constructed upon physical networks. The shortest path (the least hop count routing) according to the routing protocol is not necessarily the shortest physical path. The logical structure of the overly should take into account the physical structure of the underlying network. Techniques to exploit topology information in overlay routing include geographic layout, proximity routing and proximity-neighbor selection [11].

- *Geographic layout method* maps the overlay's logical id space to the physical network so that neighboring nodes in the id space are also close in the physical network. Topologically-aware CAN [93] adopts this approach. But this method can lead to an uneven distribution of nodes in the overlay, increasing the chances of overloading nodes and rendering the maintenance cost formidable. Xu *et al.* [117] claim their study shows that for a typical 10,000-node

Topologically-aware CAN, 5 nodes can occupy 85-98% of the entire cartesian space, and some nodes have to maintain 450-1500 neighbors. Castro *et al.* [11] indicate that this method is not suitable for a one-dimensional id space and neighboring nodes in the id space are more likely to suffer correlated failures.

- *In proximity routing*, the logical overlay is constructed without considering the underlying physical topology. In a routing, the node with the closest physical distance to the object key is chosen among the next hop candidates in the routing table whose entries are selected based on proximity metric among all nodes that satisfies the constraint of the logical overlay (e.g., in Pastry, the constraint is the nodeID prefix). This method has been applied to Chord [107] and CAN [92]. However, the benefits of this method are dependent on the determination of an appropriate number of the hop candidates.

- *Proximity neighbor selection* is a middle-ground approach between the above methods. It selects the routing table entries pointing to the topologically nearest among all nodes with nodeId in the desired portion of the id space. Castro *et al.* [11] implement this method in Pastry and Tapestry with low overhead. It maintains system load balancing and robustness with a comparable delay stretch compared to Geographic layout method. Xu *et al.* [117] point out some limitations of this application and propose a way to build topology-aware overlays using global soft-state. It combines landmark clustering and RTT measurement to generate proximity information and stores the system information (such as proximity and load information) as objects on the system itself, which is easy to update and retrieve. In addition, it uses publish/subscribe functionality that allows nodes to subscribe to the relevant soft-state and get notified as the state changes necessitate neighbor re-selection.

  Waldvogel *et al.* [112] propose an efficient topology-aware overlay network, Mithos. In contrast to other approaches, Mithos does not require full topology knowledge, even the forwarding and routing information is minimum. At the same time, Mithos provides a close conceptual integration between geographic layout and proximity routing, as well as a powerful addressing scheme directly suitable for use in DHTs. It provides locality-aware connectivity, thereby ensuring that a message reaches its destination with minimal overhead.

Fault Resilience. A good P2P system should still work well in the face of failure problems such as disconnections/unreachability, partitions, and node failures. The most vital issue of fault resilience is to ensure that all nodes in the P2P system are alive and there is no disconnection. Therefore, techniques that detect, manage, and recover from failed node or disconnection are reviewed [77].

- *Detection.* Chord uses stabilization, in which each node sends probe queries to its successor, to maintain intact ring. Sending ”I’m alive” message periodically is a method to detect the failed nodes and disconnection. Pastry can periodically perform an expanding ring multicast search for other Pastry nodes in their vicinity to check the disconnected overlay. Gnutella and Freenet resort to resource intensive naive broadcast queries to tolerate against node failures.

- *Management.* File replication is a way to solve the problem when the object hosting node has failed. P2P networks such as Napster and Gnutella have a passive and an uncontrolled replication mechanism based only on the file's popularity. Freenet and Publius [111] use controlled replication to provide persistence guarantees. OceanStore maintains a two-layered hierarchy of replicas. In contrast to promiscuous replication [40], OceanStore does not bind floating replicas to specific machines, and it does not replicate all objects at each server.

- *Recover.* In a P2P system, how can the connected nodes still collaborate well when a failure happens? How can the disconnected node join in the ongoing computation when it reappears? Previous works primarily focused on providing resiliency to servers and network failures for distributed file systems. For example, Coda [99] adopted server replication and disconnected operation. Currently, some other methods are proposed. One is to let a "relay" node to substitute for the destination temporarily, like Groove [99, 84]. Second is to let the source wait with the messages until the destination reappears, as in Magi [8]. Third, Grid computing solutions, such as Legion [42], deal with such problems by restarting computations on different nodes.

Interoperability. Different P2P networks have their advantages and disadvantages. For example, Freenet provides a certain degree of security but cannot ensure object location and each node cannot control the files stored in itself. By contrast, structured P2P networks such as Chord can ensure the object availability. Gnutella enables each node to hold what it desires. Therefore, it is desirable to combine the different kinds of P2P network as a whole to take advantage of each network's benefits. Because different P2P networks deploy distinct protocols with different characteristics, the most important issue is to let the P2P systems to interoperate. Lui *et al.* [67] propose a framework to integrate various P2P file sharing protocols using P2P gateways in order to maximize the benefit of P2P file sharing application. The community of P2P developers gather together in the P2P Working Group [85] to build common standards that would enable common understanding among P2P developers. The project JXTA [50] is Sun's foray into providing a utility application substrate for building P2P applications across platforms, OSs, and programming languages. A gateway project underway is World Free Web (WFW), which is to combine Freenet and World Wide Web. Wiley [114] explains the characteristics of five popular networks – Freenet, Gnutella, Mojo Nation, Free Haven, and Publius, and evaluates the strengths of each network that can be offered to the all-encompassing OmniNetwork. P2P interoperability also has commercial significance. For instance, new file-swapping applications, such as StreamCast's test version of Morpheus [79], are bridging separate networks, promising to improve service and content offerings.

1.3 Scaling techniques

An ideal scalable P2P network is that when increasing the number of nodes, aggregate storage space and file availability should grow linearly, response time should remain constant, and search throughput should remain high or grow. The following are the main criteria for a system's scala-

bility:

- *Decentralization.* In centralized systems, the central nodes are easily to become hot-spots that quickly become overloaded as the network grows and their limited bandwidth prevents systems to grow unlimitedly. Distribution of queries in networks decreases per node's average bandwidth and workload, which directly gain the speed of serving and overall system throughput.
- *Nodes' storage overhead.* In P2P systems, nodes need to contribute some storage space for other users' data or routing information that are the indices of other nodes. However, when a large amount of storage is required, some nodes may not afford it, especially when the nodes grow to a very large number, leading to poor scalability of system.
- *Object location cost.* Generally, with the growing number of nodes, the object location cost, namely the number of hops traversed, will increase because searching is performed among larger number of nodes. Therefore, low object location cost leads to decreasing query traffic, and increases system throughput, resulting in providing good scalability.
- *Workload: bandwidth and processing cost.* When a query passes through a number of nodes, each node uses bandwidth resources because it sends and receives query and response messages [121]. Low bandwidth and processing cost of each node, and of whole system are essential for improving scalability.
- *System fault resilience.* A scalable system can handle nodes join, leave and failure with little negative affection.

In the following, we present a classification of P2P systems. It is followed by a comprehensive survey of scaling techniques in unstructured and structured P2P systems.

1.3.1 Classification of P2P systems

Napster is a typical example of centralized P2P, in which the elements of both pure P2P and client/server systems coexist. Napster client downloads a file directly from another Napster client's machine and the Napster servers answering a list of matching files and locations and brokering client connections. Other centralized models include Aimster [3], Magi [8], Softwax [105], and iMesh [49]. As a centralized network, it attacks the scalability problem by having the peers directly download music files from the peers that possess the requested document. However, because file location is done from the central server, the cost incurred on the central server is so high that makes it very possible for the central server to be a bottleneck. Yang *et al.* [120] study issues and tradeoffs in designing a scalable hybrid P2P including chained architecture, replication and hash architectures.

Poor scalability of these centralized P2P systems motivates researchers to explore various methods for robust and scalable distributed P2P systems. As a result, unstructured P2P, structured P2P

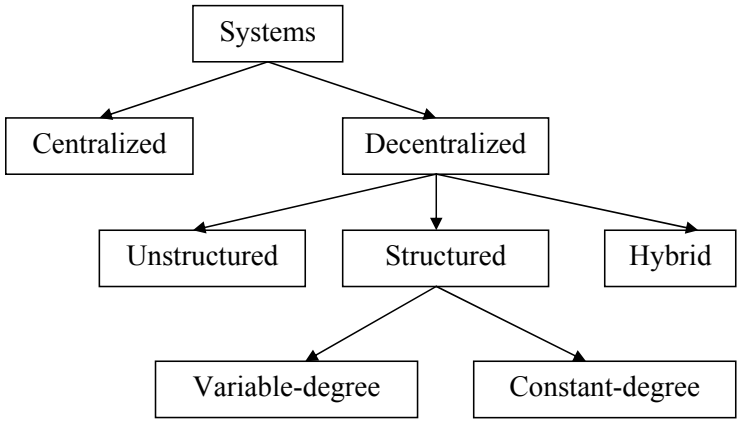


Figure 1.1: A Classification of P2P Overlay Networks.

and hybrid P2P (combination of unstructured and structured P2P) are developed. Figure 1.1 shows a classification of P2P systems of Napster offspring. Unstructured P2P overlay networks such as Gnutella and Freenet do not have strict control over the topologies, and they do not assign responsibility for data to specific nodes. Nodes join and leave the network according to some loose rules. On the contrary, structured overlay networks such as Chord, Pastry, Tapestry, CAN, have strictly controlled topologies and the data placement and lookup algorithms are precise. According to the node degree, that is, the number of entries in a node's routing table, structured P2P can be classifies into variable-degree P2P and constant-degree P2P.

Unstructured system has improved system stability by removing central servers. Its advantages include overall simplicity, its ability to endure high rate churn and its support for key word searches. However, it cannot ensure the object location and too many massage forwarding can easily result in network congestions. Structured system is developed to improve systems scalability by reducing the amount of forwarding messages and nodes' storage for routing index. It guarantees object location within a bounded lookup cost even if the system is in a continuous state of change. Because of their potential efficiency, robustness, scalability and deterministic data location, structured networks have been studied intensively in recent years. But it also has some disadvantages. First, DHT is less adept at supporting keyword searches because it is designed for exact-match query. Second, it incurs substantial repair operations for high churn rate. Third, hot-spots are generated for too frequently accessed files. People are realizing each system's benefits and drawbacks from the theoretical work, experiments and practice, and trying to develop better scalable systems. Hybrid network is such an example which takes advantages of unstructured and structured P2Ps. Representatives of it are Yappers [35] and Freedman *et al.* [33].

1.3.2 Unstructured P2P

Gnutella is a typical example for unstructured P2P. Gnutella uses flooding-based search to perform the object location. That is, to find a file, a node sends a query message to each node connected

with it. The nodes then rebroadcast this query in turn to every node they are connected to, except for the original node, and so on. Each query contains a time to live (TTL) and will be forward until the TTL is expired. If a node has the desired file, it will respond and the file will return along the route the query comes from. In a sense, Gnutella improves the scalability of Napster because it does not need to depend on the central server. However, the number of queries and the number of potential responses increases exponentially with each hop. For instance, each node has 2 connections to other nodes and the TTL is 7, then the number of queries sent will be 128 and the number of response may depend on the file's popularity. As a result, the load on each node grows linearly with the total number of queries, query number increases with the system size, and large systems quickly become overwhelmed by the query-induced load. So, Gnutella does not scale very due to a large amount of traffic. Besides, without heterogeneity consideration, some nodes with limited capabilities become bottlenecks when the network size surges, resulting in slower response time, and few availability. In the following, we will present a survey for key techniques for improving unstructured P2P scalability.

Gnutella TTL control. TTL selection is very important in Gnutella; too high a value generates much network traffic, whereas too low will result in no search results. Lv *et al.* [68] propose increasing ring search to help solve this problem. Increasing ring achieves the savings at the expense of slight increase in delay to find the object, which improves Gnutella's scalability to a certain degree. However, the method may cause the problems of same value TTL, query reprocessing and message duplication, which may lead to more redundant network traffic. These problems can be solved by iterative deepening search method proposed by Yang *et al.* [121]. Based on Gnutella, this method iteratively sends the query with different TTL with a mechanism to avoid query reprocessing until the query is satisfied. It greatly improves Gnutella's scalability by reducing the number of lookup hop.

Intelligent peer selection. Gnutella's poor scalability is mainly caused by its flooding-based search. Some proposals suggest to reduce the number of query receivers to improve system's scalability, that is, a query is sent to a part of a node's neighbors rather than all of its neighbors. Ripeanu [95] measure the Gnutella network and find that network following the power-law pattern and estimate the generated traffic via the percentage of each kind of messages with the crawler program. Lv *et al.* [68] propose random-walkers search, which greatly reduces the load produced by a query. Random-walks forwards k query messages, and each query is sent to a randomly chosen neighbor at each step until the object is found. It also adopts active replication which produces lower overall query load than non-active node-based replication. In addition, it uses uniform random graph building algorithm instead of power-law random graph [2] to reduce the likelihood of very-high degree nodes. Portmann [89] studied the cost of Gnutellas broadcasting and proposes the use of rumor mongering or gossip protocol as an alternative routing method to reduce the cost via simulation. Yang *et al.* [121] claim that Gnutella uses breadth-first traversal (BFS) over the network with depth limit TTL. They propose direct breadth-first traversal (DBFS) to overcome time consuming problem of multiple iterations. DBFS reduces the number of nodes

that query reaches, as well as maintain the quality of responses. Yang *et al.* [121] also propose local indices technique, in which a node maintains an index over the data of each node within r hops of itself, where r is a system-wide variable. Queries are then processed by a smaller set of nodes than without this technique, while yielding as many results. Crespo *et al.* [23] propose routing indices (RI) schemes including compound RI, hop-count RI, and the exponential RI. A RI is a data structure that returns a list of neighbors for a query. Simulations show that RIs can improve performance by one or two orders of magnitude compared to a flooding-base system, and by up to 100% compared to a random forwarding system.

Heterogeneity consideration. Heterogeneity is an important factor to consider for scalability. Heterogeneity means that different nodes have different capabilities, particularly in terms of bandwidth, but presumably in terms of other resources such as CPU, memory and disk. Attention to node heterogeneity means to avoid overloading any of the nodes by taking into account the nodes' capacity constraints. In an ideal scalable system, nodes with more capability should be responsible for more work, while nodes with less capability should have less work. Flow control and topology adaptation are such methods proposed by Lv *et al.* [69] based on their previous work. These methods is used to achieve (1) restricts the flow of queries into each node so they don't become overloaded and (2) dynamically evolves the overlay topology so that queries flow towards the nodes that have sufficient capacity to handle them. The random-walks based search combined with TTL and "state-keeping"algorithms [2, 68] lets the query being forwarded to high capacity nodes, and thus makes it much more possible to find the targeted files.

Chawathe *et al.* [14] indicate the DHT's disadvantages, and propose *Gia* to make Gnutella more scalable, which has the ability to deal with much higher aggregate query rates and function well even with growing system size. It modifies Gnutella's algorithms to include dynamic topology adaptation, flow control, one-hop replication, and attention to node heterogeneity. Simulations show that Gia improves Gnutella's system capacity by three to five orders of magnitude.

Super-peer topology. FastTrack [29] is one of the super-peer topology. Currently, two file sharing applications, KaZaA and Morpheus are based on the FastTrack protocol. For instance, the KaZaA application has had upwards of 20 million downloads and it can have anywhere up to 800,000 users connected at one time. The Morpheus multimedia file-sharing system reported over 470,000 users sharing a total of .36 petabytes of data as of October 26, 2001.

The FastTrack architecture consists of SuperNodes with fast connections and Nodes with slower connections. A SuperNode connects with other SuperNodes and some Nodes at the same time. A Node connects with a SuperNode. Consequently, a two-tier topology is generated in which the nodes at the center of the network are faster and therefore produce a more reliable and stable backbone. This allows more messages to be routed than if the backbone were slower and therefore allows greater scalability.

Routing on FastTrack is almost the same as on Gnutella, but broadcasting is between the Su-

perNodes only. A Node sends its query to its SuperNode only. Every SuperNode that broadcasting reaches searches an index that contains all the files of its connected Nodes. The fast connection SuperNodes lessens the workload produced by a large amount of messages. In addition, FastTrack use the nodes' heterogeneity to its advantage as the nodes with limited capabilities are shielded from query processing and traffic. By combining the P2P and client/server models, this topology provides a useful balance between the efficiency of centralized search and the autonomy, load balancing, and robustness of distributed search.

Recently, it is proposed to apply the hierarchical designs of FastTrack to the Gnutella network [37]. However, its scalability has neither been measured nor been analyzed. Yang *et al.* [122] present practical guidelines and general procedure for the design of an efficient super-peer network that is almost the same as FastTrack. They also point out a potential drawback of super-peer node network, that is, when a SuperNode fails or simply leaves, all its clients (the Nodes in the same cluster with that SuperNode) become temporarily disconnected until they find a new SuperNode to connect to. To address this problem, the method of super-peer redundancy is proposed. Instead of a single SuperNode in a cluster (the SuperNode with its clients), there are some redundant SuperNode partners with the same responsibility in a round-robin manner. To reduce the individual query load, each partner is connected with half of the clients.

Caching and content placement. Caching is a short-term solution to increasing the scalability of Gnutella. Kunwadee [106] finds that the popularity of search strings follows a Zipf-like distribution. He proves that caching a small number of query results significantly decreases the amount of traffic of network. He finds that caching at one Gnutella node can result in up to a 3.7-time reduction in traffic while using only a few megabytes of memory. As more nodes implement caching, more traffic is reduced. Markatos [74] proposes another caching mechanism that caches Gnutella query results for a limited amount of time to reduce the query traffic. Simulations show that it improves performance by as much as a factor of two.

FastTrack [29]'s proactive replication is far from optimal as the relative index capacity dedicated to each item in the SuperNodes is proportional to the number of copied nodes. Cohen *et al.* [18] study optimal way to replicate data given fixed constraints on per-probe capacity. They show that the Uniform and Proportional strategies constitute two extreme points of a large family of strategies which lie "between" the two and that any other strategy in this family has better expected search size. They then show that one of the strategies in this family, square-root replication, minimizes the expected search size on successful queries.

Clustering/arrange overlay topology. Pandurangan *et al.* [87] propose techniques for constructing Gnutella-like networks in a distributed fashion. By the heuristic proposed protocol, newly arriving nodes decide which network nodes to connect to, and existing nodes decide when and how to replace lost connections. The proposal results in connected networks of constant degree and logarithmic diameter. This method is focused on pure-search networks without copying content indices. *CAP* [57] is a cluster-based architecture for P2P systems, which uses a network-aware

clustering technique [56] to group hosts into clusters. Each cluster has one or more delegate nodes that act as directory servers for the objects stored at nodes within the same cluster. A central server tracks existing clusters and their delegates. To reduce query latency, delegate nodes maintain a cache of recently received queries. CAP would also be more stable since clusters join and leave the network less frequently than individual clients.

In Gnutella, the network topology is unrelated to the data placement, and the nodes receiving a query are unrelated to the content of the query. A node doesn't have any information about which other nods may best be able to resolve the query. Some proposals improve Gnutella by reorganizing network topology based on content location, that is, to cluster peers based on property similarity or results availability to avoid query messages flooding, reduce traffic, gain the number of query hit messages and save resources in handling irrelevant queries in the P2P network. In these approaches, in stead of blind flooding in Gnutella, the queries are forwarded to those nodes that most like to have the desired data to improve query performance.

Ramanathan *et al.* [91] propose a mechanism, using only local knowledge, to let a peer connect to the peer that frequently provides good results. This leads to clusters of peers with similar interests, and in turn allows us to limit the depth of searches required to find good results. Schlosser *et al.* address the scalability problems by imposing a deterministic shape on P2P networks, called HyperCuP [100], which relies on the concept of prefix-based routing and allows for very efficient broadcast and search. They describe a broadcast algorithm that exploits the topology to reach all nodes in the network with the minimum number of messages possible. They also provide an efficient topology construction and maintenance algorithm which does neither require a central server nor super nodes in the network. Moreover, they show how their scheme can be made even more efficient by using a global ontology to determine the organization of peers in the graph topology. Hang *et al.* [47] propose a method for clustering peers that share similar properties together. They also propose a new intelligent query routing strategy, the firework query model. Crespo *et al.* [23] propose semantic overlay networks (SON) based on the semantic relations among peers. SON establishes semantic relations between query and peers and routes query directly to relevant peers. Furthermore, layered SONs are proposed in order to increase the availability of results.

The SON supports only limited meta data such as simple filenames. On the other hand, currently schema based networks such as the Edutella [82, 6, 46, 1, 61, 83] provide complex query facilities but no sophisticated means for semantic clustering of peers and their broadcasting does not scale well. In this condition, a new class of P2P system that combines SON, super-peer topology, and schema based networks have emerged [60, 62]. Cooper *et al.* [20, 21] present a Search/Index Link (SIL) model for analyzing and visualizing P2P search networks. They suggest to reduce or eliminate redundant work of network to improve network's efficiency. The SIL model describes existing networks such as super-node networks and global index networks, while also yielding novel organizations such as parallel index networks and parallel index cluster networks. Based on SIL, Cooper *et al.* [22] examine how to reduce the load on nodes by allowing peers to self-organize into a relatively efficient network, and then self-tune to make the network even more efficient. Unlike their previously studied architectures, they propose "ad hoc, self-supervising" networks that avoid restrictions on who a node can connect to or what information can be exchanged. This

network is optimized for a heterogenous network as well as for a more homogenous network where all nodes have roughly similar capabilities. Their results indicate that their ad hoc networks are more efficient than popular super-peer topologies for several important scenarios.

New protocols. Gnutella2[39] is a reworking of the Gnutella protocol. It uses the same peer-based network as Gnutella, but it drops all of the old Gnutella protocol except for the connection handshake and adopts an entirely new and complex system. Gnutella2 uses UDP rather than TCP/IP network protocol for searches, an extensible binary XML-like packet format and includes many extensions such as intelligent query routing, SHA1 checksums, parallel downloading in slices (swarming), etc. Preceded by their previous work, iterative deepening search, DBFS, super-peer network, SON and SIL, which are based on Gnutella, Yang *et al.*[119] propose non-forwarding architecture (NFA) in the context of the GUESS protocol. In GUESS, peers directly probe each other with their own query message, instead of relying on other peers to froward the message. Unlike the flooding-based search in Gnutella, in NFA, the messages are not forwarded, and peers have complete control over message receivers and message receiving time. As a result, NFA greatly reduces the message forwarding overhead.

1.3.3 Structured P2P

The different proposals for improving scalability of Gnutella made a significant progress towards scalability. At the same time, another class of P2P, structured P2P systems, are developed with more scalable architecture. These systems have no central directory server, and so are decentralized. ”Structured” means that the P2P network topology is tightly controlled and that files are placed not at random nodes but at specified locations that will make subsequent queries easier to satisfy. These structured P2P systems include Chord, Pastry, Tapestry and CAN, etc., which support a DHT functionality. Each objects place is stored at one or more nodes selected deterministically by a uniform hash function. Though these DHT systems have great differences in implementation, they all support a hash-table interface of put(key,value) and get(key) either directly or indirectly, queries for the object will be routed incrementally to the node. Although hash functions can help locate content deterministically, they lack the flexibility of keyword searching – a useful operation to find content without prior knowledge of exact object names. These structured systems are highly scalable as it make very large systems feasible; lookups can be resolved in a bounded number of overlay routing hops.

In the following, we review and compare some of the structured DHTs by focusing on their topological aspects within variable-degree and constant-degree P2P systems each.

Variable-Degree Structured P2P

In variable-degree structured P2P, the routing table size is related to the network size, typically is O($\log$ n). As a result, the routing table size is increased with the growing number of nodes in the system.

- *Hypercube-based.* Plaxton *et al.* [88] develops perhaps the first routing algorithm that could be scalably used for P2P systems. Tapestry and Pastry use a variant of the algorithm. The approach of routing based on address prefixes, which can be viewed as a generalization of hypercube routing, is common to all theses schemes. The routing algorithm works by correcting a single digit at a time in the left-to-right order: If node number 12345 received a lookup query with key 12456, which matches the first two digits, then the routing algorithm forwards the query to a node which matches the first three digits (e.g., node 12467). To do this, a node needs to have, as neighbors, nodes that match each prefix of its own identifier but differ in the next digit. For each prefix (or dimension), there are many such neighbors (e.g., node 12467 and node 12478 in the above case) since there is no restriction on the suffix, i.e., the rest bits right to the current bit. This is the crucial difference from the traditional hypercube connection pattern and provides the abundance in choosing cubical neighbors and thus a high fault resilience to node absence or node failure. Besides such cubical neighbors spreading out in the key space, each node in Pastry also contains a leaf set L of neighbors which are the set of $|L|$ numerically closest nodes (half smaller, half larger) to the present node ID and a neighborhood set M which are the set of $|M|$ geographically closest nodes to the present node.

- *Ring-based.* Chord uses a one-dimensional circular key space. The node responsible for the key is the node whose identifier most closely follows the key numerically; that node is called the key's successor. Chord maintains two sets of neighbors. Each node has a successor list of k nodes that immediately follow it in the key space and a finger list of $O(\log n)$ nodes spaced exponentially around the key space. The i^{th} entry of the finger list points to the node that is 2^i away from the present node in the key space, or to that node's successor if that node is not alive. So the finger list is always fully maintained without any null pointer. Routing correctness is achieved with such 2 lists. A lookup(key) is, except at the last step, forwarded to the node closest to, but not past, the key. The path length is $O(\log n)$ since every lookup halves the remaining distance to the home.

- *Mesh-based.* CAN chooses its keys from a d-dimensional toroidal space. Each node is identified by a binary string and is associated with a region of this key space, and its neighbors are the nodes that own the contiguous regions. Routing consists of a sequence of redirections, each forwarding a lookup to a neighbor that is closer to the key. CAN has a different performance profile than the other algorithms; nodes have $O(d)$ neighbors and path-lengths are $O(dn^{1/d})$hops. Note that when d=$\log n$, CAN has $O(\log n)$ neighbors and $O(\log n)$ path length like the other algorithms. This actually gives another way to deploy the hypercube as an overlay network.

eCAN [118], which means expressway CAN, augments CAN's routing capacity with routing tables of larger span to achieve logarithmic routing performance. To build the expressways, CAN's entire Cartesian space is partitioned into zones of different spans with the smallest zones correspond to the CAN zones. Therefore, each node owns a CAN zone and is also a resident of the expressway zones that enclose its CAN zone. By this way, the message can be traversed between not only CAN neighboring zones but also neighboring expressway zones.

- *Tree-based.* Maymounkov [75] proposes a novel XOR-based metric topology, Kademlia. It effectively treats nodes as leaves in a binary tree, with each node's position determined by the shortest unique prefix of its ID. For each node, the binary tree is divided into a series of successively lower subtrees that don't include this node, which knows at least one node in each of its subtrees. Kademlia defines the distance between two nodes is the XOR of their IDs. Its routing table is a binary tree whose leaves are $O(\log n)$ neighbors, where the i^{th} neighbor is a node within an XOR distance of $[2^i, 2^{i+1}]$. Kademlia employs a recursive algorithm for node lookups, that is, forwarding the lookup message to a constant number of nodes in the current node's routing table in parallel. This way can avoid timeout delays from failed nodes. The number of hops per lookup is $O(\log n)$. By increasing the routing table's size to $2^b \log_{2^b} n$, the lookup hops can be reduced to $\log_{2^b} n$. PRR's algorithm [88] uses a tree-like structure. It is to maintain and distribute object locations information to let access requests locate the addresses of copies of objects in order to ensure fast access and efficient network resource utilization.

- *Butterfly-based.* Fiat *et al.* [30] build a butterfly network of depth $\log n - \log\log n$ that can tolerate massive adversarial node failures and random object deletions. Each search for a data item in the network takes O($\log n$) time and requires at most O($\log_2 n$) messages. This network is censorship resistant in the sense that even after adversarial removal of an arbitrarily large constant fraction of the nodes in the network, all but an arbitrarily small fraction of the remaining nodes can obtain all but an arbitrarily small fraction of the original data items. The network can be created in a fully distributed fashion. It requires only O($\log n$) memory in each node. A variant of the scheme is also given that has the property that it is highly spam resistant: an adversary can take over complete control of a constant fraction of the nodes in the network and yet will still be unable to generate spam. Saia*et al.* [98] create another highly fault-resilient CAN based on butterfly structure with $O(\log^3 n)$ state at each node and $O(\log^3 n)$ per message routing overhead. Xu *et al.* [115] present Ulysses. The name space of Ulysses k-butterfly consists of k level and k-dimensional cuboids. Each node with randomly assigned level l represents a zone which is a subcuboid of l level-cuboid. By maintaining routing tables no larger than $O(\log n)$, Ulysses achieves lookup cost of $O(\frac{\log n}{\log\log n})$ with high probability.

- *Random graphs-based.* So far we have discussed the deterministic graphs, another direction for building DHTs relies on properties of random graphs to achieve logarithmic-time routing. $\mathbb{H}$-graph [59] network composed of d Hamilton cycles is proposed. In a layered construction of such random expander networks, any node can be located in $O(\log n)$ time with $O(\log n)$ degree, a join operation takes $O(\log n)$ time and $O(\log n)$ messages, while a leave operation takes $O(1)$ time and $O(\log n)$ messages. Aspnes *et al.* [4] propose a P2P system where nodes

are embedded at grid points in a one-dimensional real line. Each node is connected to its immediate neighbors and to multiple long-distance neighbors. They examine greedy routing in the random graph with degree l+1 (l is a polylogarithmic value) and derive upper and lower bounds on the expected routing distance and prove that the greedy routing can be a nearly optimal mechanism for searching even in the presence of many faults. Their results show that both bounds are proportional to $\frac{\log^2 n}{l \log \log n}$.

- *Hybrid structured-based.* Actually, some P2P network structure employ different geometric models. Let's take Pastry [96] as an example. Pastry's structure is the combination of a Hypercube and a Ring. Each node's routing table has the nodes information that share the current node's nodeId in the first n digits, but whose n+1 the digit has one of the $2^b - 1$ possible values other than the $n + 1^{th}$ digit in the present node's id, which is used in the hypercube routing. Each node's leaf set has the information of numerically closest node Ids, which is used in the ring routing when the hypercube routing is failed. As mentioned in [43], sequential neighbors can make progress towards all destination identifiers and ring is the only geometry that supports sequential neighbors. Several designs such as Chord [107] and CAN [92] have incorporated sequential neighbors to improve proximity and resilience. Ganesan and Manku [34] propose optimal routing algorithms for Chord. The standard Chord routing algorithm uses edges in only one direction. These algorithms exploit the bidirectionality of edges for optimality. Given that Chord is a variant of the hypercube, the optimal routes possess a surprising combinatorial structure.

Constant-Degree Structured P2P

Contrast variable-degree P2P, in which the routing table size is increased with the growing number of nodes in the system, constant-degree DHTs let each node maintain the same size routing table no matter how large the system scales. As a result, storage space is saved and the bookkeeping required to respond to system membership changes remains small, enabling the network to scale to a large system.

- *Ring-based.* Inspired by Kleinberg's small worlds [55], Symphony [73] is proposed. It arranges all nodes along a ring. Each node manages the ring segment between its own id and it predecessor and it has two short links with its immediate neighbors and k long distance links. With k=$O(1)$, the lookup path length is proved as O($\frac{1}{k} \log^2$ n).

- *Mesh-based.* Kleinberg *et al.* [55] construct a two-dimensional grid where every point maintains four links to each of its closest neighbors and just one long distance link to a node chosen from a suitable probability function. They show that small world can route a message by greedy routing in $O(\log^2 n)$ hops with constant degree. Barriere *et al.* [5] study Kleinberg's construction and prove its optimality under certain conditions.

- *Butterfly-based.* Viceroy [72] maintains a connection graph with a constant-degree logarithmic diameter, approximating a butterfly network. Each Viceroy node in butterfly level l has 7

links to its neighbors, including pointers to its predecessor and successor pointers in a general ring, pointers to the next and previous nodes in the same level ring, and butterfly pointers to its left, right nodes of level $l+1$, and up node of level $l-1$, depending on the node location. In Viceroy, every participating node has two associated values: its identity $\in [0,1)$ and a butterfly level index l. The node id is independently and uniformly generated from a range $[0,1)$ and the level is randomly selected from a range of $[1, \log n_0]$, where n_0 is an estimate of the network size. The node id of a node is fixed, but its level may need to be adjusted during its life time in the system. Viceroy routing involves three steps: ascending to a level 1 node via up links, descending along the down link until a node is reached with no down links, and traversing to the destination via the level ring or ring pointers. Viceroy takes $O(\log n)$ hops per lookup request.

Xu *et al.* [115] study the fundamental tradeoff between the routing table size and the network diameter and prove that the Tapestry, Pastry, Chord and CAN schemes are indeed asymptotically optimal as uniform algorithms. They found that O($\log$ n) is the routing table size threshold that separates the tradeoff region dominated by congestion and the region dominated by reachability for uniform algorithms. They proposed a graph based on a modified static butterfly which can achieve $O(\frac{\log n}{\log \log n})$ and $O(\log_d n)$ lookup cost when the routing table size is $\log n$ and d respectively.

- *Random graphs-based.* Pandurangan *et al.* [86] propose a random DHT graph with a constant degree and logarithmic diameter; however, the paper does not provide an efficient routing algorithm for the proposed structure that can deterministically explore the low diameter of the graph.

- *De Bruijn graph-based.* Koorde [52] combines Chord with de Bruijn graphs. Like Viceroy, it looks up a key by contacting $O(\log n)$ nodes with $O(1)$ neighbors per node. As in Chord, a Koorde node and a key have identifiers that are uniformly distributed in a 2^d identifier space. A key k is stored at its successor, the first node whose id is equal to or follows k in the identifier space. Node $2^d - 1$ is followed by node 0. Due to the dynamic nature of the P2P systems, they often contain only a few of the possible 2^d nodes. To embed a de Bruijn graph on a sparsely populated identifier ring, each participating node maintains knowledge about its successor on the ring and its first de Bruijn node. To look up a key k, the Koorde routing algorithm must find the successor of k by walking down the de Bruijn graph. Since the de Bruijn graph is usually incomplete, Koorde simulates the path taken through the complete de Bruijn graph, passing through the immediate real predecessor of each imaginary node on the de Bruijn path. Koorde can also be a variable-degree P2P. With $O(\log n)$ neighbors per node, it need $O(\log n / \log \log n)$ hops per lookup request.

  The Distance Halving DHT network proposed in [80] is isomorphic to the r-dimensional de Bruijn graph. The network is a discretization of a continuous graph based on a dynamic decomposition of the underlying Euclidean space into cells where each node is responsible for a cell. It consists of 2^r nodes and 2^{r+1} directed edges, and each node has 2 out-degree and 1 in-degree. Its routing has two phases, the first phase is routing to an almost random destination, the second phase is from the random destination to the target. It allows logarithmic routing and load, while preserving constant degrees. In a continuous graph with the edges: $f_i(y) =$

$\frac{y}{c} + \frac{i}{c}$ (i=0, 1,...,c-1), when c=$\log n$ or c=n^{ε}(ε is a constant) the lookup path length can be reduced to $O(\frac{\log n)}{\log \log n})$ or $O(1)$.

Another network structure based on de Bruijn graph is D2B [31]. Its lookup cost is $O(\log n)$ with $O(1)$ degree. The expected number of keys managed by a node of an n-node D2B network is $|K|/n$ (K is node representation binary string length), and is, with high probability, at most O($|K| \log n/n$). It achieves a trade-off between the latency for joining or leaving the network, and the latency of a lookup. Also, a large node degree increases the connectivity of the network, and thus its robustness against processor crashes.

- *Comb-based.* Considine *et al.* [19] develop Chord to constant degree comb structure with logarithmic time searches. It has much similarities with Viceroy. In the minimal topology, there are all $m \cdot 2^m$ nodes which are grouped with rank. Each node of rank i has two links, short pointer points to the node's clock-wise closest neighbor on the circle and another is called jump pointer which points to the node of rank i that is 2^i groups away moving clock-wise. In the searching, the jump pointer is followed if it does not exceed the target, otherwise, the short pointer is followed. More links can be added for maintenance and robustness purpose. However, this comb structure needs more updates when nodes join and leave, which is within $O(\log^2 n)$ hops and it needs to estimate network size to properly construct the structure.

- *CCC-based.* Cycloid is a combination of Chord and Pastry. It emulates a Cube-Connected-Cycles (CCC) graph in the routing of lookup requests between the nodes. We will explain the details in Section 1.3.5.

- *Hybrid structured-based.* Like variable-degree P2P system, some systems are actually hybrid structure-based in a more accurate sense. For example, Koorde, Viceroy and Cycloid, they are actually the combination of the ring and their own main topology. All systems regard all nodes on a ring, and each node keeps its predecessor and successor indices on the ring.

We summarize the architectural characteristics of various P2P systems in Table 1.2. Their topological properties in variable-degree and constant-degree P2P systems respectively are shown in Table 1.3.

1.3.4 Hybrid P2P

So far, we have discussed unstructured and structured P2P systems. Another class of P2P is hybrid network which is the combination of unstructured and structured P2P systems. It brings together the advantages of unstructured and structured P2P systems, and avoids their disadvantages, to achieve higher scalability.

Recently proposed Yappers [35] is an example of hybrid network. It has no explicit control of the overlay network. Its key space is partitioned into a small number of buckets, in which the DHTs are used, and intelligent forwarding is used outside the buckets. Compared to Gnutella, its

Table 1.2: A Comparison of representative DHTs; d=$\log$ n in CAN and n=$d \cdot 2^d$ in Cycloid.

Systems		Base Network	Lookup Cost	Routing Table Size
	Pastry [96]	hypercube	*O(*$\log$ *n)*	*O(*$\lvert L \rvert$*)+ O(*$\lvert M \rvert$*)+O(*$\log$ *n)*
Variable-degree	Tapestry [124]	hypercube	*O(*$\log$ *n)*	*O(*$\log$ *n)*
	Chord [107]	ring	*O(*$\log$ *n)*	*O(*$\log$ *n)*
	CAN [92]	mesh	*O(*$dn^{1/d}$*)*	*O(d)*
	eCAN [118]	mesh	*O(*$\log$ *n)*	*O(d)*
	Kademlia [75]	tree	*O(*$\log$ *n)*	*O(*$\log$ *n)*
	PRR [88]	tree	*O(*$\log$ *n)*	*O(*$q \log^2$ *n)*
	ZIGZAG [109]	tree	*O(*$\log_k$ *n)*	k^2
	Ulysses [115]	butterfly	$O(\frac{\log n)}{\log \log n})$	$\log n$
	Censorship Resistant P2P [30]	butterfly	$\log n$	$\log n$
	Dynamically Fault-Tolerant CAN [98]	butterfly	$\log n$	$\log^3 n$
	$\mathbb{H}$-graph [59]	random Graph	*O(*$\log$ *n)*	*O(*$\log$ *n)*
	Koorde [52]	de Bruijn	*O(*$\log$ *n/*$\log\log$ *n)*	O($\log$ n)
Constant-degree	Symphony [73]	ring	*O(*$\frac{1}{k} \log^2$ *n)*	*k=O(1)*
	Small Worlds [55]	mesh	*O(*$\log^2$ *n)*	*O(1)*
	Viceroy [72]	butterfly	*O(*$\log$ *n)*	7
	Low-Diameter [86]	random Graph	*O(*$\log$ *n)*	*O(1)*
	Distance Halving [80]	de Bruijn	*O(*$\log_d$ *n)*	*O(d)*
	Koorde [52]	de Bruijn	*O(*$\log$ *n)*	$\geq$2
	D2B [31]	de Bruijn	*O(*$\log_d$ *n)*	*O(1)*
	Constant State Indexing based on Chord [19]	comb	*O(*$\log$ *n)*	$\geq$2
	Cycloid [101]	CCC	*O(d)*	7

Table 1.3: Structured P2P systems summary.

Base Network	Variable-degree P2P	Constant-degree P2P
hypercube	Pastry [96] Tapestry [124]	
ring	Chord [107]	Symphony [73]
mesh	CAN [92] eCAN [118]	Small Worlds [55]
tree	Kademlia [75] PRR [88] ZIGZAG [109]	
butterfly	Ulysses [115] Censorship Resistant P2P [30] Dynamically Fault-Tolerant CAN [98]	Viceroy [72]
random Graph	$\mathbb{H}$-graph [59]	Low-Diameter [86]
de Bruijn	Koorde [52]	Distance Halving [80] Koorde [52] D2B [31]
comb		Constant State Indexing based on Chord [19]
CCC		Cycloid [101]

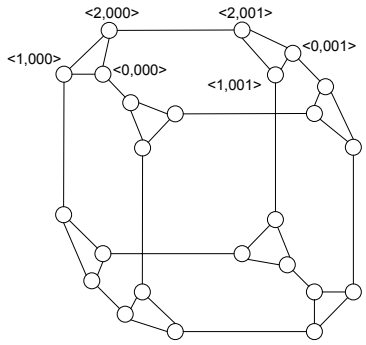


Figure 1.2: A 3-dimensional Cube-Connected-Cycles.

lookup cost is reduced by an order of magnitude. Freedman *et al.* [33] propose a DHT based on distributed tries. It employs dynamic-assignment approach, in which peers replicate lookup structure partitions that they frequently access and organize those partitions as a trie. The distributed trie converges to an accurate network map over time. Its lookup algorithm shares many similarities with the algorithm proposed in [88]. The lookup node first check its local storage for the value corresponding to the lookup key. If not present, then it initiates a distributed lookup process.

1.3.5 Cycloid: An Example of Structured P2P DHTs

Cycloid [101] is a constant-degree structured P2P overlay network, which is part of authors' work. It combines Pastry with CCC graphs. In a Cycloid system with $n = d \cdot 2^d$ nodes, each lookup takes $O(d)$ hops with $O(1)$ neighbors per node. Like Chord, it employs consistent hashing to map keys to nodes. A node and a key have identifiers that are uniformly distributed in a $d \cdot 2^d$ identifier space.

Structure and Key Allocation. A d-dimensional CCC graph is a d-dimensional cube with replacement of each vertex by a cycle of d nodes. It contains $d \cdot 2^d$ nodes of degree 3 each. Each node is represented by a pair of indices $(k, a_{d-1}a_{d-2}\ldots a_0)$, where k is a cyclic index and $a_{d-1}a_{d-2}$.....a_0 is a cubical index. The cyclic index is an integer, ranging from 0 to $d-1$ and the cubical index is a binary number between 0 and 2^d-1. Figure 1.2 shows the 3-dimensional CCC. In a Cycloid system, each node keeps a routing table and two leaf sets with a total of 7 entries to maintain its connectivity to the rest of the system. Table 1.4 shows a routing state table for node (4,10111010) in an 8-dimensional Cycloid, where x indicates an arbitrary binary value, inside leaf set maintains the node's predecessor and successor in the local cycle, and outside leaf set maintains the links to the preceding and the succeeding remote cycles.

In general, a node $(k, a_{d-1}a_{d-2}\ldots a_k\ldots a_0)$ ($k \neq 0$) has one cubical neighbor $(k-1, a_{d-1}a_{d-2}\ldots \overline{a}_k xx...x)$ where x denotes an arbitrary bit value, and two cyclic neighbors $(k-1, b_{d-1}b_{d-2}\ldots b_0)$ and

Table 1.4: Routing table of a Cycloid node (4,101-1-1010).

NodeID(4,101-1-1010)	
Routing table	
cubical neighbor: (3,101-0-xxxx)	
cyclic neighbor: (3,101-1-1100)	
cyclic neighbor: (3,101-1-0011)	
Leaf Sets (half smaller, half larger)	
Inside Leaf Set	
(3,101-1-1010)	(6,101-1-1010)
Outside Leaf Set	
(7,101-1-1001)	(6,101-1-1011)

$(k-1, c_{d-1}c_{d-2}\ldots c_0)$. The cyclic neighbors are the first larger and smaller nodes with cyclic index $k-1$ mod d and their most significant different bit with the current node is no larger than $k-1$. That is,
(k-1, $b_{d-1}\ldots b_1b_0$) = min$\{\forall$(k-1, $y_{d-1}\ldots y_1y_0$)$|y_{d-1}\ldots y_0 \geq a_{d-1}\ldots a_1a_0\}$
(k-1, $c_{d-1}\ldots c_1c_0$) = max$\{\forall$(k-1, $y_{d-1}\ldots y_1y_0$)$|y_{d-1}\ldots y_0 \leq a_{d-1}\ldots a_1a_0\}$
The node with a cyclic index $k=0$ has no cubical neighbor and cyclic neighbors. The node with cubical index 0 has no small cyclic neighbor, and the node with cubical index 2^d-1 has no large cyclic neighbor. The nodes with the same cubical index are ordered by their cyclic index mod d on a local cycle. The left inside leaf set node points to the node's predecessor and the right inside leaf set node points to the node's successor in the local cycle. The largest cyclic index node in a local cycle is called the primary node of the local cycle. All local cycles are ordered by their cubical index mod 2^d on a large cycle. The left outside leaf set node points to the primary node in the node's preceding remote cycle and the right outside leaf set node points to the primary node in the node's succeeding remote cycle in the large cycle. Our connection pattern is resilient in the sense that even if many nodes are absent, the remaining nodes are still capable of being connected. The routing algorithm is heavily assisted by the leaf sets. The leaf sets help improve the routing efficiency, check the termination condition of a lookup, and wrap around the key space to avoid the target overshooting. How the routing table and leaf sets are initialized and maintained will be explained later. The Cycloid DHT assigns keys onto its id space by the use of a consistent hashing function. For a given key, the cyclic index of its mapped node is set to its hash value modulated by d and the cubical index is set to the hash value divided by d. If the target node of a key's id $(k, a_{d-1}\ldots a_1a_0)$ is not a participant, the key is assigned to the node whose id is first numerically closest to $a_{d-1}a_{d-2}\ldots a_0$ and then numerically closest to k.

Cycloid Routing Algorithm. Cycloid routing algorithm emulates the routing algorithm of CCC [90] from source node $(k, a_{d-1}\ldots a_1a_0)$ to destination $(l, b_{d-1}\ldots b_1b_0)$, incorporating the resilient connection pattern of Cycloid. The routing algorithm involves three phases, assuming MSDB be the most significant different bit of the current node and the destination.

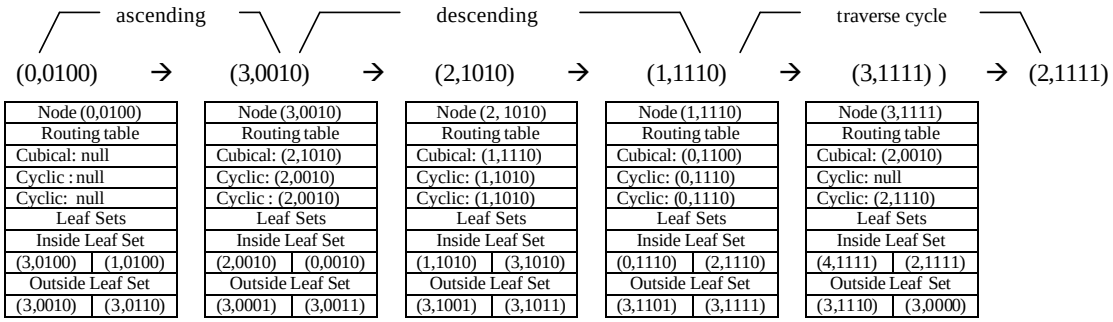


Figure 1.3: An example of routing phases and routing table states in Cycloid.

1. *Ascending*: When a node receives a request, if its k<MSDB, it forwards the request to a node in the outside leaf set sequentially until cyclic index k≥MSDB.
2. *Descending*: In the case of k≥MSDB, when k=MSDB, the request is forwarded to the cubical neighbor, otherwise the request is forwarded to the cyclic neighbor or inside leaf set node, whichever is closer to the target, in order to change the cubical index to the target cubical index.
3. *Traverse cycle*: If the target Id is within the leaf sets, the request is forwarded to the closest node in the leaf sets until the closest node is the current node itself.

Figure 1.3 presents an example of routing a request from node (0,0100) to node (2,1111) in a 4-D Cycloid DHT. The MSDB of node (0,0100) with the destination is 3. As (0,0100) cyclic index k=0 and k<MSDB, it is in the ascending phase. Thus, the node (3,0010) in the outside leaf set is chosen. Node (3,0010)'s cyclic index 3 is equal to its MSDB, then in the descending phase, the request is forwarded to its cubical neighbor (2,1010). After node (2,1010) finds that its cyclic index is equal to its MSDB 2, it forwards the request to its cubical neighbor (1,1110). Because the destination (2,1111) is within its leaf sets, (1,1110) forwards the request to the closest node to the destination (3,1111). Similarly, after (3,1111) finds that the destination is within its leaf sets, it forwards the request to (2,1111) and the destination is reached.

Each of the three phases is bounded by *O(d)* hops, hence the total path length is *O(d)*. The key idea behind this algorithm is to keep the distance decrease repeatedly. Also, the routing algorithm can be easily augmented to increase fault tolerance. When the cubical or the cyclic link is empty or faulty, the message can be forwarded to a node in the leaf sets.

Self-Organization. Cycloid deals with node joining and leaving in a distributed manner, without requiring hash information to be propagated through the entire network. When a new node joins, it needs to initialize its routing table and leaf sets, and inform other related nodes of its presence, which will update their neighbors. Before a node leaves, it needs to notify its inside leaf set nodes. The need to notify the nodes in its outside leaf set depends on whether the leaving node is a primary node. Upon receiving a leaving notification, the nodes in the inside and outside leaf

sets update themselves. In addition, the nodes in the outside leaf set need to notify other nodes in their local cycle one by one, which will take at most d steps. Updating cubical and cyclic neighbors are the responsibility of system stabilization, as in Chord. Undoubtedly, low degree P2P networks perform poorly in failure-prone environments, where nodes fail or depart without warning. Usually, the system maintains another list of nodes to handle such problems, such as the successor list in Chord [107] and the bucket in Viceroy [72]. In our proposal, we assume that nodes must notify others before leaving, as the authors of Koorde argued that the fault tolerance issue should be handled separately from routing design.

Simulation results show that Cycloid delivers a higher location efficiency in the average case, it has a much shorter path length per lookup request in the average case than Viceroy and Koorde, another two constant degree overlay networks. Cycloid distributes keys and query load more evenly between the participating nodes than Viceroy. Also, Cycloid is more robust as it continues to function correctly and efficiently with frequent node joins and leaves.

1.4 Security Concerns

Currently, most of the P2P systems assume that the nodes involved are cooperative or even trustful. However, because of P2P systems' openness feature and decentralization, it is not desirable to constrain the membership of a P2P system. As a result, P2P systems are faced with threats such as denial of service, masquerading and tampering. In these situations, the system must be able to operate even though some participants are malicious. In the following, we will explain security issues in P2P systems from four aspects: authenticity, anonymity, attacks and secure routing.

1.4.1 Authenticity

Searching for data on a P2P network is a straight forward process. A node on the network issues a request for a file, and the request propagates the network until nodes who have that file are found. Finally, these nodes send back a reply informing they have the file, and the file transfer can begin. While this might seem simple, there is an important question that complicates this process. How can a node know it is downloading the exact file it was searching for? This file could have been altered or be a different file altogether.

There are a number of definitions for an authentic file, and a number of approaches to verify authenticity. If we view an authentic file as being the original file on the network, then a time-stamping system would be helpful in determining which of the search results is the oldest file available. An example is Mammoth [27], a file system which presents objects with unique identifiers made up of the creator's node address and a timestamp. Another is PAST [97] which assigns unique identifiers to files and storage nodes. These identifiers are used to determine which nodes will store replicas of the file. If we were to have trusted nodes which kept signatures of

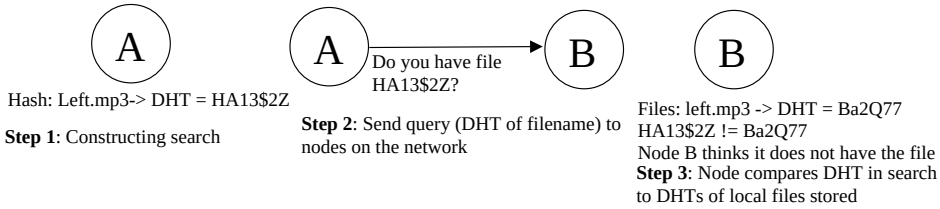


Figure 1.4: An example of a DHT file search.

files uploaded by users, nodes would be able to compare signatures of files they downloaded with signatures on these servers. Identical signatures would verify a file is identical to the original file uploaded and has not been modified. An example of such a signature is the MD5 checksum which creates a virtually unique signature for every file. Even the smallest modification to the file would create a completely different signature and the change would be detected. Finally, the reputation based approach, in which nodes deemed more trustworthy than others have a bigger influence on the decision of a query. This type of approach would require the system to maintain and propagate node reputation throughout the network. There has been significant research in this area, such as deploying a reputation system for the Gnutella network [45]. There are, however, many inherent problems with reputation systems on P2P networks. For example, nodes might not be eligible to vote on other nodes. This could be because of their location on the network or the tasks that they are required to perform. This issue has been addressed by introducing the need for different types of reputations for different scenarios on the same network [113]. Another proposal has been to add reputation policies which would determine the pairs of nodes eligible to interact [108].

Since P2P networks consist of nodes which can join the network without being verified as legitimate, these nodes can not be trusted to have authentic data or to make a valid vote. Also, since P2P networks have taken a decentralized approach, there is no central main node which can be used to verify nodes or files. A solution to these problems was created, however, it is not without its drawbacks. P2P programs such as Freenet [15, 16] attempt to answer these problems by introducing a DHT. This is a unique hash made, in this case, of a file name. Unlike other P2P programs such as Kazaa [54] or Gnutella [38] where a search is constructed of a partial or complete file name, a file search on Freenet is composed of a DHT. A user searching for a file on the network would have to know its exact name in order to compose the correct DHT. If a file name is different by even one character, hashing it would result in a different DHT and the search would be unsuccessful. Figure 1.4 illustrates an example of a DHT file search where only one letter is different by being uppercase. This search is unsuccessful because the DHT constructed does not match the original.

A few question arise about Freenet and its DHT approach.

(1) How can a user know the exact name of a file if he or she can not search for it to begin with?

(2) By using a DHT search, isn't the user missing search results of identical files but with slightly different names?

(3) What are the benefits to using a restrictive DHT search instead of a standard name search?

The answers to these questions lie with the solution Freenet has developed on how to find files on its network. In order to publicize files on Freenet, users publish SHA hashes on their Freenet sites. Users retrieve this information from these sites and search for the acquired hash on the network. This ensures that the user is downloading the exact file requested, even though other copies of the file named differently could exist. Since the download is limited to a file a user has information about, the two main authenticity issues are resolved. First, tampering with data in the file could easily be detected by adding a file integrity hash to the site the file is published on. Secondly, file renaming, where a different file is named like the one a user is searching for, can also be easily detected by publishing file integrity hashes. This hash could be compared to the hash of a file just downloaded to verify its authenticity. The clear benefits of this approach are that since the user will have identifying information about the file being downloaded, it will be simple to assure it is authentic. On the other hand, performance considerably degrades since nodes that do have this file under a different name will not return positive search results. A more important feature disabled by this approach is the partial name search. Having to know the exact file name beforehand makes it more difficult to find a file. Since nodes are not authenticated, and there is no authoritative entity to verify data, so far this has been the only completely successful approach to circumvent these issues.

1.4.2 Anonymity

Anonymity has been an important issue in P2P networks. Most implementations such as Kazaa, Napster or Gnutella offer no anonymity and have brought forth numerous lawsuits by companies seeking to protect their copyright material. The main focus of anonymous peer networks is not only to allow sharing or retrieving of any data desired, but also to allow individuals who live under internet-censure laws to freely browse the internet and express their ideas.

There are different ways to achieve anonymity on a network, depending on the user's actions [32]. A user who wishes to upload or share a file but remain anonymous, will need to use a peer network that keeps the originating IP address or node ID hidden. Freenet [16] is one such example of an anonymous P2P program that ensures author anonymity, while Free Haven [26] provides server anonymity. Once a file is available on the network, users may want to retrieve it anonymously. Nodes retrieving a file will need to have their IP addresses or node IDs hidden as well.

Many P2P programs increase file availability by replicating files on various nodes on the network. This can be done by path replication [16], where copies of files are stored along the path they are retrieved, or by replicating files based on calculations such as distance [7]. This brings forward another form of anonymity, one that protects a user from its own data. Since a user's P2P program might unknowingly be replicating data that is illegal, such as copyrighted material, it is necessary to protect that user from being held responsible for it. This can be accomplished by encrypting

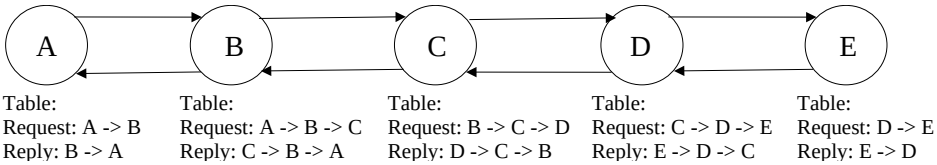


Figure 1.5: A tunnelled communication between nodes A and E.

the data stored locally on a node and only allowing the P2P program to decrypt it when it is being accessed [7, 111]. Another solution would be to split the file into chunks and allow a node to only store a part of the file. This, combined with data encryption would prevent a user from tampering with data it is replicating on the network.

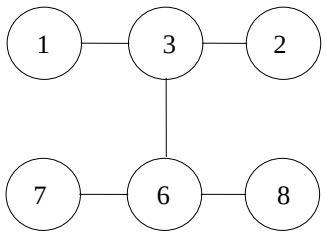
In order to hide the source and destination of a communication on a P2P network it is necessary to form an anonymous path between the two nodes and ensure that nodes on the path do not know where the endpoints are. This is implemented with an overlay network that restricts a node's view only to its neighbors. A node on a message path knows only the previous and next step in the path, two of its neighbors. Figure 1.5 depicts a tunnelled communication between nodes A and E. The path of the communication is not directly between the two nodes, rather it travels through three other nodes. Each node on the path does not know of the source and final destination of the messages transferred. For example, node C does not know if the communication is between nodes B and D or if they are passing the messages elsewhere.

Providing anonymity on the network has proven to affect network efficiency. The amount of messages on the network increases significantly since extra nodes are used to relay messages instead of using direct communication. And most importantly, download and upload speeds decrease since data must be routed through a prolonged path.

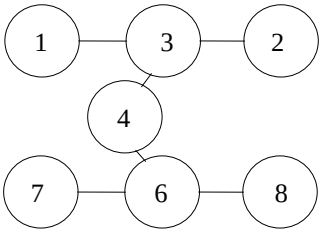
1.4.3 Attacks on the network

The lack of security on P2P networks has not gone unnoticed. Many attacks have been introduced in the last few years, most of which deny access to the network. These range from being aimed at a single node, to attacks which can cause a large branch of the network to fail. The three most powerful attacks will be introduced here, Sybil, Byzantine, and DOS attacks.

The Sybil attack [28] is named after Sybil Dorsett who suffered from a multiple personality disorder. This attack involves one node on the network which masquerades as multiple identities. This can lead to many situations which are harmful to the network. With this attack it would be possible to actively monitor and control communication between nodes. A malicious node can impersonate all the different routes to another node and control all of its communication. Even worse, a malicious node can segment off a part of the network if it positions itself accordingly. Figure 1.6 shows an example of such an attack. In this example, node 4 controls communication



(a). An example of a partial mesh. Nodes 3 and 6 bridge the two parts of the network.



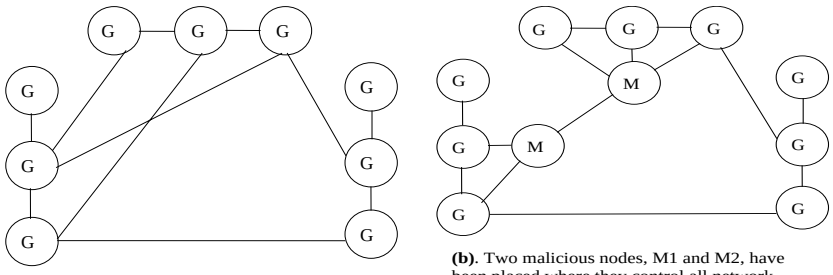
(b). Attacker (node 4) positions itself between nodes 3 and 6. Each assumes node 4 has been placed legitimately.

Figure 1.6: An example of a Sybil attack.

between two network segments. It can monitor, alter or disrupt communication passing between the segments, and can even separate them effectively disabling all communication between them. A more direct attack involves masquerading as a node and performing actions on its behalf. If illegal actions are undertaken, such as download of illegal content, an innocent user can be prosecuted.

Another form of attack capable of disrupting a node's network communication is called a Denial of Service attack. A node can cause a DOS attack against another in a number of ways. The traditional DOS attack involves the malicious node sending messages to the victim at a high rate. On P2P networks this is accomplished by forwarding all requests a malicious node receives to the victim node. However, today with available high speed internet and fast processing speeds at the node, such attacks originating from only one node do not generate messages fast enough to cause a real threat. The victim node will most likely be able to process the incoming messages before they fill up the message queue and cause legitimate messages to be dropped. A more efficient way to successfully construct a DOS attack is for a malicious node to reply to every request it receives stating the victim node as a source for the file. Therefore, all the nodes whose queries reach the malicious one, will be directed to contact the victim for files it does not have. On large P2P networks such as the ones deployed today on the internet, thousands of nodes query the network every few minutes. If the victim receives a large number of requests in a short time, it will be bombarded with more messages than it can process, and will be unable to send or receive legitimate communication.

Finally, the byzantine attack brings forward the problem of reaching a consensus among nodes when an unknown number of them may be traitors [58]. In a P2P network where decisions can be based on neighboring nodes' responses, it is important to act on the legitimate ones. For example, P2P implementations that use a voting system to decide from which node to download a file, or those that determine the message path based on nodes' input of their network load. In these cases, decisions based on data sent from malicious nodes can be devastating to the network. The byzantine attack involves multiple nodes collaborating to disrupt the network. This attack is more difficult to set up than the sybil attack, because multiple nodes must be set up on the network instead of one. The malicious possibilities a byzantine attack can cause are similar to those a sybil attack can perform. However, after malicious nodes have been placed on the network, byzantine attacks becomes significantly more powerful and are easier to execute than the sybil attack. Fig-



(a). 3 groups partially connected to each other.

(b). Two malicious nodes, M1 and M2, have been placed where they control all network communication between groups G1 and G2.

Figure 1.7: An example of a Byzantine attack.

ure 1.7 illustrates a byzantine attack. A sybil attack would not be able to segment the network efficiently since one node is not present in different segments of the network. In this attack, however, the malicious user has placed nodes in different segments of the network, taking control of each one. There has been significant work in solving the byzantine problem. One method proposed an algorithm for byzantine fault tolerant (BFT) state machine replication [13] which ensures that eventually clients receive correct replies. Another approach separates agreement from execution on a BFT state machine [123]. Agreement is what orders requests and execution is what processes requests, these have both been coupled together in other BFT state machine architectures.

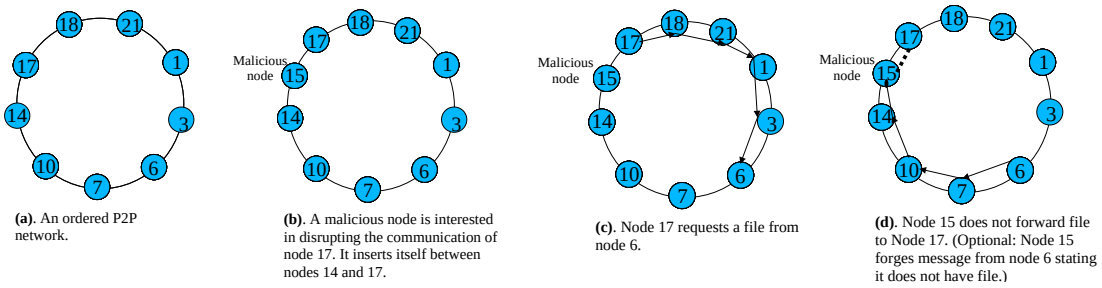
Since P2P networks are insecure by design, these attacks have remained an open problem. The solutions discovered by research so far have not managed to eliminate the threat, only to minimize it.

1.4.4 Secure routing

Secure routing implementations are necessary to ensure that a message will be delivered despite malicious node activity such as message corruption or misrouting. There are three crucial tasks a P2P protocol must implement securely in order to ensure node connectivity [10].

First, node IDs must be securely assigned to nodes on the network. This will prevent attackers from choosing node IDs. If a malicious node were to choose its node ID, it could choose where to place itself on the network. This is especially dangerous in P2P networks using ordered IDs [107]. Attackers placing malicious nodes in strategic places on the network can control traffic of messages and data, connections can be manipulated or blocked altogether. Figure 1.8 shows an example of malicious use of a node's ability to choose its location on the network. This attack can be prevented if node IDs are related to the node's IP address [104], as is the case with Pastry [96]. The assumption is that a malicious node can not gain a large number of diverse addresses and position itself in multiple places on the Internet.

Another task that must be securely implemented is the constant updates to the routing table. This involves ensuring that the fraction of faulty nodes in the table will be lower than the fraction



(a). An ordered P2P network.

(b). A malicious node is interested in disrupting the communication of node 17. It inserts itself between nodes 14 and 17.

(c). Node 17 requests a file from node 6.

(d). Node 15 does not forward file to Node 17. (Optional: Node 15 forges message from node 6 stating it does not have file.)

Figure 1.8: Malicious use of a node's ability to choose its location on the network.

of faulty nodes in the network. A large number of faulty nodes inserted in the routing table can cause nodes to depend on a route which consists of malicious nodes.

Finally, message propagation must be securely implemented. Messages must be able to reach nodes which will further replicate them on the network. It is not required for all copies of a message to reach message-replicating nodes, only a minimum of one is required. Although the rate of replication and amount of results will be lower, if a node's routing table is poisoned with invalid entries, the desire is not to keep the connection efficient, it is to keep it connected.

Assigning node IDs securely without the use of a central authority remains an open problem. There have been attempted solutions but they have not been successful. First, an introduction of cryptographic puzzles [76] which need to be solved before a node can join. This proposal only slows down the malicious user from connecting, it does not prevent him from doing so. Assuming a malicious user would obtain enough resources to solve the puzzle faster, this would be ineffective altogether. Also, since connecting to the network is done anonymously, a node can connect as many times as it wants, attempting to connect as many times as it wishes. Another proposal suggested sending a message on the network and waiting for a reply, confirming the route is active and correct. However, since malicious nodes can act as intermediaries of a communication they can choose to forward the confirmation message untouched while modifying the remaining data transferred. Also, a malicious node can masquerade as the destination node and confirm the route is correct. This would be more difficult to accomplish if nodes would have predetermined keys to encrypt the communication. However, it is unrealistic to require every communication to use a unique predetermined key. P2P networks are dynamic, searching and retrieving files from nodes on the network requires no previous communication or prior knowledge of them. All solutions that exist to encrypt the communication or verify file integrity with checksums are prone to failure. A malicious node on the path can intercept the key exchange in the beginning of the communication and modify the checksum as well as the data being transferred. Since central authorities on today's decentralized P2P networks are not a feasible option, a possible solution is to use PGP keys available on public servers worldwide. This would require all users on the network to create such keys and keep them secure. The main problem with this solution, however, is that user anonymity will not be available.

1.5 Concluding Remarks

In this chapter, we have presented an overview of P2P overlay networks with a focus on scalability and security issues. P2P systems can be classified into structured, unstructured, and hybrid classes. Representatives of the systems and related scaling techniques in each class are reviewed. P2P systems provide security challenges in four aspects: authenticity, anonymity, attacks, and secure routing. The requirements and current security measures are discussed.

We conclude this chapter with discussion about a number of open issues for building P2P cooperative computing applications.

- **Heterogeneity.** With the increasing emergence of various end devices equipped with networking capability, coupled with the diverse network technology development, the heterogeneity of participating peers of a practical wide-deployed P2P system is pervasive in the Internet domain. Their instinct properties, including computing ability, differ a lot and deserve serious consideration for the construction of a real efficient wide-deployed application.

- **Data integrity.** Due to the heterogeneity of end devices, we envision that the transformation is a very likely happening in any distributed cooperative applications, including image transcoding, personalized Web pages, etc. By doing so, although it is easy to provide data integrity for static files, for dynamic generated content (e.g., dynamic Web content by content assembly [102], transcoded content by adaptation [41]), it is really a challenge for both content providers and clients to validate the correctness of the content. By taking advantage of other peers' effort to leverage the content (including transcoded content, dynamic pages) transferring burden, we realize that the data integrity becomes a more and more serious problem.

- **Application-specific behavior.** This is neglected in most existing applications, but we believe it will play an important role in the future. In a distributed cooperative application, only peers showing application-specific interest would carry out the application-specific actions. Otherwise, there is no obligation for peers to serve the heavy burden of those un-interested applications. For example, to build an application-layer multicast tree, asking a peer that is not interested in the specific multicast streaming content, to contribute its computing power to help the construction of the multicast tree seems reasonable. However, routing the high bandwidth streaming content is too high a price for that peer. Therefore, we envision that application-specific behavior should be considered in partner discovery.

- **Proximity.** Mismatch between logical proximity abstraction, derived from DHTs algorithms, and physical proximity information in reality are a big obstacle for the deployment and performance optimization issues for P2P applications.

- **Reconfiguration semantics and overhead.** Using dynamic reconfiguration to adapt to dynamic environment changing and/or peer failures has been widely proposed in several systems [12, 116, 125]. However, few of these previous work takes the reconfiguration over-

head into consideration. We argue that the dynamic reconfiguration should consider the application-related semantics and further the overhead of reconfiguration as well.

References

[1] K. Aberer, P. Cudrè-Mauroux, and M. Hauswirth. The chatty web: Emergent semantics through gossiping. In *Proc. of the 12th International World Wide Web Conference*, 2003.

[2] L. A. Adamic, B. A. Huberman, R. M. Lukose, and A. R. Puniyani. Search in power law networks. In *Physical Review E*, volume 64, 2001. 46135-46143.

[3] Aimster. http://computer.howstuffworks.com/question587.htm.

[4] J. Aspnes, Z. Diamadi, and G. Shah. Fault-Tolerant Routing in Peer-to-Peer Systems. In *Proc. of the 12th ACM Symposium on Principles of Distributed Computing (PODC)*, 2002.

[5] L. Barriere, P. Fraigniaud, E. Kranakis, and D. Krizanc. Efficient routing in networks with long range contacts. In *Proc. of the 15th International Conference on Distributed Computing (DISC)*, pages 270–284, 2001.

[6] P. A. Bernstein, F. Giunchiglia, A. Kementsietsidis, J. Mylopoulos, L. Serafini, and I. Zaihrayeu. Data management for peer-to-peer computing: A vision. In *Proc. of the 5th International Workshop on the Web and Databases*, 2002.

[7] S. Blackheath. The grapevine project. http://grapevine.sourceforge.net/.

[8] G. Bolcer. Magi: An architecture for mobile and disconnected workflow. In *White paper (www.endeavors.com)*, 2001.

[9] M. Castro, M. Costa, and A. Rowstron. Performance and dependability of structured peer-to-peer overlays. Technical report, Microsoft Research, 2003.

[10] M. Castro, P. Druschel, A. Ganesh, A. Rowstron, and D. S. Wallach. Secure routing for structured peer-to-peer overlay networks. In *Proc. of the 5th USENIX Symposium on Operating Systems Design and Implementation*, December 2002.

[11] M. Castro, P. Druschel, Y. C. Hu, and A. Rowstron. Topology-aware routing in structured peer-to-peer overlay networks. In *Future Directions in Distributed Computing*, 2003.

[12] M. Castro, P. Druschel, A. Kermarrec, and A. Rowstron. SCRIBBLE: A large-scale and decentralized application-level multicast infrastructure. *IEEE Journal on Selected Areas in Communications*, 20(8), 2002.

[13] M. Castro and B. Liskov. Practical byzantine fault tolerance. *Proc. of the 3rd Symposium on Operating Systems Design and Implementation*, pages 173–186, 1999. USENIX Association.

[14] Y. Chawathe, S. Ratnasamy, L. Breslau, N. Lanham, and S. Shenker. Making gnutella-like p2p systems scalable. In *Proc. of ACM SIGCOMM*, 2003.

[15] I. Clarke, T. W. Hong, S. G. Miller, O. Sandberg, and B. Wiley. Protecting free expression online with freenet. *IEEE Internet Computing*, 6(1):40–49, 2002.

[16] I. Clarke, O. Sandberg, B. Wiley, and T. W. Hong. Freenet: A distributed anonymous information storage and retrieval system. In *Proc. of the ICSI Workshop on Design Issues in Anonymity and Un-observability*, 2000. http://freenet.sourceforge.net.

[17] I. Clarke, O. Sandberg, B. Wiley, and T. W. Hong. Freenet: A distributed anonymous infor-

mation storage and retrieval system. In *Proc. Int'l Workshop on Design Issues in Anonymity and Unobservability*, pages 46–66, 2001.

[18] E. Cohen and S. Shenker. Replication strategies in unstructured peer-to-peer networks. In *Proc. of ACM SIGCOMM*, 2002.

[19] J. Considine and T. A. Florio. Scalable Peer-to-Peer Indexing with Constant State. Technical report, CS Department, Boston Universiay, 2002.

[20] B. Cooper and H. Garcia-Molina. Studying search networks with sil. In *Proc. of the 2nd International Workshop on Peer-to-Peer Systems (IPTPS)*, 2003.

[21] B. Cooper and H. Garcia-Molina. SIL: Modeling and Measuring Scalable Peer-to-Peer Search Networks, 2004. Submitted for Publication.

[22] B. F. Cooper and H. Garcia-Molina. Ad hoc, self-supervising peer-to-peer search networks. Technical report, CS Department, Stanford University, 2003.

[23] A. Crespo and H. Garcia-Molina. Routing indices for peer-to-peer systems. In *Proc. of the 22nd International Conference on Distributed Computing Systems (ICDCS)*, 2002.

[24] N. Daswani and H. Garcia-Molina. Query-flood dos attacks in gnutella. In *Proc. Ninth ACM Conference on Computer and Communications Security*, 2002.

[25] N. Daswani, H. Garcia-Molina, and B. Yang. Open problems in data-sharing peer-to-peer systems. In *In 9th International Conference on Database Theory*, 2003.

[26] R. Dingledine, M. J. Freedman, and D. Molnar. The free haven project: Distributed anonymous storage service. In *Proc. of the Workshop on Design Issues in Anonymity and Unobservability*, July 2000.

[27] B. Dmitry, B. Alex, P. Jody, G. Shihao, F. Michael, and H. Norman. Using file-grain connectivity to implement a peer-to-peer file system. In *21st IEEE Symposium on Reliable Distributed Systems (SRDS)*, 2002.

[28] J. Douceur. The sybil attack? In *Proc. of the 1st International Workshop on Peer-to-Peer Systems (IPTPS)*, 2002.

[29] Fasttrack product description, 2001. www.fasttrack.nu/index_int.html.

[30] A. Fiat and J. Saia. Censorship Resistant Peer-to-Peer Content Addressable Networks. In *Symposium on Discrete Algorithms*, 2002.

[31] P. Fraigniaud and P. Gauron. The content-addressable network d2b. Technical report, Univerisity Paris-Sud, 2003.

[32] M. J. Freedman, E. Sit, J. Cates, and R. Morris. Introducing tarzan, a peer-to-peer anonymizing network layer. In *Proc. of the 1st International Workshop on Peer-to-Peer Systems (IPTPS)*, 2002.

[33] M. J. Freedman and R. Vingralek. Efficient Peer-to-Peer Lookup Based on a Distributed Trie. *IPTPS02*, 2002.

[34] P. Ganesan and G. S. Manku. Optimal routing in chord. In *Proc. of the Symposium on Discrete Algorithms (SODA)*, 2004.

[35] P. ganesan, Q. Sun, and H. Garcia-Molina. YAPPERS: A Peer-to-Peer Lookup Service over Arbitrary Topology. In *Proc. of IEEE Conference on Computer Communications (INFOCOM)*, 2003.

[36] A. Ganesh, A. Rowstron, M. Castro, P. Druschel, and D. Wallach. Security for structured peer-to-peer overlay networks. In *Proc. of the 5th USENIX Symposium on Operating Sys-*

tems Design and Implementation, 2002.
[37] Gnutella development forum. The Gnutella Ultrapeer Proposal, 2002.
[38] Gnutella home page. http://www.gnutella.com.
[39] Gnutella2 home page. http://www.gnutella2.com.
[40] J. Gray, P. Helland, P. ONeil, and D. Shasha. The dangers of replication and a solution. In *Proc. of ACM SIGMOD*, pages 173–182, 1996.
[41] S. D. Gribble and et al. The Ninja Architecture for Robust Internet-Scale Systems and Services. *Journal of Computer Networks*, 35(4), 2001.
[42] A. Grimshaw, W. Wulf, J. French, A. Weaver, and Reynolds Jr. P. Legion: The next logical step toward a nation-wide virtual computer. Technical report, Department of Computer Science, University of Virginia, 1994.
[43] K. Gummadi, R. Gummadi, S. Gribble, S. Ratnasamy, S. Shenker, and I. Stoica. The impact of dht routing geometry on resilience and proximity. In *Proc. of ACM SIGCOMM*, August 2003.
[44] I. Gupta, K. Birman, P. Linga, A. Demers, and R. van Renesse. Kelips: building an efficient and stable p2p dht through increased memory and background overhead. In *Proc. of the 2nd International Workshop on Peer-to-Peer Systems (IPTPS)*, pages 81–86, 2003.
[45] M. Gupta, P. Judge, and M. Ammar. A reputation system for peer-to-peer networks.
[46] A. Y. Halevy, Z. G. Ives, P. Mork, and I. Tatarinov. Piazza: Data management infrastructure for semantic web applications. In *Proc. of the 12th International World Wide Web Conference*, 2003.
[47] C. Hang and K. C. Sia. Peer clustering and firework query model. In *Proc. of the 12th International World Wide Web Conference*, 2003.
[48] K. Hildrum, J. Kubiatowicz, S. Rao, and B. Y. Zhao. Distributed Object Location in a Dynamic Network. In *Proc. of ACM SPAA*, 2002.
[49] iMesh. http://www.imesh.com/.
[50] Jxta 2001. JXTA home page: www.jxta.org.
[51] F. Kaashoek. Peer-to-peer computing: a new direction in distributed computing. In *Proc. of Principles of Distributed Computing (PODC)*, 2002.
[52] M. F. Kaashoek and R. Karger. Koorde: A simple degree-optimal distributed hash table. In *Proc. of the 2nd International Workshop on Peer-to-Peer Systems (IPTPS)*, 2003.
[53] D. Karger, E. Lehman, T. Leighton, M. Levine, D. Lewin, and Panigrahy R. Consistent hashing and random trees: Distributed caching protocols for relieving hot spots on the World Wide Web. *Proc. of the 29th Annual ACM Symposium on Theory of Computing (STOC)*, pages 654–663, 1997.
[54] Kazaa, 2001. Kazaa home page: www.kazaa.com.
[55] J. Kleinberg. The small-world phenomenon: An algorithmic perspective. In *Proc. 32nd ACM Symposium on Theory of Computing*, pages 163–170, 2000.
[56] B. Krishnamurthy and J. Wang. On network-aware clustering of web clients. In *Proc. of ACM SIGCOMM*, 2000.
[57] B. Krishnamurthy, J. Wang, and Y. Xie. Early measurements of a cluster-based architecture for p2p systems. In *Proc. of SIGCOMM Internet Measurement Workshop*, 2001.
[58] L. Lamport, R. Shostak, and M. Pease. The Byzantine generals problem. *ACM Transactions*

on Programming Languages and Systems, 4(3):382–401, 1982.

[59] C. Law and K.-Y. Siu. Distributed Construction of Random Expander Graphs. In *Proc. of IEEE Conference on Computer Communications (INFOCOM)*, 2003.

[60] A. L*ö*ser, F. Naumann, W. Siberski, W. Nejdl, and U. Thaden. Semantic overlay clusters within super-peer networks. In *Proc. of the International Workshop on Databases, Information Systems and Peer-to-Peer Computing in Conjunction with the VLDB*, 2003.

[61] A. L*ö*ser, W. Nejdl, M. Wolpers, and W. Siberski. Information integration in schema-based peer-to-peer networks. In *Proc. of the 15th International Conference of Advanced Information Systems Engieering (CAiSE)*, 2003.

[62] A. L*ö*ser, M. Wolpers, W. Siberski, and W. Nejdl. Efficient data store discovery in a scientific p2p network. In *Proc. of the Workshop on Semantic Web Technologies for Searching and Retrieving Scientific Data, International Semantic Web Conference (ISWC)*, 2003.

[63] J. Ledlie, J. Taylor, L. Serban, and M. Seltzer. Self-organization in peer-to-peer systems. In *Proc. of the ACM SIGOPS European Workshop*, 2002.

[64] J. Li, J. Stribling, T.M. Gil, R. Morris, and F. Kaashoek. Comparing the performance of distributed hash tables under churn. In *Proc. of the 2nd International Workshop on Peer-to-Peer Systems (IPTPS)*, 2004.

[65] D. Liben-Nowell, H. Balakrishnan, and D. Karger. Observations on the dynamic evolution of peer-to-peer networks. In *Proc. of the 1st International Workshop on Peer-to-Peer Systems (IPTPS)*, 2002.

[66] P. Linga, I. Gupta, and K. Birman. A churn-resistant peer-to-peer web caching system. In *Proc. of the 2nd International Workshop on Peer-to-Peer Systems (IPTPS)*, 2004.

[67] S. M. Lui and S. H. Kwok. Interoperability of peer-to-peer file sharing protocols. *ACM SIGecom Exchanges*, 3(3):25–33, 2002.

[68] Q. Lv, P. Cao, E. Cohen, K. Li, and S. Shenker. Search and replication in unstructured peer-to-peer networks. In *Proc. of ACM International Conference on Supercomputing (ICS)*, 2001.

[69] Q. Lv, S. Ratnasamy, and S. Shenker. Can heterogeneity make gnutella scalable? In *Proc. of the 1st International Workshop on Peer-to-Peer Systems (IPTPS)*, 2002.

[70] N. A. Lynch. *Distributed algorithms*. Morgan Kaufmann Publisher, 1996.

[71] R. Mahajan, M. Castro, and A. Rowstron. Controlling the cost of reliability in peer-to-peer overlays. In *Proc. of the 2nd International Workshop on Peer-to-Peer Systems (IPTPS)*, 2003.

[72] D. Malkhi, M. Naor, and D. Ratajczak. Viceroy: A Scalable and Dynamic Emulation of the Butterfly. In *Proc. of Principles of Distributed Computing (PODC)*, 2002.

[73] G. S. Manku, M. Bawa, and P. Raghavan. Symphony: Distributed Hashing in a Small Wold. In *Proc. of the 4th USENIX Symposium on Internet Technologies and Systems (USITS)*, 2003.

[74] E. P. Markatos. Tracing a large-scale peer-to-peer system: an hour in the life of gnutella. In *2nd IEEE/ACM International Symposium on Cluster Computing and the Grid*, 2002.

[75] P. Maymounkov and D. Mazires. Kademlia: A Peer-to-peer Information Systems Based on the XOR Metric. *Proc. of the 1st International Workshop on Peer-to-Peer Systems (IPTPS)*, 2002.

[76] R. C. Merkle. secure communications over insecure channels. In *Communications of the ACM*, pages 294–299, 1978.

[77] D. Milojicic, V. Kalogerai, R. Lukose, K. Nagaraja, J. Pruyne, B. Richard, S. Rollins, and Z. Xu. Peer-to-peer computing. Technical report, HP Laboratories Palo Alto, 2002.

[78] D. Molnar, R. Dingledine, and M. Freedman. *Chapter 12: Peer-to-Peer: Harnessing the Power of Disruptive Technologies*. O'Reilly & Associates, 2001.

[79] Morpheus home page. http://www.musiccity.com.

[80] M. Naor and U. Mieder. Novel Architectures for P2P Applications: the Continuous-Discrete Approach. In *Proc. of ACM SPAA*, 2003.

[81] Napster. http://computer.howstuffworks.com/napster2.htm.

[82] W. Nejdl, W. Siberski, M. Wolpers, , and C. Schmnitz. Routing and clustering in schema-based super peer networks. In *Proc. of the 2nd International Workshop on Peer-to-Peer Systems (IPTPS)*, 2003.

[83] W. Nejdl, M. Wolpers, W. Siberski, A. L*ö*ser, I. Bruckhorst, M. Schlosser, and C. Schmitz. Super-peer-based routing and clustering strategies for rdf-based peer-to-peer networks. In *Proc. of the 12th International World Wide Web Conference*, 2003.

[84] Groove Networks. Groove networks product backgrounder, groove networks white paper, 2001. www.groove.net/pdf/groove_product_backgrounder.pdf.

[85] Peer-to-peer working goup, 2001. www.p2pwg.org.

[86] G. Pandurangan, P. Raghavan, and E. Upfal. Building Low-Diameter P2P Networks. In *IEEE Symposium on Foundations in Comp. Sci.*, 2001.

[87] G. Pandurangan, P. Raghavan, and E. Upfal. Building low-diameter peer-to-peer networks. In *IEEE FOCS*, 2001.

[88] C. Plaxton, R. Rajaraman, and A. Richa. Accessing nearby copies of replicated objects in a distributed environment. In *Proc. of ACM SPAA*, 1997.

[89] M. Portmann and A. Seneviratne. The cost of application-level broadcast in a fully decentralized peer-to-peer network. In *ISCC02*, 2002.

[90] F. P. Preparata and J. Vuillemin. The cube-connected cycles: A versatile network for parallel computation. *CACM*, 24(5):300–309, 1981.

[91] M. K. Ramanathan, V. Kalogeraki, and J. Pruyne. Finding good peers in peer-to-peer networks. In *Proc. of International Parallel and Distributed Processing Symposium (IPDPS)*, 2002.

[92] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A scalable content-addressable network. In *Proc. of ACM SIGCOMM*, pages 329–350, 2001.

[93] S. Ratnasamy, M. Handley, R. Karp, and S. Shenker. Topologically-aware overlay construction and server selection. In *Proc. of IEEE Conference on Computer Communications (INFOCOM)*, 2002.

[94] S. Rhea, D. Geels, T. Roscoe, and J. Kubiatowicz. Handling churn in a dht. technical report ucb/csd-3-1299. Technical report, UC Berkeley, Computer Science Division, 2003.

[95] M. Ripeanu and I. Foster. Mapping the gnutella network. *IEEE Internet Computing special issue on Peer-to-Peer Networking*, 6(1):50–57, 2002.

[96] A. Rowstron and P. Druschel. Pastry: Scalable, decentralized object location and routing for large-scale peer-to-peer systems. In *Proc. of the 18th IFIP/ACM International Conference*

on Distributed Systems Platforms (Middleware), 2001.

[97] A. Rowstron and P. Druschel. Storage management and caching in past, a large-scale persistent peer-to-peer storage utility. In *Proc. of the 18th ACM Symp. on Operating Systems Principles (SOSP-18)*, 2001.

[98] J. Saia, A. Fiat, S. Gribble, A.R. Karlin, and S. Saroiu. Dynamically Fault-Tolerant Content Addressable Networks. In *Proc. of the 1st International Workshop on Peer-to-Peer Systems (IPTPS)*, 2002.

[99] M. Satyanarayanan, J. Kistler, P. Kumar, M. Okasadi, E. Siegel, and D. Steere. Coda: A highly available file system for a distributed workstation environment. In *IEEE Transactions on Computers*, pages 447–459, 1990.

[100] M. Schlosser, M. Sintek, S. Decker, and W. Nejdl. Hypercup - Hypercubes, Ontologies and Efficient Search on P2P Networks. In *Proc. of the Workshop on Agents and P2P Computing*, 2002.

[101] H. Shen, C. Xu, and G. Chen. Cycloid: A constant-degree and lookup-efficient p2p overlay network. In *Proc. of International Parallel and Distributed Processing Symposium (IPDPS)*, 2004.

[102] W. Shi and V. Karamcheti. CONCA: An architecture for consistent nomadic content access. In *Proc. of the Workshop on Cache, Coherence, and Consistency (WC3)*, 2001.

[103] C. Shirky. What is p2p...and what isn't? *The O'Reilly Netowrk*, 2000.

[104] E. Sit and R. Morris. Security considerations for peer-to-peer distributed hash tables. In *Proc. of the 1st International Workshop on Peer-to-Peer Systems (IPTPS)*, 2002.

[105] SoftWax. http://www.softwax.com/.

[106] K. Sripanidkulchai. The popularity of gnutella queries and its implications on scalability. In *The O'Reilly Peer-to-Peer and Web Services Conference*, 2001.

[107] I. Stoica, R. Morris, D. Liben-Nowell, Kaashoek M. F. Karger, D. R. Karger, F. Dabek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup protocol for Internet applications. In *IEEE/ACM Trans. on Networking*, 2002.

[108] G. P. Thanasis and D. S. George. Effective use of reputation in peer-to-peer environments. In *Proc. of the International Workshop on Global and P2P Computing (GP2PC)*, 2004.

[109] D. Tran, K. Hua, and T. Do. Zigzag: An efficient peer-to-peer scheme for media streaming. In *Proc. of IEEE Conference on Computer Communications (INFOCOM)*, 2003.

[110] A. Veytsel. There is no p-to-p market... but there is a market for p-to-p. In *Aberdeen Group Presentation at the P2PWG*, 2001.

[111] M. Waldman, A. Rubin, and L. Cranor. Publius: A robust, tamper-evident, censorship-resistant web publishing system. In *Proc. of the USENIX Security Symposium*, 2000.

[112] M. Waldvogel and R. Rinaldi. Efficient topology-aware overlay network. In *Proc. of the 1st ACM Workshop on Hot Topics in Networks (HotNets-I)*, 2002.

[113] Y. Wang and J. Vassileva. Trust and reputation model in peer-to-peer networks. In *Proc. of the 3rd International Conference on Peer-to-Peer Computing (P2P)*, September 2003.

[114] B. Wiley. *Chapter 19: Peer-to-Peer: Harnessing the Power of Disruptive Technologies*. O'Reilly & Associates, 2001.

[115] J. Xu, A. Kumar, and X. Yu. On the Fundamental Tradeoffs between Routing Table Size and Network Diameter in Peer-to-Peer Networks. In *To Appear in IEEE JSAC*, 2003.

[116] Z. Xu, C. Tang, S. Banerjee, and S. Lee. Rita: Receiver initiated just-in-time tree adaptation for rich media distribution. In *The 13th International Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV)*, 2003.

[117] Z. Xu, C. Tang, and Z. Zhang. Building topology-aware overlays using global soft-state. In *Proc. of the 23rd International Conference on Distributed Computing Systems (ICDCS)*, 2003.

[118] Z. Xu and Z. Zhang. Building Low-maintenance Expressways for P2P systems. In *Hewlett-Packard Labs: Palo Alto*, 2001.

[119] B. Yang, , P. Vinograd, and H. Garcia-Molina. Evaluating GUESS and Non-Forwarding Peer-to-Peer Search. In *Proc. of the 24rd International Conference on Distributed Computing Systems (ICDCS)*, 2004.

[120] B. Yang and H. Garcia-Molina. Comparing hybrid peer-to-peer systems. In *Proc. of the 27th Intl. Conf. on Very Large Databases*, 2001.

[121] B. Yang and H. Garcia-Molina. Improving search in peer-to-peer networks. In *Proc. of the 22nd International Conference on Distributed Computing Systems (ICDCS)*, pages 5–14, 2002.

[122] B. Yang and H. Garcia-Molina. Designing a super-peer network. In *Proceedings ICDE*, 2003.

[123] J. Yin, J. Martin, A. Venkataramani, L. Alvisi, and M. Dahlin. Separating agreement from execution for Byzantine fault tolerant services. In *Proc. of the 19th ACM Symp. on Operating Systems Principles (SOSP-19)*, 2003.

[124] B. Zhao, J. Kubiatowicz, and A. Joseph. Tapestry: An Infrastructure for Fault-tolerant wide-area location and routing. Technical Report UCB/CSD-01-1141, Computer Science Division, UC Berkeley, 2001.

[125] S. Zhuang, B. Zhao, A. Joseph, R. Kotz, and J. Kubiatowicz. Bayeux: An architecture for scalable and fault-tolerant wide-area data dissemination. In *The 11th International Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV)*, 2001.