Open Problems in Data-Sharing Peer-to-Peer Systems

Neil Daswani, Hector Garcia-Molina, and Beverly Yang

Stanford University, Stanford CA 94305, USA,
{daswani, hector, byang}@db.stanford.edu,
http://www-db.stanford.edu

Abstract. In a Peer-To-Peer (P2P) system, autonomous computers pool their resources (e.g., files, storage, compute cycles) in order to inexpensively handle tasks that would normally require large costly servers. The scale of these systems, their "open nature", and the lack of centralized control pose difficult performance and security challenges. Much research has recently focused on tackling some of these challenges; in this paper, we propose future directions for research in P2P systems, and highlight problems that have not yet been studied in great depth. We focus on two particular aspects of P2P systems – *search* and *security* – and suggest several open and important research problems for the community to address.

1 Introduction

Peer-to-peer (P2P) systems have recently become a very active research area, due to the popularity and widespread use of P2P systems today, and their potential uses in future applications. Recently, P2P systems have emerged as a popular way to share huge amounts of data (e.g., [1, 16, 17]). In the future, the advent of large-scale ubiquitous computing makes P2P a natural model for interaction between devices (e.g., via the web services [18] framework).

P2P systems are popular because of the many benefits they offer: adaptation, self-organization, load-balancing, fault-tolerance, availability through massive replication, and the ability to pool together and harness large amounts of resources. For example, file-sharing P2P systems distribute the main cost of sharing data – bandwidth and storage – across all the peers in the network, thereby allowing them to scale without the need for powerful, expensive servers.

Despite their many strengths, however, P2P systems also present several challenges that are currently obstacles to their widespread acceptance and usage – e.g., security, efficiency, and performance guarantees like atomicity and transactional semantics. The P2P environment is particularly challenging to work in because of the scale of the network and unreliable nature of peers characterizing most P2P systems today. Many techniques previously developed for distributed systems of tens or hundreds of servers may no longer apply; new techniques are needed to meet these challenges in P2P systems.

In this paper, we consider research problems associated with *search* and *security* in *data-sharing* P2P systems. Though data-sharing P2P systems are capable of sharing enormous amounts of data (e.g., 0.36 petabytes on the Morpheus [17] network as of October 2001), such a collection is useless without a search mechanism allowing users to quickly locate a desired piece of data (Section 2). Furthermore, to ensure proper, continued operation of the system, security measures must be in place to protect against availability attacks, unauthentic data, and illegal access (Section 3). In this paper, we highlight several important and open research issues within both of these topics.

Note that this paper is not meant to be an exhaustive survey of P2P research. First, P2P can be applied to many domains outside of data-sharing; for example, computation (e.g., [19, 20]), collaboration (e.g., [21]), and infrastructure systems (e.g., [22]) are all popular applications of P2P. Each application faces its own unique challenge (e.g., job scheduling in computation systems), as well as common issues (e.g., resource discovery). In addition, within data-sharing systems there exists important research outside of search and security. Good examples include resource management issues such as fairness and administrative ease. Finally, due to space limitations, the issues we present within search and security are not comprehensive, but illustrative. Examples are also chosen with a bias towards work done at the Stanford Peers group [23], because it is the research that the authors know best.

2 Search

A good *search mechanism* allows users to effectively locate desired data in a resource-efficient manner. Designing such a mechanism is difficult in P2P systems for several reasons: scale of the system, unreliability of individual peers, etc. In this section, we outline the basic architecture, requirements and goals of a search mechanism for P2P systems, and then suggest several areas of open research.

2.1 Overview

In a data-sharing P2P system, users submit queries and receive results (such as data, or pointers to data) in return, via the search mechanism. Data shared in the system can be of any type. In most cases users share files, such as music files, images, news articles, web pages, etc. Other possibilities include data stored in a relational DBMS, or a queryable spreadsheet. Queries may take any form that is appropriate given the type of data shared. For example, in a file-sharing system, queries might be keywords with regular expressions, and the search may be defined over different portions of the document (e.g., header, title, metadata).

A search mechanism defines the behavior of peers in three areas:

- **Topology**: Defines how peers are connected to each other. In some systems (e.g., Gnutella [1]), peers may connect to whomever they wish. In other systems, peers are organized into a rigid structure, in which the number and nature of connections is dictated by the protocol. Defining a rigid topology may increase efficiency, but will restrict autonomy.

- **Data placement**: Defines how data or metadata is distributed across the network of peers. For example, in Gnutella, each node stores only its own collection of data. In Chord [2], data or metadata is carefully placed across nodes in a deterministic fashion. In super-peer networks [12], metadata for a small group of peers is centralized onto a single super-peer.
- **Message routing**: Defines how messages are propagated through the network. When a peer submits a query, the query message is sent to a number of the peer's "neighbors" (that is, nodes to whom the peer is connected), who may in turn forward the message sequentially or in parallel to some of their neighbors, and so on. When, and to whom, messages are sent is dictated by the routing protocol. Often, the routing protocol can take advantage of known patterns in topology and data placement, in order to reduce the number of messages sent.

In an actual system, the general model described above takes on a different form depending on the *requirements* of the system. Requirements are specified in several main categories:

- **Expressiveness**: The query language used for a system must be able to describe the desired data in sufficient detail. Key lookups are not expressive enough for IR searches over text documents, and keyword queries are not expressive enough to search structured data such as relational tables.
- **Comprehensiveness**: In some systems, returning any single result is sufficient (e.g., anycast), whereas in others, *all* results are required. The latter type of system requires a comprehensive search mechanism, in which all possible results are returned.
- **Autonomy**: Every search mechanism must define peer behavior with respect to topology, data placement, and message routing. However, autonomy of a peer is restricted when the mechanism limits behavior that a peer could reasonably expect to control. For example, a peer may wish to only connect to its friends or other trusted peers in the same organization, or the peer may wish to control which nodes can store its data (e.g., only nodes on the intranet), and how much of other nodes' data it must store. Depending on the purpose and users of the system, the search mechanism may be required to meet a certain level of autonomy for peers.

In this paper, we assume the additional requirement that the search mechanism be decentralized. A P2P system may have centralized search, and indeed, such "hybrid systems" have been very useful and popular in practice (e.g., [16]). However, centralized systems have been well-studied, and it is desirable that the search mechanism share the same benefits of P2P mentioned in Section 1; hence, here we focus only on decentralized P2P solutions.

While a well-designed search mechanism must satisfy the requirements specified by the system, it should also seek to maximize the following goals:

- **Efficiency**: We measure efficiency in terms of absolute resources consumed – bandwidth, processing power, storage, etc. An efficient use of resources results in lighter overhead on the system, and hence, higher throughput.
- **Quality of Service**: We can measure quality of service (QoS) along many different metrics depending on the application – number of results, response time, etc. Note the distinction between QoS and efficiency: QoS focuses on user-perceived qualities, while efficiency focuses on the resource cost (e.g., bandwidth) to achieve a level of service.
- **Robustness**: We define robustness to mean stability in the presence of failures: quality of service and efficiency are maintained as peers in the system fail or leave. Robustness to attacks is a separate issue discussed in Section 3.

By placing current work in the framework of requirements and goals above, we can identify several areas in which research is much needed. In the following section, we mention just a few of these areas.

2.2 Expressiveness

In order for P2P systems to be useful in a wide range of applications, they must be able to support query languages of varying levels of expressiveness. Thus far, work in P2P search has focused on answering simple queries, such as key lookups. An important area of research therefore lies in developing mechanisms for richer query languages. Here, we list a few examples of useful types of queries, and discuss the related work and challenges in supporting them.

- **Key lookup**: The simplest form of query is an object lookup by key or identifier. Protocols directly supporting this primitive have been widely studied, and efficient solutions exist (e.g., [2–4]). Ongoing research is exploring how to make these protocols more efficient and robust [5].
- **Keyword**: While much research has focused on search techniques for keyword queries (e.g., [11, 10, 6]), all of these techniques have been geared towards efficient, *partial* (not comprehensive) search – e.g., all music-sharing systems currently only support partial search. Partial search is acceptable in those applications where a few keywords can usually uniquely identify the desired file (e.g., music-sharing systems, as opposed to web page repositories), because the first few matches are likely to satisfy the user's request. Techniques for partial search can always be made comprehensive simply by sending the query message to every peer in the network; however, such an approach is prohibitively expensive. Hence, designing techniques for efficient, comprehensive search remains an open problem.
- **Ranked keyword**: If many results are returned for comprehensive keyword search, users will need results to be ranked and filtered by relevance. While the query language for ranked keyword search remains the same, the additional information in the results (i.e., the relevance ranking) poses additional challenges and opportunities. For example, ranked search can be built on top of regular search by retrieving all results and sorting locally; however, state-of-the-art ranking functions usually require global statistics over the total

collection of documents (e.g., document frequency). Collecting and maintaining these statistics in a robust, efficient, and distributed manner is a challenge. At the same time, ranked results allow the system to return "top k" results, which provides the opportunity to optimize search if k is much less than the total number of results (which is generally the case, for example, in web searches). Techniques for ranked search exists for distributed systems of moderate scale (e.g., [7]), but future research must extend these techniques to support much larger systems.
- **Aggregates**: A user may sometimes be interested in knowing *aggregate* properties of the system or data collection as a whole, rather than locating specific data. For example, to collect global statistics to support ranked keyword search mentioned earlier, a user could submit several `SUM` queries to sum the number of documents that contain a particular term. Ongoing research [8] addresses `COUNT` queries defined over a predicate – for example, counting the number of nodes that belong to the `stanford.edu` domain. Further research is needed to extend these techniques into more expressive aggregates like `SUM`, `MAX`, and `MEDIAN`.
- **SQL**: As a complex language defined over a rich data model, SQL is the most difficult query language to support among the examples listed. Current research on supporting SQL in P2P systems is very preliminary. For example, the PIER project [9] supports a subset of SQL over a P2P framework, but they report significant performance "hotspots" in their preliminary implementation. A great deal of additional research is needed to advance current work into a search mechanism with reasonable performance, and to investigate alternative approaches to the problem.

2.3 Autonomy, Efficiency and Robustness

Autonomy, efficiency and robustness are all desirable features in any system. These features conceptually define an informal space of P2P systems, as shown in Figure 1a, where a point in the space represents a system with the corresponding "values" for each feature. Note that the value of a system with respect to a feature only provides a partial order, since features can be measured along several metrics (e.g., efficiency can be measured by bandwidth, processing power, and storage). Hence, Figure 1 illustrates the space by showing just a few points for which the relative order (and not the actual coordinates) along each feature is fairly obvious.

The space defined by autonomy, efficiency and robustness is not fully explored; in particular, there appears to be some correlation between autonomy and efficiency (Figure 1b), and autonomy and robustness (Figure 1c). A partial explanation for the first correlation is that less autonomy allows the search mechanism to specify a data placement and topology such that:

- There exist a deterministic way to locate data within bounded cost (e.g., Chord)
- There is a small set of nodes that is guaranteed to hold the answer, if it exists (e.g., super-peer networks, concept clusters [13])

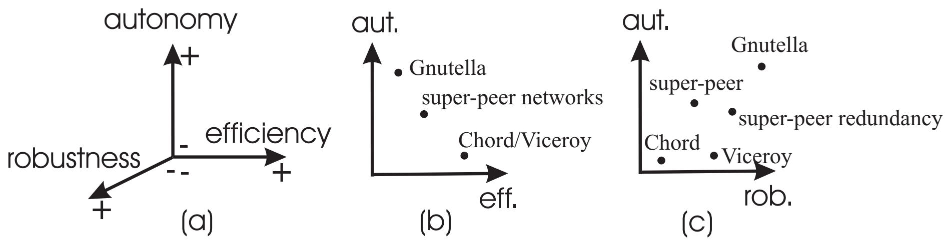

Fig. 1. The space of systems defined by autonomy, efficiency and robustness (*a*). Looking at a few example systems within this space, there appears to be a relationship between autonomy and efficiency *(b)*, and autonomy and robustness (*c*)

– There is an increased chance of finding results on a random node (e.g., replication [6]).

At the same time, these rigidly organized networks can be difficult or expensive to maintain, especially as peers join and leave the network at the rapid rate characteristic of many P2P systems. As a result, robustness is also correlated with autonomy.

One important area of research is finding techniques that push beyond the current tradeoffs between efficiency, autonomy and robustness. Decoupling efficiency from autonomy seems to be the greatest challenge, since existing techniques almost uniformly sacrifice autonomy to achieve efficiency. However, the potential gain is the greatest: a search mechanism that is efficient, robust, *and* preserves peer autonomy. Decoupling autonomy from robustness is also important, because it allows greater flexibility in choosing the desired properties of the mechanism. For example, a search mechanism that is robust, but has low peer autonomy, can be desirable if the lack of autonomy leads to efficiency, and peer autonomy is not a requirement of the system.

Several research projects have tackled the autonomy/robustness tradeoff. For example, the Viceroy [14] network construction maintains a low level of peer autonomy, but increases robustness and efficiency by reducing the cost of maintaining the network structure to a constant term, for each join/leave of a peer. In comparison, most distributed hash tables (DHTs) with the same functionality have logarithmic maintenance cost. As another example, super-peer redundancy [12] imposes slightly stricter rules on topology and data placement within a cluster of peers, but this decrease in autonomy results in greater robustness of the super-peer and improved efficiency in the overall network.

Another interesting area of research is providing fine-granularity tuning of the tradeoff between autonomy and efficiency within a single system. A single user may have varying needs; for example, a company may have a few sensitive files that must remain on the intranet, but the remaining files can be stored anywhere. A single system that can be tuned to support all of these needs is more desirable than requiring users to use different systems for different purposes. A good example of a tunable system is SkipNet [15]. SkipNet allows users to specify a range of peers on which a document may be stored (e.g., all peers within the `stanford.edu` domain). At one extreme, if the range is always limited to a single peer, then user autonomy is high, but the system ceases to be P2P and loses good properties such as load-balancing and self-organization. At the other extreme, if the range always includes all peers in the network, SkipNet functions as a traditional P2P lookup system with low autonomy, but other good properties. While SkipNet does not push beyond existing tradeoffs, its value lies in allowing users to choose the point along the tradeoff that meets their needs.

2.4 Quality of Service

In the previous discussion, we implicitly assume a fixed level of service (e.g., number of results per query) that must be maintained as other factors (e.g., autonomy) are varied. However, quality of service (QoS) can be measured with many different metrics, depending on the application, and a spectrum of acceptable performance exists along each metric. Examples of service metrics include number of results (e.g., in partial-search systems), response time, and relevance (e.g., precision and recall in ranked keyword searches). A constant challenge in designing P2P systems is achieving a desired level of QoS as efficiently as possible. Because metrics and applications differ so widely, this challenge must often be tackled on a per-case basis.

As an example, the number of results returned is an important QoS metric for partial-search systems like Gnutella. However, in systems where there is high autonomy (such as Gnutella), there is a clear and unavoidable tradeoff between number of results and cost; hence, the interesting problem is to get as close as possible to the lower bounds of the tradeoff. For example, the directed BFS technique in [11] attempts to minimize cost by sending messages to "productive" nodes (e.g., nodes with large collections). Concept-clustering networks (e.g., [13]) cluster peers together according to "interest" (e.g., music genre), and send queries to the cluster that best matches the queries' area of interest. These techniques do improve the tradeoff between cost and number of results, but are clearly not optimal: performance of directed BFS depends on the ad-hoc topology and is therefore unpredictable, while concept-clustering only works well if queries and interests fall cleanly into single categories. Can there exist a general technique that can guarantee (with high probability) that the cost/QoS tradeoff is optimal?

With other metrics of QoS, there is not such an obvious tradeoff between quality and cost. In these cases, the goal is to maintain the same level of service while decreasing cost. For example, consider the "satisfaction" metric, which is binary and is true when a threshold number of results is found. Satisfaction is an important metric in partial-search systems where only the first k results are displayed to the user (e.g., [16, 1]). Reference [11] shows that, compared to current techniques, cost can be drastically reduced while maintaining satisfaction. Furthermore, even better performance is probably possible if we discard this work's requirement of peer autonomy and simplicity. Additional research is required to explore this space further.

3 Security

Securing P2P data sharing applications is challenging due to their open and autonomous nature. Compared to a client-server system in which servers can be relied upon or trusted to always follow protocols, peers in a P2P system may provide no such guarantee. The environment in which a peer must function is a hostile one in which any peer is welcome to join the network; these peers cannot necessarily be trusted to route queries or responses correctly, store documents when asked to, or serve documents when requested. In this part of the paper, we outline a number of security issues that are characteristic to P2P data sharing systems, discuss a few examples of research that has taken place to address some of these issues, and suggest a number of open research problems.

We organize the security requirements of P2P data sharing systems into four general areas: availability, file authenticity, anonymity, and access control. Today's P2P systems rarely address all of the necessary requirements in any one of these areas, and developing systems that have the flexibility to support requirements in all of these areas is expected to be a research challenge for quite some time.

For each of these areas, it will be important to develop techniques that *prevent*, *detect*, *manage*, and are able to *recover* from attacks. For example, since it may be difficult to prevent a denial-of-service attack against a system's availability, it will be important to develop techniques that are able to 1) detect when a denial-of service attack is taking place (as opposed to there just being a high load), 2) manage an attack that is "in-progress" such that the system can continue to provide some (possibly reduced) level of service to clients, and 3) recover from the attack by disconnecting the malicious nodes.

3.1 Availability

There are a number of different node and resource availability requirements that are important to P2P file sharing systems. In particular, each node in a P2P system should be able to accept messages from other nodes, and communicate with them to offer access to the resources that it contributes to the network.

A denial-of-service (DoS) attack attempts to make a node and its resources unavailable by overloading it. The most obvious DoS attack is targeted at using up all of a node's bandwidth. This type of attack is similar to traditional network-layer DoS attacks (e.g. [31]). If a node's available bandwidth is used up transferring useless messages that are directly or indirectly created by a malicious node, all of the other resources that the node has to offer (including CPU and storage) will also be unavailable to the P2P network.

A specific example of a DoS attack against node availability is a chosen-victim attack in Gnutella that an adversary constructs as follows: a malicious super-node maneuvers its way into a "central" position in the network and then responds to every query that passes thru it claiming that the victim node has a file that satisfies the query (even though it does not). Every node that receives one of these responses then attempts to connect to the victim to obtain the file that they were looking for, and the number of these requests overloads the bandwidth of the victim such that any other node seeking a file that the victim does have is unable to communicate the victim.

The key aspect to note here is that in our example the attacker exploited a vulnerability of the Gnutella P2P protocol (namely, that any node can respond to any query claiming that any file could be anywhere). In the future, P2P protocols need to be designed to make it hard for adversaries to construct DoS attacks by taking advantage of loosely constrained protocol features.

Attackers that construct DoS attacks typically need to find and take advantage of an "amplification mechanism" in the network to cause significantly more damage than they could with only their own resources. In addition, if they would like to have control over how their attack is carried out, they must also find or create a back-door communication channel to communicate with "zombie" hosts that they infiltrate using manual or automatic means. It is important to design future P2P protocols such that they do not open up new opportunities for attackers to use as amplifiers and back-door communication channels.

Some research has taken place to date to specifically address DoS attacks in P2P networks. In particular, [38] addresses DoS attacks based on query-floods in the Gnutella network. However, more research is necessary to understand the effects of other types of DoS attacks in various P2P networks.

Aside from DoS attacks, node availability can also be attacked by malicious users that infiltrate victim nodes and induce their failure. These types of attacks can be modeled as fail-stop or byzantine failures, which could potentially be dealt with using many techniques that have already been developed (e.g. [34]). However, these techniques have typically not been popular due to their inefficiency, unusually high message overhead, and complexity. In addition, these techniques often assume complete and secure pairwise connectivity between nodes, which is not the case in most P2P networks. Further research will be necessary to make these or similar techniques acceptable from a performance and security standpoint in a P2P context.

In addition, there are many proposals to provide significant levels of fault-tolerance in the face of node failure including CAN [3], Chord [2], Pastry [4], and Viceroy [14]. Security analyses of these types of proposals cound be found in [43] and [36]. The IRIS [25] project seeks to continue the investigation of these types of approaches.

A malicious node can also directly attack the availability of any of the particular resources at a node. The CPU availability at a node can be attacked by sending a modest number of complex queries to bog down the CPU of a node without consuming all of its bandwidth. The available storage could be attacked by malicious nodes who are allowed to submit bogus documents for storage. One approach to deal with this is to allocate storage to nodes in a manner proportional to the resources that a node contributes to the network as proposed in [28].

We might like to ensure that all files stored in the system are always available regardless of which nodes in the network are currently online. File availability ensures that files can be perpetually preserved, regardless of factors such as the popularity of the files. Systems such as Gnutella and Freenet provide no guarantees about the preservation of files, and unpopular files tend to disappear.

Even if files can be assured to physically exist and are accessible, a DoS attack can still be made against the quality-of-service with which they are available. In this type of a DoS attack, a malicious node makes a file available, but when a request to download the file is received, it serves the file so slowly that the requester will most likely lose patience and cancel the download before it completes. The malicious node could also claim that it is serving the file requested but send some other file instead. As such, techniques such as hash trees [26] could to be used by the client to incrementally ensure that the server is sending the correct data, and that data is sent at a reasonable rate.

3.2 File Authenticity

File authenticity is a second key security requirement that remains largely unaddressed in P2P systems. The question that a file authenticity mechanism answers is: given a query and a set of documents that are responses to the query, which of the responses are "authentic" responses to the query? For example, if a peer issues a search for "Origin of Species" and receives three responses to the query, which of these responses are "authentic"? One of the responses may be the exact contents of the book authored by Charles Darwin. Another response may be the content of the book by Charles Darwin with several key passages altered. A third response might be a different document that advocates creationism as the theory by which species originated.

Note that the problem of file authenticity is different than the problem of file (or data) integrity. The goal of file integrity is to ensure that documents do not get inadvertently corrupted due to communication failures. Solutions to the file integrity problem usually involve adding some type of redundancy to messages in the form of a "signature." After a file is sent from node A to node B, a signature of the file is also sent. There are many fewer bits in the signature than in the file itself, and every bit of the signature is dependent on every bit of the file. If the file arrived at node B corrupted, the signature would not match. Techniques such as CRCs (cyclic redundancy checks), hashing, MACs (message authentication codes), or digital signatures (using symmetric or asymmetric encryption) are well-understood solutions to the file integrity problem.

The problem of file authenticity, however, can be viewed as: given a query, what is (or are) the "authentic" signature(s) for the document(s) that satisfy the query? Once some file authenticity algorithm is used to determine what is (or are) the authentic signatures, a peer can inspect responses to the query by checking that each response has an authentic signature.

In our discussion until this point, we have not defined what it means for a file to be authentic. There are a number of potential options: we will outline four reasonable ones.

Oldest Document. The first definition of authenticity considers the oldest document that was submitted with a particular set of metadata to be the authentic copy of that document. For example, if Charles Darwin was the first author to ever submit a document with the title "Origin of Species," then his document would be considered to be an authentic match for a query looking for "Origin of Species" as the title. Any documents that were submitted with the title "Origin of Species" after Charles Darwin's submission would be considered unauthentic matches to the query even if we decided to store these documents in the system. Timestamping systems (e.g. [35]) can be helpful in constructing file authenticity systems based on this approach.

Expert-Based. In this approach, a document would be deemed authentic by an "expert" or authoritative node. For example, node G may be an expert that keeps track of signatures for all files ever authored by any user of G. If a user searching for documents authored by any of G's users is ever concerned about the potential authenticity of a file received as a response to a query, node G can be consulted. Of course, if node G is unavailable at any particular time due to a transient or permanent failure, is infiltrated by an attacker, or is itself malicious, it may be difficult to properly verify the authenticity of files that G's users authored. Offline digital signature schemes (i.e., RSA) can be used to verify file authenticity in the face of node failures, but are limited by the lifetime and security of public/private keys.

Voting-Based. To deal with the possible failure of G or a compromised key in our last approach, our third definition of authenticity takes into account the "votes" of many experts. The expert nodes may be nodes that are run by human experts qualified to study and assess the authenticity of particular types of files, and the majority opinion of the human experts can be used to assess the authenticity of a file. Alternatively, the expert nodes may simply be "regular" nodes that store files, and will vote that a particular file is authentic if they store a copy of it. In this scheme, users are expected to delete files that they do not believe are authentic, and a file's authenticity is determined by the number of copies of the file that are distributed throughout the system. The key technical issues with this approach are how to prevent spoofing of votes, of nodes, and of files.

Reputation-Based. Some experts might be more trustworthy than others (as determined by past performance), and we might weight the votes of more trustworthy experts more heavily. The weights in this approach are a simple example of "reputations" that may be maintained by a reputation system. A reputation system is responsible for maintaining, updating, and propagating such weights and other reputation information [41]. Reputation systems may or may not choose to use voting in making their assessments. There has been some study of reputation systems in the context of P2P networks, but no such system has been commercially successful (e.g. [33, 24]).

3.3 Anonymity

There is much work that has taken place on anonymity in the context of the Internet both at the network-layer (e.g. [30]) as well as at the application-layer

Type 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?

Table 1. Types of Anonymity

(e.g. [40]). In this section we specifically focus on application-layer anonymity in P2P data sharing systems.

While some would suggest that many users are interested in anonymity because it allows them to illegally trade copyrighted data files in an untraceable fashion, there are many legitimate reasons for supporting anonymity in a P2P system. Anonymity can enable censorship resistance, freedom of speech without the fear of persecution, and privacy protection. Malicious parties can be prevented from deterring the creation, publication, and distribution of documents. For example, such a system may allow an Iraqi nuclear scientist to publish a document about the true state of Iraq's nuclear weapons program to the world without the fear that Saddam Hussein's regime could trace the document back to him or her. Users that access documents could also have their privacy protected in such a system. An FBI agent could access a company's public information resources (i.e., web pages, databases, etc.) anonymously so as not to arouse suspicion that the company may be under investigation.

There are a number of different types of anonymity that can be provided in a P2P system. It is difficult for the adversary to determine the answers to different questions for different types of anonymity. Table 1 summarizes a few types of anonymity discussed in [39].

We would ideally like to provide anonymity while maintaining other desirable search and security features such as efficiency, decentralization, and peer discovery. Unfortunately, providing various types of anonymity often conflicts with these design goals for a P2P system.

To illustrate one of these conflicting goals, consider the natural trade-off between server anonymity and efficient search. If we are to provide server anonymity, it should be impossible to determine which nodes are responsible for storing a document. On the other hand, if we would like to be able to efficiently search for a document, we should be able to tell exactly which nodes are responsible for storing a document. A P2P system such as Free Haven that strives to provide server anonymity resorts to broadcast search, while others such as Freenet [27] provide for efficient search but do not provide for server anonymity. Freenet does, however, provide author anonymity. Nevertheless, supporting server anonymity and efficient search concurrently remains an open issue.

There exists a middle-ground: we might be able to provide some level of server anonymity by assigning pseudonyms to each server, albeit at the cost of search efficiency. If an adversary is able to determine the pseudonym for the server of a controversial document, the adversary is still unable to map the pseudonym to the publisher's true identity or location. The document can be accessed in such a way as to preserve the server's anonymity by requiring that a reader (a potential adversary) never directly communicate with a server. Instead, readers only communicate with a server through a chain of intermediate proxy nodes that forward requests from the reader to the server. The reader presents the server's pseudonym to a proxy to request communication with the server (thereby hiding a server's true identity), and never obtains a connection to the actual server for a document (thereby hiding the server's location). Reader anonymity can also be provided using a chain of intermediate proxies, as the server does not know who the actual requester of a document is, and each proxy does not know if the previous node in the chain is the actual reader or is just another proxy. Of course, in both these cases, the anonymity is provided based on the assumption that proxies do not collude. The degradation of anonymity protocols under attacks has been studied in [44], and this study suggests that further work is necessary in this area.

Free Haven and Crowds [40] are examples of systems that use forwarding proxies to provide various types of anonymity with varying strength. Each of these systems differ in how the level of anonymity degrades as more and more potentially colluding malicious nodes take on the responsibilities of proxies. Other techniques that are commonly found in systems that provide anonymity include mix networks (e.g. [32]), and using cryptographic secret-sharing techniques to split files into many shares (e.g. [42]).

3.4 Access Control

Intellectual property management and digital rights management issues can be cast as access control problems. We want to restrict the accessibility of documents to only those users that have paid for that access. P2P systems currently cannot be trusted to successfully enforce copyright laws or carry out any form of such digital rights management, especially since few assumptions can be made about key management infrastructure. This has led to blatant violation of copyright laws by users of P2P systems, and has also led to lawsuits against companies that build P2P systems.

The trade-offs involved in enforcing access control in a P2P data sharing system are challenging because if a system imposes restrictions over what types of data it shares (i.e., only copy-protected content), then its utility will be limited. On the other hand, if it imposes no such restrictions, then it can be used as a platform to freely distribute any content to anyone that wants it [37].

Further effort must go into exploring whether or not it is reasonable to have the P2P network enforce access control, or if the enforcements should take place at the endpoints of the network. In either case, only users that own (or have paid for) the right to download and access certain files should be able to do so to legally support data sharing applications.

If the benefits of P2P systems are to be realized, we need to explore the feasibility of and the technical approaches to instrumenting them with appropriate mechanisms to allow for the management of intellectual property.

4 Conclusion

Many of the open problems in P2P data sharing systems surround search and security issues. The key research problem in providing a search mechanism is how to provide for maximum expressiveness, comprhensiveness, and autonomy with the best possible efficiency, quality-of-service, and robustness. The key to securing a P2P network lies in designing mechanisms that ensure availabiity, file authenticity, anonymity, and access control. In this paper, we have illustrated some of the trade-offs at the heart of search and security problems in P2P data sharing systems, and outlined several major areas of importance for future work.

References

1. Gnutella website. http://www.gnutella.com
2. Stoica, I., Morris, R., Karger, D., Kaashoek, M. F., Balakrishnan, H.: Chord: A scalable peer-to-peer lookup service for internet applications. Proc. ACM SIGCOMM (2001)
3. Ratnasamy, S., Francis, P., Handley, M., Karp, R., Shenker, S.: A scalable content addressable network. Proc. ACM SIGCOMM (2001)
4. Rowstron, A., Druschel, P.: Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. Proc. of the 18th IFIP/ACM Intl. Conf. on Distributed Systems Platforms (2001)
5. Ratnasamy, S., Shenker, S., Stoica, I.: Routing Algorithms for DHTs: Some Open Questions. Proc. IPTPS (2002)
6. Lv, Q., Cao, P., Cohen, E., Li, K., Shenker, S.: Search and replication in unstructured peer-to-peer networks. Proc. of Intl. Conf. on Supercomputing (2002)
7. Cuenca-Acuna, F. M., Peery, C., Martin, R. P., Nguyen, T. D.: PlanetP: using gossiping to build content addressable peer-to-peer information sharing communities. Technical Report DCS-TR-487, Dept. of Computer Science, Rutgers Univ. (2002)
8. Bawa, M., Garcia-Molina, H., Gionis, A., Motwani, R.: Estimating the size of a peer-to-peer network (2002)
9. Harren, M., Hellerstein, M., Huebsch, R., Loo, B., Shenker, S., Stoica, I.: Complex Queries in DHT-based Peer-to-Peer Networks. Proc. IPTPS (2002)
10. Crespo, A., Garcia-Molina, H.: Routing indicies for peer-to-peer systems. Proc. 28th Intl. Conf. on Distributed Computing Systems (2002)
11. Yang, B., Garcia-Molina, H.: Improving search in peer-to-peer systems. Proc. 28th Intl. Conf. on Distributed Computing Systems (2002)
12. Yang, B., Garcia-Molina, H.: Designing a super-peer network. Proc. ICDE (2003)
13. Schlosser, M., Sintek, M., Decker, S., Nejdl, W.: A scalable and ontology-based P2P infrastructure for semantic web services (2002)
14. Malkhi, D., Nao, M., Ratajczak, D.: Viceroy: a scalable and dynamic emulation of the butterfly. Proc. PODC (2002)
15. Harvey, N., Jones, M., Saroiu, S., Theimer, M., Wolman, A.: SkipNet: a scalable overlay network with practical locality properties (2002)

16. Napster website. http://www.napster.com
17. Morpheus website. http://www.musiccity.com
18. W3C website on Web Services. http://www.w3.org/2002/ws
19. Seti@Home website. http://setiathome.ssl.berkely.edu
20. DataSynapse website. http://www.datasynapse.com
21. Groove Networks website. http://www.groove.net
22. Stoica, I., Adkins, D., Zhuang, S., Shenker, S., Surana, S.: Internet Indirection Infrastructure. Proc. SIGCOMM (2002)
23. Stanford Peers group website. http://www-db.stanford.edu/peers
24. Reputation research network home page. http://databases.si.umich.edu/reputations/
25. Iris: Infrastructure for resilient internet systems. http://iris.lcs.mit.edu/
26. Personal communication with Dan Boneh.
27. Clarke, I., Sandberg, O., Wiley, B., Hong, T.W.: Freenet: A distributed anonymous information storage and retrieval system. Workshop on Design Issues in Anonymity and Unobservability, pages 46–66 (2000)
28. Cooper, B., Garcia-Molina., H.: Peer to peer data trading to preserve information. ACM Transactions on Information Systems (2002)
29. Dingledine, R., Freedman, M.J., Molnar, D.: The free haven project: Distributed anonymous storage service. Workshop on Design Issues in Anonymity and Unobservability, pages 67–95 (2000)
30. Freedman, M.J., Morris, R.: Tarzan: A peer-to-peer anonymizing network layer. Proc. 9th ACM Conference on Computer and Communications Security, Washington, D.C. (2002)
31. Garber, L.: Denial-of-service attacks rip the internet. Computer, pages 12-17 (April 2000)
32. Hill, R., Hwang, A., Molnar, D.: Approaches to mixnets.
33. Lethin, R.: Chapter 17: Reputation. Peer-to-Peer: Harnessing the Power of Disruptive Technologies ed. Andy Oram, O'Reilly and Associates (2001)
34. Lynch, N.A.: Distributed algorithms. Morgan Kaufmann (1996)
35. Maniatis, P., Baker, M.: Secure History Preservation Through Timeline Entanglement. Proc. 11th USENIX Security Symposium, SF, CA, USA (2002)
36. Ganesh, A., Rowstron, A., Castro, M., Druschel, P., Wallach, D.: Security for structured peer-to-peer overlay networks. Proc. 5th OSDI, Boston, MA (2002)
37. Peinado, M., Biddle, P., England, P., Willman, B.: The darknet and the future of content distribution. http://crypto.stanford.edu/DRM2002/darknet5.doc.
38. Daswani, N., Garcia-Molina, H.: Query-flood DoS Attacks in Gnutella. Proc. Ninth ACM Conference on Computer and Communications Security, Washington, DC (2002)
39. Molnar, D., Dingledine, R., Freedman, M.: Chapter 12: Free haven. Peer-to-Peer: Harnessing the Power of Disruptive Technologies ed. Andy Oram, O'Reilly and Associates (2001)
40. Reiter, M.K., Rubin, A.D.: Crowds: anonymity for Web transactions. ACM Transactions on Information and System Security, 1(1):66–92 (1998)
41. Resnick, P., Zeckhauser, R., Friedman, E., Kuwabara, K.: Reputation systems. Communications of the ACM, pages 45-48 (2000)
42. Shamir, A.: How to share a secret. Communications of the ACM, 22:612–613 (1979)
43. Sit, E., Morris, R.: Security considerations for peer-to-peer distributed hash tables. IPTPS '02, http://www.cs.rice.edu/Conferences/IPTPS02/173.pdf (2002)
44. Wright, M., Adler, M., Levine, B., Shields, C.: An analysis of the degradation of anonymous protocols. Technical Report, Univ. of Massachusetts, Amherst (2001)