A Survey of Data Management in Peer-to-Peer Systems

ROLANDO BLANCO, NABEEL AHMED, DAVID HADALLER,
L. G. ALEX SUNG, HERMAN LI, and MOHAMED ALI SOLIMAN

David R. Cheriton School of Computer Science
University of Waterloo

Technical Report CS-2006-18

June 20, 2006

Peer-to-Peer (P2P) systems provide a decentralized infrastructure for resource sharing. In particular, file sharing was the initial motivation behind many of the first successful P2P systems. As P2P systems evolve to support sharing of structured and semantically rich data, several data management issues must be addressed. Specifically, P2P systems need to deal with data location, data integration, data querying, and data consistency issues. Data management is further complicated in P2P systems due to the volatility and scale of these systems.

In this survey we propose a reference architecture for P2P systems that focuses on the data management activities required to support sharing of structured data. Based on this reference architecture, we present and classify technologies that have been proposed to solve the data management issues particular to P2P systems.

Categories and Subject Descriptors: C.2.1 [**Computer Communication Networks**]: Network Architecture and Design—*Network Topology*; C.2.4 [**Computer Communication Networks**]: Distributed Systems—*Databases*; H.2.1 [**Database Management**]: Logical Design—*Schema and Subschema*; H.2.4 [**Database Management**]: Systems—*Distributed Databases, Query Processing*; H.2.5 [**Database Management**]: Heterogeneous Databases—*Data Translation*; H.3.4 [**Information Storage and Retrieval**]: Systems and Software—*Distributed Systems*

General Terms: Algorithms, Design, Performance, Reliability

Additional Key Words and Phrases: Data Management, Data Sharing, Integration, Peer-to-Peer

1. INTRODUCTION

Peer-to-Peer (P2P) systems are massively distributed and highly volatile systems for sharing large amounts of resources. Peers communicate through a self-organizing and fault tolerant network topology which runs as an overlay on top of the physical network. Initially developed to allow sharing of unstructured data, recently proposed P2P systems attempt to provide support for sharing structured data. Data management issues that must be addressed when dealing with the scale and volatility of such systems include:

—Data Location: peers must be able to refer to and locate data stored in other peers.

—Query Processing: given a query, the system must be able to discover the peers that contribute relevant data and efficiently execute the query.

—Data Integration: when data sources being shared in the system follow different schemas or representations, peers should still be able to access that data, ideally using the data representation used to model their own data.

—Data Consistency: if data is replicated or cached in the system, a key issue is to maintain the consistency between these duplicates.

In this survey we look at the techniques that have been proposed to address these data management issues in P2P systems. Our approach is to define a reference architecture that identifies the major components and the functionality that a peer in a P2P system will need to implement. We relate each component in the architecture to the data management issue addressed by that component and classify the research work done as it relates to the component being discussed. We assume the reader has a general familiarity with traditional database and distributed systems literature.

1.1 Reference Architecture

Figure 1 shows a reference architecture for a peer participating in a data sharing P2P system. Depending on the functionality of the P2P system, one or more of the components in the reference architecture may not exist, may be combined together, or may be implemented by specialized peers. The key aspect of the proposed architecture is the separation of the functionality into three main components: (1) an interface used for submitting the queries; (2) a data management layer that handles query processing and metadata information (e.g. catalogue services); and (3) a P2P infrastructure, which is composed of the P2P network sublayer and P2P network. In this survey, we focus on the P2P data management layer and P2P infrastructure.

Queries are submitted using a user interface or data management API and handled by the data management layer. Queries may refer to data stored locally or globally in the system. The query request is processed by a query manager module that retrieves semantic mapping information from a repository when the system integrates heterogeneous data sources. This semantic mapping repository contains meta-information that allows the query manager to identify peers in the system that store data relevant to the query and to reformulate the original query in terms

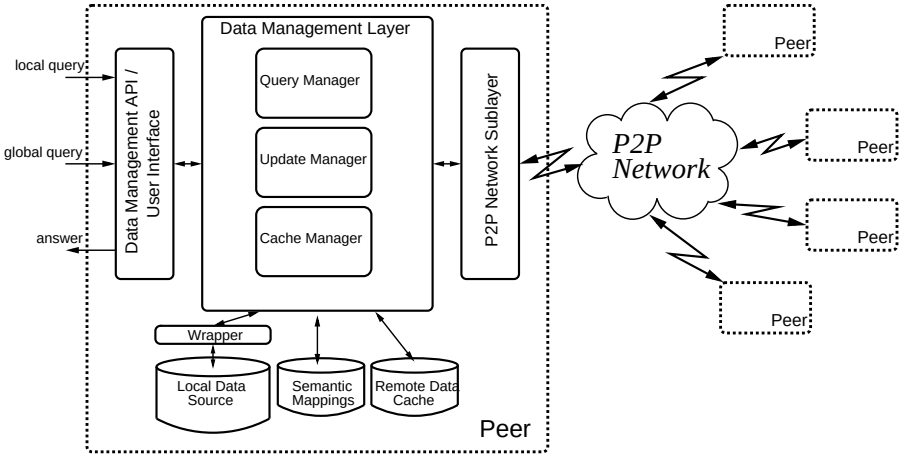


Fig. 1. Peer Reference Architecture

that other peers can understand. Some P2P systems may store the semantic mapping in specialized peers. In this case, the query manager will need to contact these specialized peers or transmit the query to them for execution. If all data sources in the system follow the same schema, no semantic mapping repository nor its associated query reformulation functionality are required at all.

Assuming a semantic mapping repository, the query manager invokes services implemented by the P2P network sublayer to communicate with the peers that will be involved in the execution of the query. The actual execution of the query is influenced by the implementation of the P2P infrastructure. In some systems, data is sent to the peer where the query was initiated and then combined at this peer. Other systems provide specialized peers for query execution and coordination. In either case, result data returned by the peers involved in the execution of the query may be cached locally to speed up future executions of similar queries. The cache manager maintains the local cache of each peer. Alternatively, caching may occur only at specialized peers.

The query manager is also responsible for executing the local portion of a global query when data is requested by a remote peer. A wrapper may hide data, query language, or any other incompatibilities between the local data source and the data management layer. When data is updated, the update manager coordinates the execution of the update between the peers storing replicas of the data being updated.

The P2P network infrastructure, which can be implemented as either structured or unstructured network topology, provides communication services to the data management layer. In unstructured networks, peers can typically join at any point in the network. Structured P2P networks on the other hand, enforce a tight control on the topology and message routing. Hence, peers joining the system are assigned specific locations in the network and are required to share some of the network responsibility.

1.2 Document Organization

Starting with the P2P network infrastructure, Section 2 compares unstructured P2P systems, and Section 3 compares structured systems. The remaining sections focus on the data management layer. Section 4 introduces the integration issues that P2P systems must deal with when sharing structured data. The discussion in this section is centered around the different ways to define and maintain semantic mappings between heterogeneous schemas in P2P systems. Section 5 focuses on the query manager module in the reference architecture, by presenting techniques for query execution. Section 6 presents data consistency issues that the cache manager and update manager must address when the P2P system supports data duplication. Section 7 concludes the survey.

2. UNSTRUCTURED PEER-TO-PEER SYSTEMS

2.1 Background

Unstructured P2P systems refer to P2P systems with no restriction on data placement in the overlay topology. The core feature of all widely deployed systems is file sharing. Some P2P systems also provide the query functionality to locate the files by keyword search. The most significant difference of unstructured P2P systems to the traditional client/server architecture is the high availability of files and network capacity among the peers. Replicated copies of popular files are shared among peers. Instead of downloading from the central server (or some mirror sites), peers can download the files from other peers inside the network. Obviously, the total network bandwidth for popular file transmissions is undoubtedly greater than the amount most central server systems able to provide.

In the following sections, we focus on data management issues in unstructured P2P systems. First of all, unstructured P2P systems are classified based on their network structure, which is determined by where files and peer indices are stored. Then, we will talk about the data management issues on top of the infrastructure. Issues regarding managing indices in terms of what is being stored, how indices are updated and how to locate files using indices, are discussed. Furthermore, peer selection, file identification and privacy problems are addressed. Lastly, we discuss the user incentives to share files and query processing power. Of the many existing P2P systems, we survey the most popular ones: Napster [Napster 2001], Gnutella (v.0.4) [Gnutella 2005], Freenet [Clarke et al. 2001; Clarke et al. 2002], FastTrack [FastTrack 2002]/KaZaA [KazaaMediaDesktop 2005], eDonkey2000 [edonkey2000 2005] and BitTorrent [BitTorrent 2005]. If not otherwise specified, the term Gnutella refers to Gnutella (v.0.4). The KaZaA network uses the commercial FastTrack protocol.

2.2 Network structure

We classify the systems into three groups according to where the file and peer indices are stored: Pure P2P, Pure P2P with supernodes, Hybrid P2P. The indices which map files to peers are used to process queries. In the pure P2P architecture, each peer stores its file indices locally and all peers are treated equally. In the pure P2P with supernodes architecture, different peers store their indices at different supernodes. In the hybrid P2P architecture, all peers store their indices on the

Table I. A comparison of different unstructured P2P systems on network structure.

P2P System	**Network Structure**
Napster	Hybrid P2P with central cluster of approximately 160 servers for all peers. [Saroiu et al. 2002]
Gnutella	Pure P2P. [Clarke et al. 2001]
Freenet	Pure P2P. A loosely structured system, where nodes maintain their own dynamic routing table mapping node address to file identifiers they are thought to hold. [Clarke et al. 2001]
FastTrack /KaZaA	Pure P2P with supernodes. [Liang et al. 2004a]
eDonkey2000	Hybrid P2P with tens of servers around the world. Peers can host their own server. [edonkey2000 2005]
BitTorrent	Hybrid P2P with central servers called Tracker. Each file can be managed by a different tracker. [BitTorrent 2005]

central server (or cluster of servers). The classification is shown in Table I and illustrated in Figure 2.

2.3 Index management

Indices are stored in the form of metadata defined by the system. Systems have their own choice of data to put into the metadata and how it is being updated. In general, the metadata includes at least information on the shared file names and sizes.

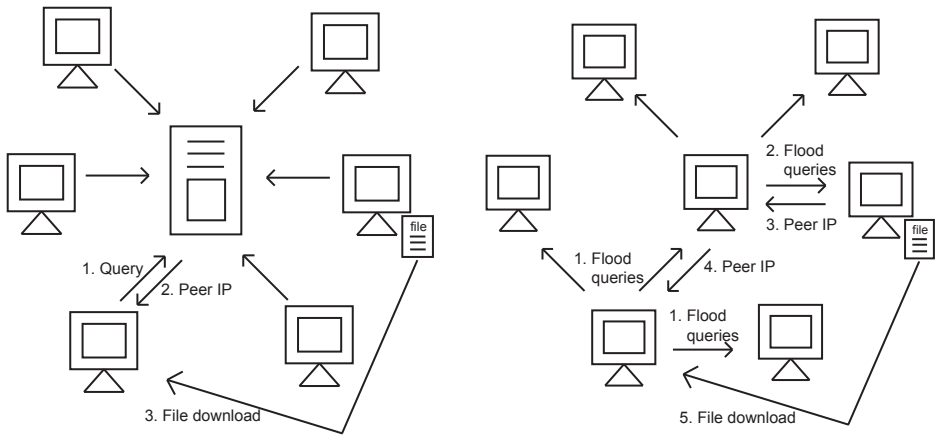
The Napster server gathers the following metadata from each peer: 1. peer-reported bandwidth; 2. number of currently shared files; 3. current number of uploads and downloads in progress; 4. the file name and size of all shared files; and 5. IP address of the peer. The metadata is uploaded every time when the peer connects to the Napster network.

The Gnutella file indices are stored locally and they depend on the implementation of the client program. Details of how to index files and match queries to local files are left undefined in the Gnutella specification.

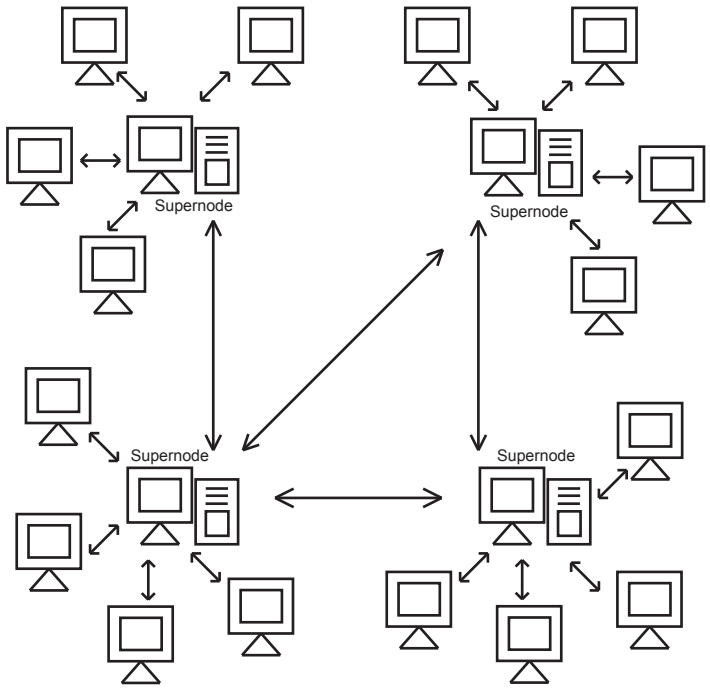
In Freenet, the shared files indices are kept locally just like Gnutella. Those files may not originally belong to the peer. Each file is indexed by its keyword-signed and signed-subspace key. For details, please refer to the Data identifier section.

The FastTrack/KaZaA metadata includes the file name, file size, last modified time, content hash and a variable number of file descriptors that describe the file content [KaZaAFileFormat 2003]. When the client program launches, the files in the shared local folder are checked against the supernode file database in order to add or remove files. The whole library of metadata is sent to the supernode. An opened KaZaA client program also monitors that folder for changes. As FastTrack is a commercial protocol, details of the protocol are unknown. [Liang et al. 2004b] finds that there is no index information exchange between supernodes. Upon query request, the supernodes sends the query to some of their connected supernodes. The supernode-supernode connection changes every tens of minutes [Liang et al. 2004a; FasttrackSuperNode 2005].

eDonkey2000 uses an ed2k link to store the metadata for clients who want to download a particular file. These are like normal html links except that they start the file downloading inside the eDonkey2000 network. The link contains the file name, file size and the file hash. The server maintains a database with file IDs mapped to server-generated client IDs. Each server maintains about 20,000 clients



(a) Hybrid P2P (b) Pure P2P



(c) Pure P2P with supernodes

Fig. 2. Network Structure of Unstructured P2P Systems

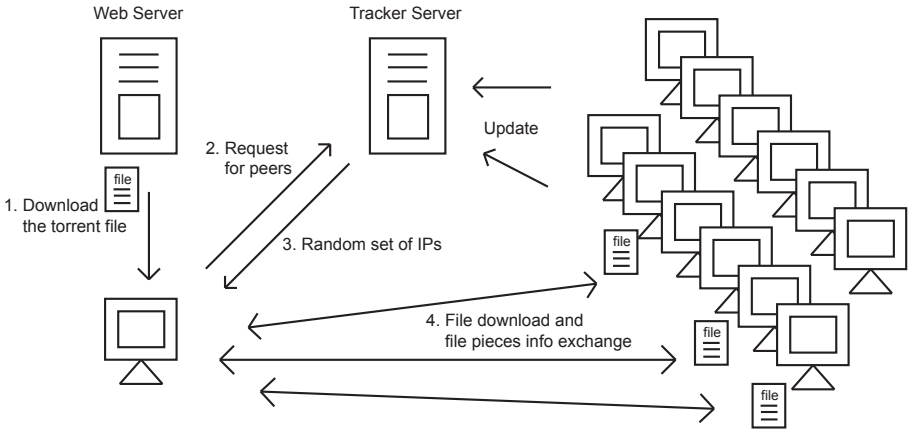


Fig. 3. Downloading a file from the BitTorrent network

[Bickson and Malkhi 2003].

The metadata for BitTorrent (BT) is stored in ".torrent" files. Torrent files are usually available at some web torrent search engines or public newsgroups. They contain information including target file name, file size, tracker server URL, size of each file piece and SHA-1 hashes for all pieces. BT client program reads the metadata file and sends an HTTP GET message to the tracker containing file ID, peer-generated peer ID, IP, port, number of bytes downloaded, uploaded and left. The number of bytes download, uploaded and left is included purely for statistical purposes on the tracker. The tracker keeps track of all peers that are downloading the same file. In this way, it can return a set of random peers who are sharing a certain file upon request. A BT client exchanges pieces IDs it has with other the connected peers. Information about pieces IDs is not available to the tracker. The above process is illustrated in Figure 3. Note that a torrent file can contain a set of file IDs. Therefore, using the torrent file, the user can download a folder which contains a set of files and sub-folders. BT is the only protocol that allows downloading a set of files at a time using one identifier.

2.4 Locating data

In unstructured P2P systems, locating data means finding a file or its pieces. The idea is analogous to key lookup in DHTs (section 3).

2.4.1 *Keyword search.* All systems discussed above, except BitTorrent, support keyword searches. Please refer to the section on Query Processing (section 5.1) for the details of how the keyword search queries are routed.

2.4.2 *File identifier.* The file identifier contains enough information to identify the file outside the P2P network. When a client program reads the identifier, it is able to download the file from the P2P network. Users can share the identifier by email or instant messenger with each other. This approach is supported by BitTorrent and eDonkey2000. BitTorrent uses the torrent files while eDonkey2000

uses the ed2k links. There is no available literature that studies the popularity of ed2k link providers for eDonkey2000.

Since BitTorrent has no querying ability, users need to get the torrent files from the web or newsgroup. [Pouwelse et al. 2005] have listed four popular torrent sites, namely suprnova.org, youceff.com, piratebay.org and lokitorrent.com. Each site had around 30,000 - 50,000 available files in October 2004. The top two sites went offline in December 2004.

2.5 Data identifier

The data here refers to the whole file or file pieces. This section focuses on how the client program identifies the data in order to resume downloading a file or finding the right file pieces from different peers.

In Napster, the user needs to download the whole file at a time from another user. Downloading from multiple sources is not supported. If the download fails, the user needs to select another peer to download the file from. Therefore, there is no need for the client program to identify the integrity of the file.

For Gnutella, a node can resume the previous incomplete downloads by sending the "resuming message". However, it is only possible when the disconnected node connects to the network again. The file identifier is an index number supplied by the peer sharing the file.

In Freenet, files are being identified by binary file keys obtained by applying the 160-bit SHA-1 hash function. There are three types of keys. The first one is the Keyword-Signed Key (KSK), which is a short descriptive text string chosen by the user. The string is used to generate a public/private key pair. The public half is hashed to yield the data file key. The private half is used to sign the data file and check the integrity of the retrieved data. As nothing prevents users from selecting the same descriptive string independently, personal namespaces are enabled by the Signed-Subspace Key (SSK). The public namespace and the descriptive string are hashed separately and they are then XOR'ed together and hashed to yield the data file key. The user publishes the namespace public key and the descriptive string together to retrieve files. The third key type is the Content-Hash Key (CHK) used for updating and splitting contents. It is derived from hashing the contents of the file. Data files are also encrypted by a randomly generated encryption key. The user publishes the CHK together with the decryption key in order to retrieve files. CHK can be used for splitting data files into multiple parts to optimize storage and bandwidth resources.

FastTrack/KaZaA hashes every file to a hash signature, which becomes the ContentHash of the file. It is the only tag used to identify a file in a download request. The ContentHash enables the KaZaA client to search for the specific file automatically, without issuing a new keyword query. The ContentHash contains 20 bytes, for which 16 bytes are calculated from an MD5 one-way digest and 4 bytes belong to the FastTrack smallhash. It is the fingerprint for file content. Files with identical content have an identical Message Digest value. The FastTrack smallhash adds more content identifying bits [KaZaAFileFormat 2003].

eDonkey2000 identifies the files using an MD4-based hash value and the file size. This method allows identifying files with the same content but different names. Each file piece is a 9500KB portion of the file. Each part is hashed individually

with the MD4 algorithm to obtain a 128-bit content hash. The hashes are combined into the unique file ID [Kulbak and Bickson 2005].

In BitTorrent, each file maps to a string whose length is a multiple of 20. The string can be subdivided into strings of length 20 bytes, each of which is the SHA-1 hash of the piece at the corresponding index. The whole string is stored in the torrent file. Each piece is by default 256KB in size.

2.6 Neighbour selection

Neighbour selection refers to how the user or the client program chooses which peers to connect to in order to download files. There are three main ways, namely manual, semi-automatic and automatic. The neighbour selection process depends on two important things that the P2P system provides: Query ability and file ID. If query ability is provided, the user needs to choose a file (and hence the peers sharing that file) among the return results based on the information provided. This part of selection is done manually. If a globally unique file ID is available in the protocol, the program can search for the peers sharing that file periodically in the background. This part is done automatically with no user involvement. If both query and file ID are available, when the a list of peers and their files are returned after a query, the user first chooses the initial peers to download from and then the client program will search for more peers later. This refers to the semi-automatic type.

2.6.1 *Manual.* In Napster, the client program displays a list of file names, file size, bit rate, user-reported connection speed, ping time, peer ID and user availability in the returned results for the user to select. User availability means the peer has an uploading connection that you can use to start downloading immediately.

Gnutella is not specifically designed for music sharing. Basic information available to the user is just file names and size. Whether other information, say Ping time and peer IP, is shown depends on the client implementation.

2.6.2 *Semi-automatic.* Both the KaZaA client and the eDonkey2000 client will return the number of sources the file has in the query result. The user can choose the file with a larger initial peer set. Higher availability files (i.e. popular files among peers) usually give a better downloading performance since you can download from multiple sources. At the beginning, each newly connected KaZaA node receives a list of 200 supernodes from its parent supernode. The list is a subset of all supernodes in the parent supernodes cache. The KaZaA client will search for new peers sharing the current downloading files in the background every 30 minutes. [Liang et al. 2004b] hypothesize that KaZaA peers mainly use two criteria, workload and locality, for neighbour selection. Workload refers to the number of connections that the supernode is supporting. For file downloads, KaZaa does not make optimal use of the available content. Users can only download from remote peers that have complete copies of the files. Peers with incomplete files are ignored [Heckmann et al. 2005].

The eDonkey2000 client can be configured to periodically search for new peers in the background every few seconds. In contrast to KaZaA, incomplete files are also shared.

2.6.3 *Automatic.* In Freenet, a user must first obtain or calculate the binary file key before he can retrieve a file. The user sends a request message to the local node specifying that key and the hops-to-live value. Hops-to-live refers to the maximum hop count before the request is dropped. When the node receives the request, it will look into the local data store first. If the data is not found, the node will broadcast the request to other nodes until the data is found and returned. As Freenet is an anonymous system, the user has no idea which files other peers are sharing. Hence, neighbour selection is completely done by the program.

In BT, the client program reads the torrent file and extracts the tracker server's address. It contacts the tracker server for a set of peers who are downloading a specific file (or group of files and folders). The tracker server will respond with a set of 50 random peer IPs. By default, if the BT client program determines that number of online peers is below 20, it will contact the tracker server for another set of 50 random peers. Peers need to contact the tracker server within every 30 minute interval. Otherwise, the peers are treated as disconnected and will be removed from the peer set maintained at the tracker server. Peer set selection is done by the server. However, the peers among the set to exchange file pieces is determined by the client program according to the Tit-for-Tat strategy, which will be discussed in section 2.8.

2.7 Privacy

[Dhamija et al. 2002] have compared the uploader/downloader anonymity, linkability and content deniability of Napster, Gnutella(v0.4 and v0.6) and Freenet. Linkability refers to identifying the correlation between uploaders and downloaders. Content deniability refers to whether the party involved can deny the knowledge on the content transmitted. [Bickson and Malkhi 2003] have studied privacy issues on Gnutella as well. They answer the following questions: Is it possible to track who is sending queries, sharing files, downloading files? Can we know if a user is connected to the network or not? Is it possible to track users based on their interest in a specific topic or their geographic location? [Good and Krekelberg 2003] have found out that users are not aware of sharing some files that may contain personal data, eg. the Outlook email database file. In this section, we compare P2P systems by the four criteria suggested in [Dhamija et al. 2002] mentioned at the beginning of this paragraph.

For Napster, every search and download status is forwarded to the Napster central server. There is no privacy at all. As the contents are transmitted in clear text, the central server, the downloader, and the uploader cannot deny knowledge on the content and linkability.

Gnutella was initially designed to support privacy. And that is why the query messages use hop-to-hop routing and the sender address is not included in the query. The uploader can identify the downloader when the TCP connection is established at the point of download, but not when replying to the query message. The modification of Gnutella(V.0.6), which creates a more centralized network using supernodes (ultrapeer nodes), makes monitoring on a large number of users possible by a relatively small number of supernodes. Users can be probabilistically tracked by their IP address, DNS name, client program version, files sharing, queries initiated, and queries answered. Both Gnutella(V.0.4) and Gnutella(V.0.6) have no down-

loader or uploader anonymity as peer identities are eventually revealed at either end of the connection. Knowledge on linkability can be denied on Gnutella(V.0.4 or V.0.6) nodes, but not the Gnutella(V.0.6) supernodes. In Gnutella(V.0.4), no party can deny content knowledge. Gnutella(V.0.6) supernodes and intermediate nodes can deny knowledge on content as they see only hashed values of file names while the uploader/downloader cannot.

In Freenet, uploader anonymity is protected by occasional resetting the data source field in response messages. The node listed as the data source does not imply that it actually supplied that data. For downloader anonymity, while a node can get some indication on how early the request message is on the forwarding chain using the hop-to-live, the true requester is kept private. As all stored file are encrypted, a node operator can deny the knowledge of the content.

In FastTrack/KaZaA, there is no anonymity for all parties. KaZaA directly returns nodes which contain files whose metadata matches the query keywords. Since, file transfers are not encrypted, no one can deny the knowledge of the content and linkability.

In eDonkey2000, same as Napster, there is no anonymity at all. The file searches are always text searches. The query string is split up into single words and all known filenames are searched for occurrences of all these words [Heckmann and Bock 2002]. No one can deny knowledge on the content and linkability.

In BitTorrent, similar to KaZaA, there is no anonymity for all parties. The tracker server returns the set of peers who are downloading the requested file. Nothing is encrypted. Nobody can deny the knowledge of the content and linkability.

2.8 Incentive to share

2.8.1 *Free riding.* The problem of free-riding was first studied by [Adar and Huberman. 2000] in 2000 on the number of files shared by Gnutella users. [Hughes et al. 2005] revisited the free-riding experiments done by [Adar and Huberman. 2000] in 2005. They showed that 85% of peers shared no files, and 86% shared 10 or fewer files where the original findings are 66% of peers share no files, while 73% of peers share 10 or fewer. Moreover, it was found that the top 1% of the sharing peers provide 50% of all query responses, while the top 25% provide 98%. The original findings are the top 1% of peers sending QUERYHIT messages were responsible for 47% of all QUERYHITs, while the top 25% of peers provided 98%. [Saroiu et al. 2002] in 2002 showed that 25% of the Gnutella clients do not share any file, and, about 75% of the clients share no more than 100 files whereas 7% of the clients share more than 1000 files. Additionally, 7% of users together share more files than all of the other users combined.

[Saroiu et al. 2002] also showed that about 40-60% of the Napster peers shared only 5-20% of the shared files. 30% of the users that reported their bandwidth as 64 Kbps or less, but they actually had a significantly higher bandwidth.

[Saroiu et al. 2002] performed a KaZaA traffic study on the University of Washington campus in 2002. It was found that 8.6% of KaZaA peers were serving 80% of the requests. Also studying the KaZaA traffic on the University of Washington campus, [Gummadi et al. 2003] in 2003 showed that the top 1000 peers(14% of all peers) among the studied 7153 peers contributed 64% of query hits. In 2004, [Liang

et al. 2004b] found that 13% of the normal peers were responsible for over 80% of the meta-data uploaded.

[Heckmann et al. 2005] showed that the average eDonkey user shares 57.8 (max. 470) files with an average size of 217 MB (max 1.2 TByte) but more than half the peers share less than 20 files.

To the best of our knowledge, there is no measurement study done on Freenet.

It is obvious that there is no clear incentive for peers to share files or query processing power. Apart from the number of files they share, users can manipulate through the client program the uploading bandwidth, number of peers to upload, and number of simultaneous upload per peer. For sharing query power, KaZaA internally determines whether the peer should become a supernode. The KaZaA 2-tier has therefore been effectively built, although it is not relied on user incentives.

2.8.2 *Incentive-Built BitTorrent.* BT is the first widely deployed system that incorporates users' incentives of sharing network bandwidth in its protocol. The incentive comes from the Tit-for-Tat-based choking (refusal to upload) algorithm used during the file exchange [Cohen 2003]. BT's choking algorithms attempt to achieve Pareto efficiency[1]. By default, downloaders will upload to the top four peers that give them the best downloading rate. Unutilized connections are optimistically unchoked (uploaded to) on a trial basis to see if better transfer rates could be found. The choking evaluation is performed every 10 seconds. The decision on which peers to un/choke is based solely on the download rate, which is evaluated on a rolling, 20-second average. The optimal strategy is to offer maximum uploading rate. Free riders are strongly discouraged as they will get choked quickly. One thing to note is that users may not have incentive to share a file to the BT network or share their uploading bandwidth once they have finished downloading.

2.9 Latest Work and Summary

Recently, there was a beta-version release of a commercial P2P system eXeem [eXeem 2005] (version 0.22 at the time this paper was written) that is gaining popularity. In general, it uses BT's idea to build incentive into the system and add a query function on it. That means users can share files and let others query for them just like any traditional unstructured P2P system. Users IPs are hidden from the client program interface to increase privacy. As it is a commercial product, not much detail is known.

From what we have shown above, P2P systems are evolving. The index storage location has shifted from the central server to normal peers and then to supernodes, which is more scalable. File identifiers have become globally unique. Neighbour selection has moved from manual to automatic. Sharing incentives have been injected into the system. Privacy issues have been taken into account, but not fully addressed. We believe that later generations of unstructured P2P systems will be based on supernodes, use global file IDs, select neighbours automatically, provide anonymity and, most importantly, have built-in incentive of sharing files, network bandwidth and query processing power.

[1]A change that can make at least one individual better off, without making any other individual worse off is called a Pareto improvement. An allocation of resources is Pareto efficient when no further Pareto improvements can be made.

3. STRUCTURED PEER-TO-PEER SYSTEMS

3.1 Overview of DHTs

Structured P2P systems are also referred to as Distributed Hash Tables (DHTs). Although, in theory, hybrid P2P systems also maintain some kind of structure over the network (as indicated in Section 2) they are not classified under the umbrella of structured P2P systems. Thus, the focus of this section is on P2P systems that employ DHTs as the underlying framework. The evolution of research in DHTs was motivated by the poor scaling properties of unstructured P2P systems [Ritter 2001; Ratnasamy et al. 2001; Stoica et al. 2001]. Thus, a key contribution of DHTs is *extreme scalability* [IRIS 2005]. As a result, DHTs exhibit some very unique properties that traditional P2P systems lack. In particular, they provide perfect *recall* and high *precision*[2]. As a result, there have been many proposals to further enhance the functionality of DHTs. And with the growing list of benefits and applicability of DHTs, many proposals to enhance them have been studied. In an attempt to better understand the most fundamental contributions to this field, a general taxonomy for DHTs is introduced in this section.

In order to truly appreciate the strength of the DHT approach, it is important to understand the differences between structured and unstructured P2P systems. At a fundamental level, DHTs can be characterized simply as data addressing and lookup engines. Taking traditional P2P file-sharing systems as an example, two important features encapsulate the basic functionality of these systems; namely, (1) Data Lookup, and (2) Data Retrieval. Having realized this basic functionality, DHTs aim to optimize data lookup in a P2P systems by implementing a virtual overlay network over the underlying physical network. Thus, DHTs are not involved in the actual retrieval of the data itself, and only support efficient lookups[3].

The remainder of this section is organized as follows: in Section 3.2, we briefly discuss some application-level issues for DHTs, Section 3.3 presents a discussion on a general architecture for DHTs, Section 3.4 discusses the implications of using virtual overlays as part of the DHT approach, Section 3.5 presents a taxonomy for DHTs, and finally Section 3.6 concludes with a summary.

3.2 Application-level Issues in DHTs

In the past four years, a tremendous amount of research has gone into creating applications for DHTs. DHTs have been used in applications ranging from file-systems [Karp et al. 2004; Dabek et al. 2001; Rowstron and Druschel 2001b; Kubiatowicz et al. 2000], and event notification services, to content distribution [Castro et al. 2003], indirection services [Stoica et al. 2002; Cox et al. 2002], and large-scale query processing engines [Huebsch et al. 2003]. However, there still has not been a widespread commercial adoption of DHTs, due in much part to the limited API flexibility support for such systems. A majority of the research focus in

[2]Precision refers to the proportion of correct results that the system returns, while recall refers to the proportion of the results that are returned out of the total results that are contained in the system

[3]However, some systems such as the one mentioned in [Harvey et al. 2003] do augment them to provide data retrieval functionality as well

DHTs has been in optimizing the lookup service: given a key, map the key onto a node. Extensions to this basic service have been proposed [Dabek et al. 2001], but application flexibility has been minimal. This, as the DHT community later realized, became a major hurdle in the adoption of DHTs, as many researchers in the DHT community itself were faced with such issues. Additionally, all previous DHT proposals have mostly advocated participating in the system as a DHT node, which has made adoption of the technology even harder. Very recently, however, DHT services like OpenDHT[4] have been deployed for researchers and the general public alike as a means of experimenting with such systems. OpenDHT has been maintained strictly as a *service*[5] in order to encourage interest of users that only want to experiment with the system.

Several initiatives have been proposed to address issues related to the adoption of DHT technology. One major initiative is Project IRIS (Infrastructure Resilience for Internet Systems). IRIS aims at providing a general platform (Internet scale) based on DHTs that enables the creation of new types of distributed applications. These and other efforts (e.g. OpenDHT) are being well received, and a huge initiative is underway to create systems that are practically realizable and allow deployment at a larger scale. It is important to note that by virtue of being network overlays, DHTs are not hindered by deployment issues that have plagued many networking research initiatives in the past as discussed in Section 3.4.

3.3 Architecture of DHTs

DHTs implement the following basic components: (1) Data/Node Addressing (i.e. a hash function), (2) Routing Protocol (and corresponding overlay structure), and a (3) Routing State Maintenance Protocol. We briefly discuss these components further:

(1) **Data/Node Addressing**
As discussed earlier, since the DHT approach employs a virtual overlay network, it is important to provide addressing (i.e. hashing) of the data and the nodes in order to support lookups in the overlay. Intuitively, a naïve technique could be to use the URI of the content/data for the data and a node's IP address for the node, and indeed, some DHTs [Harvey et al. 2003] do use such naming schemes. However, it is important to note that in order to provide a uniform distribution of data in the overlay, URIs/IP addresses do not provide sufficient flexibility. As a result, *consistent hashing* techniques that provide uniform hashing of values are used to evenly place the data on the overlay. Although many hash functions may be employed for generating *virtual address mappings* for the data (e.g. proximity-aware, locality-aware functions), SHA-1 has become the most widely accepted *base*[6] hash function that supports both uniformity as well as security (by supporting data-integrity for the keys). The result of the hash

[4]Currently running on the PlanetLab testbed
[5]Service refers to the ability to utilize OpenDHT without having to install any software or explicitly join the system
[6]A base hash function is defined as a function which is used as a basis for the design of another hash function

is a *key* that maps the address of the data to a particular node in the DHT. However, the actual design of the hash function is implementation dependent and thus is not explored any further in subsequent sections.

(2) **Routing Protocol**
The most important task in a DHT is the lookup process. This lookup involves hashing the key of the data item (refer to (1)) onto the node that is responsible for the key. The lookup is then initiated on the overlay network to locate the target node in question. This is referred to as the routing protocol for the DHT; it differs between implementations and is closely associated with the overlay structure used. A more detailed discussion will be presented in Section 3.5, but it suffices to say that all routing protocols attempt to not only provide efficient lookups but also minimize the routing state that needs to be maintained in the overlay.

(3) **Routing State Maintenance Protocol**
A routing protocol also requires each of the nodes to maintain routing state. This routing information differs between routing protocols and overlay structures. However, all implementations require the use of maintenance algorithms in order to keep the routing state up-to-date and consistent. In contrast to routers in the Internet that also maintain routing databases, P2P systems pose a greater challenge since they are characterized by high node volatility and undependable network links. Since DHTs also need to support perfect recall, routing state consistency becomes a key challenge for such systems. Therefore, they must be able to maintain consistent routing state in the face of concurrent lookups and also during periods of high network volatility.

3.4 Network Overlay Issues in DHTs

In the past few years, overlay networks (ONs) have become one of the most popular tools in both network research and development. They provide a great deal of flexibility in supporting new tools and services. Due to the inherent ossification of the IP routing infrastructure, the importance of ONs has become even more crucial. Network overlays are generally regarded as abstract tools for networking problems [Birman 2000]. More recently [Andersen 2005; Jain et al. 2003], they have been classified into three basic types: routing/measurement-based overlays, storage/lookup-based overlays, and random power-law overlays. In the context of P2P systems, DHTs fall into the category of lookup-based overlays, although some unstructured P2P systems such as Gnutella[7] employ "naturally emerging overlays" that follow the power law for random graphs[Faloutsos et al. 1999].

However, overlays in general and lookup-based overlays in particular, have their own characteristic set of challenges that are often ignored by many DHT implementations. Fundamentally, since they abstract themselves from the physical links of the underlying network, DHTs tradeoff many desirable features of the network: (1) network performance, (2) security, (3) management complexity, and (4) practical Internet routing concerns (e.g compatibility with NATs, Firewalls, etc). As indicated, in Section 3.2, since research efforts are underway to expand the scale and applica-

[7]Here we only refer to pure P2P systems, not Hybrid systems as indicated in Section 2

DHT	Space Complexity (No. of Nodes)	Time Complexity (No. of Hops)
Chord	O(log(n))	O(log(n))
Tapestry (b=base of IDs)	$O(\log_b(n))$	$O(\log_b(n))$
CAN (d=no. of dimensions)	$2d$	$O(n^{1/d})$
Pastry (2^b=base of each digit in IDs)	$O(\log_{2^b}(n))$	$O(\log_{2^b}(n))$
Viceroy	7	O(log(n))
Koorde (Degree-2 DeBruijn)	2	O(log(n))
Kademlia	O(log(n))	O(log(n))
Structured SuperPeers	$O(\sqrt{n})$	O(1)

Table II. Computational Complexity of some DHT Implementations

bility of DHTs, in the long term, these challenges will need to be addressed. Some of these challenges (e.g. those dealing with performance), however, have already been well-studied. Recently, [Jain et al. 2003] did a simulation study to compare the performance of several DHT implementations augmented with different routing and topology-aware construction heuristics. This study is fairly extensive since it provides a classification for all types of performance enhancing heuristics (as *routing heuristics* or *topology-aware overlay construction heuristics*), and presents characteristic upper bounds for each class. It provides a suitable framework in which to compare any kind of performance heuristic. The authors compare popular DHT implementations augmented with different performance optimizations (using metrics such as *Relative Delay Penalty* and *Link Stress*) and show that the use of good heuristics can drastically improve the performance of DHTs, making them a viable option for large scale deployment. Some other DHT issues indicated above, however, are still open questions in the research community and are currently being investigated.

3.5 DHT Taxonomy

Many DHT routing overlays have been proposed over the years. [Gummadi et al. 2003] use the terms *routing geometry* and *routing algorithm* to classify the different DHT overlays. Routing geometry essentially defines the manner in which neighbours and routes are arranged. The routing algorithm corresponds to the routing protocol discussed earlier and is defined as the manner in which next-hops/routes are chosen on a given routing geometry. Due to its wide acceptance in the DHT community, we will present the taxonomy discussed in [Gummadi et al. 2003].

3.5.1 *Basic Routing Geometry.* Many routing geometries have been proposed for DHTs and still many more are being explored today. However, in general, most routing geometries exhibit similarities and can be easily classified into a small representative set of routing structures. Although we do not claim that this classification is exhaustive, we do point out that the presented taxonomy provides a good coverage of most of the fundamental work that has been done on routing geometries for DHTs. The classification of these routing geometries is discussed further with the help of case examples of some popular implementations that have been proposed in each class.

At the overlay level, most DHT implementations exhibit almost similar space

and time requirements (with the exception of a few that provide tighter bounds [Malkhi et al. 2002; Kaashoek and Karger 2003]). Therefore, in order to provide a more informed discussion on the different geometries, we compare them by their *degree-of-flexibility*. We use two main criteria, *neighbour selection flexibility* and *route selection flexibility*. Neighbour selection is defined as the amount of flexibility that a node has in choosing its neighbours based on the geometry. A high neighbour selection flexibility can allow easier augmentation of different optimization heuristics to the routing geometry (e.g. for path locality, content locality, etc). Route selection corresponds to the choice of next-hop neighbours that a node has to choose from when performing routing/lookup (once the neighbour selection criterion has been defined). In this regard, fault resilience is directly correlated with route selection flexibility. Therefore, a low route selection flexibility corresponds to poor fault-tolerance characteristics of the routing geometry but possibly also a lower routing state maintenance overhead. However, this may not be true in all cases since basic routing geometries may be augmented with additional capabilities (Viceroy and CAN have been augmented with *sequential neighbours*[8]). Nevertheless, for the sake of this discussion, we omit such modifications here. For illustration purposes, Table II presents the space and time complexities of some popular DHT implementations.

(1) **Tree**

The binary tree structure is probably the first routing geometry that had been proposed for DHTs (PRR)[Plaxton et al. 1997]. In the tree approach, the leaf nodes of the tree correspond to the node identifiers that store the keys to be searched. The height of the tree is log(n), where n is the number of nodes in the tree. The search proceeds down the tree by doing a longest prefix match at each of the intermediate nodes until the target node is found. Therefore, in this case, matching can be thought of as correcting bit values from left-to-right at each successive hop on the tree. A popular DHT implementation that falls into this category is Tapestry [Zhao et al. 2004]. It uses *surrogate routing*[9] in order to forward requests at each node to the closest digit in the routing table. The tree structure for Tapestry, however, is slightly different from PRR[Plaxton et al. 1997]. In Tapestry, each unique identifier is associated with a node that is the root of a unique spanning tree used to route messages for the given identifier. Therefore, in Tapestry, lookups proceed from the base of the spanning tree all the way to the root node of the identifier. Nevertheless, the Tapestry routing geometry is very closely associated to a tree structure and we classify it as such. Therefore, in further discussions, Tapestry is assumed to be classified into the tree routing geometry.

When analyzing the flexibility of the tree structure, we first observe its flexibility in terms of neighbour selection. Any node in the system has 2^{i-1} nodes to choose from as its neighbour from the subtree whose $\log(n-i)$ prefix bits are in common with it. The number of potential neighbours increases exponentially

[8]A Sequential neighbour is a neighbour that makes progress towards all destinations from that node

[9]Surrogate routing is defined as routing to the *closest* digit when an exact match in the longest prefix cannot be found

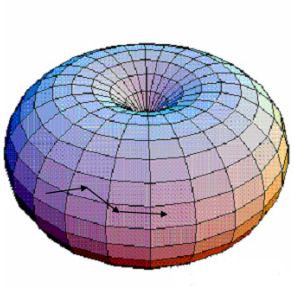


Fig. 4. An Example of a *torus* for a 2-dimensional coordinate space. The arrows indicate an example path that could be followed during a lookup.

as we proceed further up in the tree. Thus, in total we have $n^{\log(n)/2}$ possible routing tables per node (Note, however that, we can only select one such routing table for the node). Therefore, the tree geometry has good neighbour selection characteristics. However, routing can only be done through one neighbouring node when sending to a particular destination. Therefore, as a result, the tree structure does not provide any flexibility in the selection of routes.

(2) **HyperCube**

The Hypercube routing geometry is based on a d-dimensional Cartesian coordinate space which essentially resembles a d-torus (Figure 4). The coordinate space is partitioned into an individual set of zones such that each node maintains a separate zone of the coordinate space. The number of neighbours that a node may have in a d-dimensional coordinate space is $2d$ (for the sake of discussion, we consider $d = \log(n)$). If we consider each coordinate to represent a set of bits, then each node identifier can be represented as a bit string of length $\log(n)$. In this way, the Hypercube geometry is very similar to the Tree since it also simply *fixes* the bits at each hop to reach the destination. However, in the HyperCube, since the bits of neighbouring nodes only differ in *exactly* one bit each forwarding node needs to modify only a single bit in the bit string, which can be done in any order. Thus, if we consider the correction of the bit string, the first correction can be applied to any $\log(n)$ nodes, the next correction can be applied to any $\log(n) - 1$ nodes, etc. Therefore, we have $(\log(n))!$ possible routes between nodes which provides high route flexibility in the HyperCube routing geometry. However, each node in the coordinate space does not have any choice over its neighbours coordinates since adjacent coordinate zones in the coordinate space can't change. Therefore, Hypercubes have poor neighbour selection flexibility.

An example of HyperCubes is the Content Addressable Network (CAN) [Ratnasamy et al. 2001]. Due to the large route selection flexibility of HyperCubes and the resulting high-fault tolerance, CAN supports a very lightweight route maintenance protocol that does not incur huge overheads during periods of high volatility. For example, if a node fails, the CAN maintenance protocol only requires updating of nodes that are immediate neighbours of the failed node, without requiring updates to any other nodes in the network.

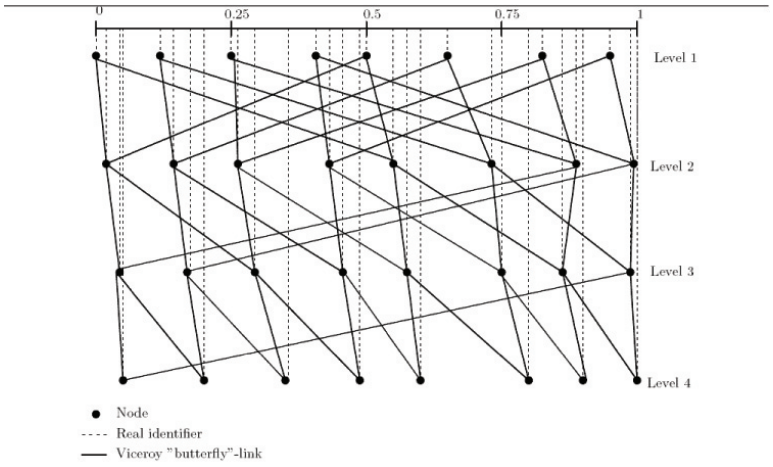


Fig. 5. An Example of a Butterfly Network (*Image from [Malkhi et al. 2002]*)

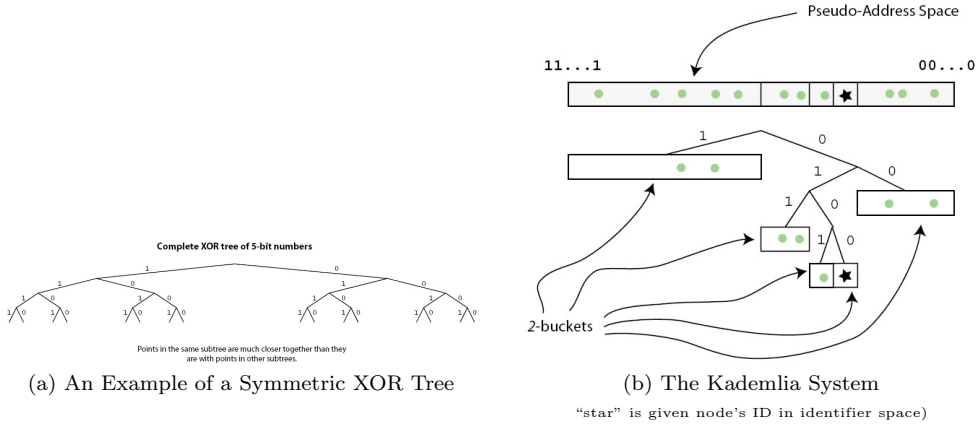
(3) **Ring**

The Ring geometry is represented as a one-dimensional circular identifier space where the nodes are placed at different locations on the circle. The distance between any two nodes on the circle is the numeric identifier difference (clockwise) around the circle. Since the circle is one-dimensional, the data identifiers can be represented as single decimal digits (represented as binary bit strings) that map to a node that is closest in the identifier space to the given decimal digit. Chord is an implementation that employs the use of the ring geometry. Specifically, in Chord, node a maintains information about $\log(n)$ other neighbours on the ring where the i^{th} neighbour is the node closest to $a + 2^{i-1}$ on the circle. Using these fingers, Chord is able to route to any other node in $\log(n)$ hops.

If we analyze the structure of Chord carefully, we observe that a node does not necessarily need to maintain the node closest to $a + 2^{i-1}$ as its neighbour. In fact, we can still maintain the $\log(n)$ lookup bound if we take any node from the range $[(a + 2^{i-1}),(a + 2^{i})]$. Therefore, in terms of route flexibility, overall, we are able to select between $n^{\log(n)/2}$ routing tables for each node. This provides a great deal of neighbour selection flexibility. Moreover, for routing to any node, the first hop has $\log(n)$ neighbours that progress to the destination and the next node has $\log(n)$ - 1 nodes, and so on. Therefore, there are typically $(\log(n))!$ possible routes to the destination. Therefore, the ring geometry also provides good route selection flexibility.

(4) **Butterfly**

The Butterfly geometry essentially resembles traditional butterfly networks and extends this structure to support the scalability requirements of DHTs. Moreover, it improves on existing implementations since it only requires a constant space per node where as traditional DHTs usually require $O(\log(n))$ space per node (refer to Table II). The Viceroy [Malkhi et al. 2002] algorithm uses the



(a) An Example of a Symmetric XOR Tree

(b) The Kademlia System

"star" is given node's ID in identifier space)

Fig. 6. The XOR Topology (*Image from [Maymounkov and Mazieres 2002]*)

butterfly geometry to support efficient lookups (see Figure 5). The nodes are partitioned across a set of levels and are subsequently connected at each level to each other in the form of a ring (using successor/predecessor pointers). The identifier space is mapped in the range of [0-1]. A lookup in Viceroy proceeds in three stages: (1) The request climbs up the connections to a level-1 node (O(log(n)) hops), (2) It descends down the levels until it reaches a level containing the destination node (also O(log(n)) hops), successively decreasing the search space by 1/2, and finally, (3) It traverses the ring geometry *sequentially* to the destination node (O(log(n)) hops). This technique allows Viceroy to maintain only state size of 7 per node for the network.

However, we note that since the 3^{rd} stage of the lookup involves a sequential search of the node, it does not provide route or neighbour selection flexibility. This can be improved by maintaining additional state at the node, however, as indicated earlier, we only attempt to compare basic DHT implementations in our discussion.

(5) **XOR**

The XOR approach is the only approach discussed here that employs a symmetric unidirectional topology (Figure 6). This topology allows the DHT to receive lookup queries from precisely the same nodes that are contained in a particular node's routing table. This allows *in-band* exchange of routing state that is not possible using other approaches (i.e. lookup requests can also serve as routing state messages for route maintenance). The Kademlia [Maymounkov and Mazieres 2002] system utilizes this topology and is the first DHT to exploit the fact that node failures in a P2P system are inversely proportional to their respective uptimes [Saroiu et al. 2002].

The XOR topology computes the distance between any two nodes as the XOR of the node's identifiers. Each of the nodes maintain routing state for nodes in the interval [(a + 2^{i-1}),(a + 2^i)] in a set of k-buckets (Figure 6), where each bucket is sorted by the last seen node (in terms of node requests). Each successive bucket contains sets of nodes that are further away from the given

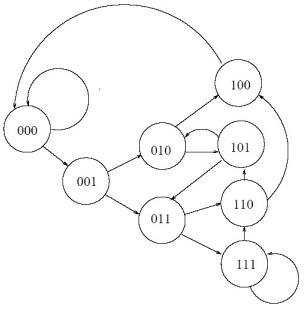


Fig. 7. An Example of a deBruijn graph (*Image from [Kaashoek and Karger 2003]*)

node (based on the XOR metric). By doing so, Kademlia provides the same kind of neighbour selection capabilities as the tree approach. However, in terms of route selection, the XOR topology does not require strict *left-to-right* bit fixing, and can still make progress towards the destination using other nodes. However, the arbitrary fixing of bits may not guarantee preservation of low-order bits that have been fixed as higher-order bits are fixed.

(6) **Hybrid**

Hybrid geometries typically consist of multiple instantiations of previously discussed *basic* geometries. Pastry [Rowstron and Druschel 2001a] is a popular DHT implementation that uses a combination of tree and ring geometries by default, in order to provide *proximity* properties. As it will be described later, the proximity metric takes into account the closeness of the nodes in the underlying physical network in order to improve the lookup efficiency. Pastry achieves this by requiring each node to maintain a *routing table*, a *leaf set*, and a *a neighbourhood set*. The routing table is analogous to the tree structure described previously. However, the leafset acts as the ring during the routing in Pastry. The neighbourhood set is used to maintain locality properties.

As we analyze the corresponding routing geometry for Pastry, we find that it provides the same degree of flexibility in terms of neighbour selection as the tree geometry described above. This is due to the fact that Pastry utilizes the tree structure as a first step in its lookup procedure and only falls-back to the ring if routing in the tree fails. However, in terms of route selection flexibility, although the hybrid structure does provide a greater selection of routes then a simple tree-structured approach, the routing is more akin to the kind of route selection flexibility that is seen in Kademlia (in terms of preservation of fixed bits). However, since Pastry maintains an ordered leafset as well at each node, it is able to "hop" between nodes with the same prefix (i.e. between branches of the tree) and still preserve the bits that were *fixed* previously.

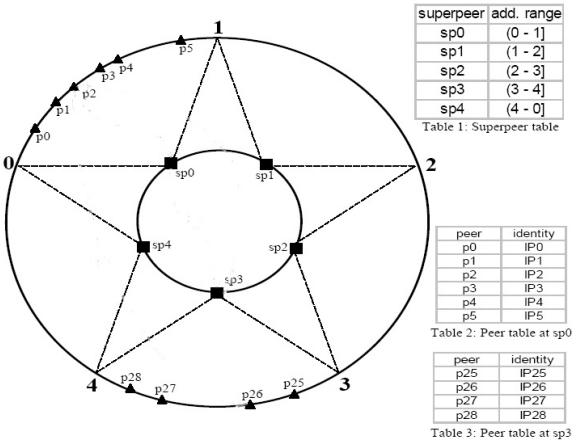
Nevertheless, in general, its routing protocol and routing geometry are much more involved than approaches we described previously.

3.5.2 *Extended Routing Geometry.* In continuing with the discussions presented in the previous section, we briefly discuss some extensions to the basic geometries in order to improve the routing properties of the previous algorithms.

(1) **SkipNet**
SkipNet [Harvey et al. 2003] is an extension of the ring approach that combines the ring structure with the idea of SkipLists, first proposed by [Pugh 1990]. A SkipList is a sorted linked list that contains supplementary pointers at some nodes that facilitate *large jumps* in the list in order to reduce the search time of a node in the list. This idea is applied to the ring structure, where nodes maintain supplementary pointers in a circular identifier space. This technique is able to provide the same scalability of traditional DHTs and also supports locality properties. Specifically, SkipNet facilitates placement of Keys based on a nameID scheme that allows the keys to be stored locally or within a confined administrative domain (*content locality*). In addition, SkipNet also provides *path locality* by restricting the lookups in the DHT only to domains that may contain the required key.
However, since this technique provides (*content locality*) and (*path locality*), it trades-off the fault-tolerance of the underlying ring since the content is not uniformly distributed across the ring. Additionally, load balancing issues may also arise in such a system. [Ganesan et al. 2004] also discuss a similar system that imposes a hierarchy on Chord to provide locality and support for service composition.

(2) **Koorde**
Koorde [Kaashoek and Karger 2003] is also an extension of the Chord protocol and implements deBruijn graphs on top of the ring geometry. An illustration of deBruijn graphs is shown in Figure 7. A deBruijn graph maintains two pointers to each node in the graph, thereby requiring only constant (i.e. 2 nodes) state per node in the ring. Specifically, given that each node ID is represented as a set of binary digits, each node is connected to nodes with identifiers $2m$ and $2m + 1$ (where m is the decimal value of the node's ID). These operations can be regarded as simple left shifts and additions of the given node's ID. Therefore, by successive shifting of bits, lookup times of $\log(n)$ can be maintained. If instead of 2, we consider a degree-k deBruijn graph where k=$\log(n)$, the effective lookup is tightly bounded by $O((\log n)/\log\log n)$. Therefore, Koorde is able to improve on the scalability properties of Chord while at the same time maintaining the same amount of state per node. It is worth noting that Koorde's route maintenance protocol is identical to Chord's and therefore it shares the same fault-tolerance capabilities as Chord.

(3) **Structured Superpeers**
Structured Superpeers [My'zrak et al. 2003] also utilizes the Ring routing structure but augment it with *super peers*. Super peers in this context are considered to be the same type as discussed in the unstructured P2P context. Fundamentally, Structured Superpeers are enhanced/higher priority nodes that are considered more reliable and of higher capacity than regular peers in the network. These super peers also maintain additional information on the order of $O(\sqrt{n})$ on all of the other super peers connected on a separate "out-of-band"



superpeer	add. range
sp0	(0 - 1]
sp1	(1 - 2]
sp2	(2 - 3]
sp3	(3 - 4]
sp4	(4 - 0]

Table 1: Superpeer table

peer	identity
p0	IP0
p1	IP1
p2	IP2
p3	IP3
p4	IP4
p5	IP5

Table 2: Peer table at sp0

peer	identity
p25	IP25
p26	IP26
p27	IP27
p28	IP28

Table 3: Peer table at sp3

Fig. 8. Layout of Structured SuperPeers structure. The inner ring corresponds to the out-of-band ring connecting the Super Peers. Each super peer is responsible for peers on the outer ring corresponding to the intervals presented on the outer ring; Sp0 is responsible for the interval between [0-1] on the outer ring and so on. (*Image from [My'zrak et al. 2003]*)

ring (Figure 8). Each super peer is also responsible for a portion of the original ring that it is associated with. Lookups are initiated at a super peer and require only 2 hops (i.e. constant time) to reach the destination. This significantly improves the lookup time over traditional approaches. However, these improvements come at the cost of additional storage space and reduced fault-tolerance. Specifically, a super-peer failure is akin to a drastic failure whereas a regular peer node failure is handled in the same way as traditional ring-based routing algorithms.

Although many other DHT extensions have been proposed over the years, we have attempted to highlight the most prominent work in this area. It is not a coincidence that many of these extensions have been based specifically on the ring geometry. In recent years, the DHT community has begun to recognize the merits of the ring-based approach and as a result much of the future work is focused on this particular routing structure.

3.6 Summary

Structured P2P systems/Distributed Hash Tables have been studied in great depth in the past few years. DHTs exhibit *extreme scalability* properties and support tremendous potential for future applications. However, as has been outlined, many challenges still exist for DHTs both from a systems as well as an applications perspective. Nevertheless, over the years, DHTs have received much scrutiny both from algorithm designers as well as systems builders [Hellerstein 2003]. As a result, many research boundaries have been chalked out and it has become widely accepted that a *Ring* structure is the best compromise that supports many of the properties we desire from such systems. Future research in DHTs is most definitely following this path as such technologies become more widespread and a fundamental part of

University of Waterloo (UW):
- Areas(area_id, name, description)
- Projects(proj_id, area_id, name)
- Publications(pub_id, title, year, howpublished)
- AuthorPublications(author_id, pub_id)
- ProjectPublications(proj_id, pub_id)
- Researcher(res_id, name)
- ProjectMembers(res_id, proj_id, role)

University of Toronto (UT):
- Project(projID, name, desc)
- Student(studID, fname, lname, status)
- Faculty(facID, fname, lname, rank, office)
- ProjMember(projID, memberID)
- Paper(paperID, title, forum, year)
- Author(authorID, paperID)

The University of British Columbia (UBC):
- Area(areaId, name, descr)
- Project(projName, sponsor)
- ProjArea(projName, areaID)
- Pubs(pubID, projName, title, venue, year)
- Author(pubID, author)
- Member(projName, member)

...

Fig. 9. Research Project Database Schemas

massively distributed systems like the Internet.

4. DATA INTEGRATION

The autonomous nature of the peers in P2P systems, the potential heterogeneity of data schemas, and the unwillingness to share the schema information make sharing structured data in P2P systems a challenging and difficult problem. Moreover, semantic issues are introduced when data schemas among peers overlap or are related. As shown later in this section, traditional approaches for data sharing in federated and other distributed database environments make assumptions that no longer hold in P2P systems, or suffer from scalability problems when applied to systems with massive number of participants. Furthermore, data integration techniques developed for P2P systems need to consider and cope with the volatile nature of the peers. Data sharing is, nevertheless, the motivation behind many of the most popular P2P applications, and support for structured data sharing seems to be the natural evolution for the functionality offered in these types of systems. The research work discussed in this section relates to the query manager component in our reference architecture and the semantic mapping repository, see Figure 1.

Example 4.1: To illustrate some of the issues in P2P data sharing, assume a P2P system that integrates research project data from diverse universities (example adapted from [Halevy et al. 2004]). In such a system different universities may store the data using different schemas and models (e.g. some using XML, others using relations). Figure 9 shows the data schemas that some of the universities participating in such a system could have. For simplicity, all schemas in this example are assumed to follow the relational model.

UW may want to be able to access UT's research database and vice versa. Semantically, the databases store the same type of data. Moreover, if UW and UT participate in some common projects, the information about these joint projects may appear in both databases. Assuming that UW has defined a semantic mapping between its own database and UT's, when UBC joins the system, UBC will be able to access UT's data by creating a semantic mapping between its own database and UW's. The willingness to expose one's schema may limit the mechanisms used to specify and implement semantic mappings. Different systems, make different assumptions regarding the amount of schema information peers provide. The general

```
{
    UW.Projects.proj_id ⟼ UT.ProjectID,
    UW.Researcher.name ⟼ concat(UT.Faculty.fname, ' ', UT.Faculty.lname),
    UW.Researcher.name ⟼ concat(UT.Student.fname, ' ', UT.Student.lname)
}
```

Fig. 10. Example Schema Mappings

consensus is that peers should be able to expose only the portions of their database that they wish to contribute to the system.

Formally, a semantic mapping describes the relationships between the terms used in two or more schemas for the purpose of sharing and integrating data [Madhavan and Halevy 2003]. Two techniques are used in P2P systems to define semantic mappings: *schema mappings* and *data mappings*. Both techniques are presented in this section. Schema mappings are implemented when different schemas use different names or formalisms to represent the same data. Data mappings are used when semantic differences between the schemas make the schema mapping inapplicable. Hence, schema and data mappings complement each other.

4.1 Schema Mappings

Schema mappings define the semantic equivalence between relations and attributes in two or more different schemas. Some possible schema mappings for Example 4.1 are shown in Figure 10.

Schema mappings may or may not be one-to-one and reflexive. They are nevertheless transitive. As with traditional distributed databases, the purpose behind integrating data using schema mappings in P2P systems is to provide a uniform querying environment that hides the heterogeneity and distribution of the data sources. Hence, once the mappings from UW to UT's database are completed, users at UW should be able to specify queries on UW's schema and receive data from both UW and UT's databases as part of the query results.

Typically, schema mappings are specified manually, but several approaches for automating these tasks have been proposed [Rahm and Bernstein 2001].

4.1.1 *Traditional Approaches.* The traditional approach to data sharing has been the definition of a mediated schema between data sources [Wiederhold 1992]. This mediated schema provides a global unified schema for the data in the system. Users submit their queries in terms of the mediated schema, and schema mappings between the mediated schema and the local schemas allow the original query to be reformulated into subqueries executable at the local schemas [Xu and Embley 2004]. Hence, although no data are stored at the mediated schema (just metadata), the mediated schema is queried as if it stored the data. Wrappers close to the local sources may need to be implemented to provide translation services between the mediated schema and the local query language, model, or any other local feature incompatible with the mediated schema [Ullman 1997]. Mediated schemas can be constructed based on other mediated schemas, effectively forming a *semantic tree* where the schema of a node is mediated by its parent.

Strategies for the definition of the schema mappings between the mediated and local schemas include: *global-as-view (GAV)*, *local-as-view (LAV)*, and *global-and-local-as-view (GLAV)* [Lenzerini 2002]. In the GAV approach, the mediated schema

is described based on the local sources. In the LAV approach the local sources are described based on the mediated schema. When GLAV is used, the integration between the mediated and local schemas is performed by using both GAV and LAV.

4.1.2 *Schema Mappings in P2P Systems.* Given the characteristics of P2P systems, the definition of a unique global mediated schema is impractical. The main issues are:

(1) Volatility: as peers join and leave the system, the global mediated schema would need to be modified to reflect the data that are made (un)available.

(2) Peer autonomy: some of the participants may be willing to share all their data while others may be interested only in contributing a portion to the system.

(3) Scalability: assuming the existence of a global mediated schema, the key issue becomes the allocation of this schema. If the mediated schema is centralized, the hosting site may require a substantial investment in hardware and connectivity. Moreover, the hosting site becomes a single point of failure. If the mediated schema is distributed, the issue becomes the development of techniques to guarantee an integrated view of the mediated schema. Yet another possibility is to fully replicate the schema, potentially imposing unacceptable storage requirements on peers. When the mediated schema is updated, coordination among peers will be required, thereby increasing the complexity of the system.

Due to these issues, most P2P systems avoid the requirement for a unified mediated schema. Instead their approaches can be categorized in one of three groups: *pair mappings*, *peer-mediated mappings* and *super-peer mediated mappings*.

Pair mappings: The simplest approach to schema mappings in P2P systems is to implement mappings only between pairs of peers. In Example 4.1 a pair mapping would be any mapping defined between any two universities (e.g. (UW, UT), (UBC, UW), ..., etc). These schema mappings would be stored at the site interested in accessing the other peer's data. Relying on the transitivity of the mappings, UBC would be able to access UT's data only if UBC has a pair mapping definition for UT's data or if UBC has a pair mapping to UW, and UW has a pair mapping to UT (Figure 11a). The Local Relational Model (LRM) follows this approach to schema integration [Bernstein et al. 2002]. In the LRM proposal, a peer specifies translation rules and coordination formulas that define how its data schema relates to a data schema on another peer, referred to as an *acquaintance*. A semantic network is then defined by the acquaintance links between peers.

Peer-mediated mappings: In peer-mediated mappings, a peer can define a schema mapping that relates two or more peers. Hence, peer-mediated mappings are a generalization of pair mappings. In Example 4.1, UW could provide a mapping that includes data from UW, UT and UBC. A fourth university, SFU would be able to access UW, UT, and UBC's data by accessing UW's peer mediated schema (Figure 11b). Piazza [Tatarinov et al. 2003] and PeerDB [Ng et al. 2003] are systems that follow this approach for integration. As it will be discussed later, PeerDB can create the mediated mappings at query time.

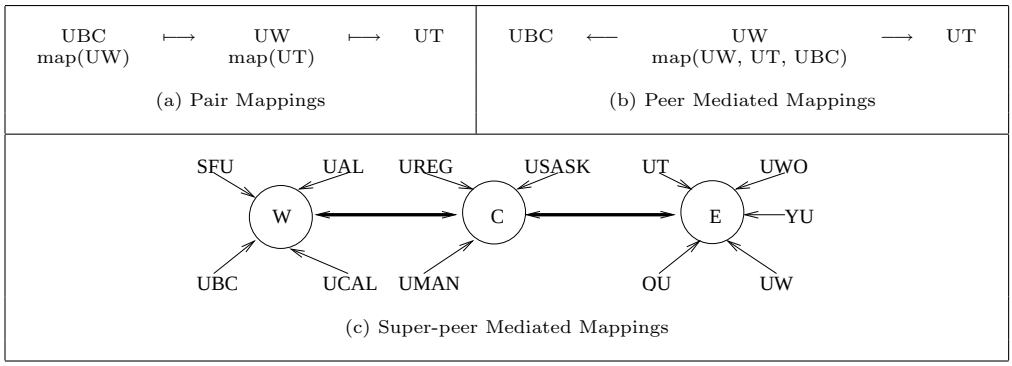


Fig. 11. P2P Schema Mapping Types

Super-peer mediated mappings: In super-peer P2P systems, mediated schemas are defined at the super-peer level. *super-peer to peer* mediated schemas contain the mappings for all the regular peers associated to the super-peer. Schemas between super-peers, called *super-peer to super-peer* mediated schemas can also be defined to implement data sharing between peers associated to different super-peers. In Example 4.1, a super-peer W (West) may provide the mediated mapping for the data at SFU, UAL, UBC, and UCAL. A super-peer C (Central), may provide the mediated mapping for UReg, USask, and UMan. Finally, a super-peer E (East) may provide the mediated mapping for UW, UT, UWO, QU and YU (Figure 11c). When UW queries for data in UT, the mediated schema in E is used. If UW queries for data in UBC, a mediated schema between super-peers E and W will be used as well as the super-peer to peer mediated schema at W. Edutella [Nejdl et al. 2002a; Löser et al. 2003] is a system that implements super-peer mediated mappings in a HyperCuP super-peer topology . Independently of the approach used to implement the schema mappings, P2P systems attempt to exploit the transitive relationships among peer schemas to perform data sharing and integration [Halevy et al. 2004]. While in traditional distributed systems, schema mappings form a semantic tree, in P2P systems the mappings form a *semantic graph*.

4.1.3 *Creation and Maintenance of Schema Mappings.* Techniques used by or applicable to P2P systems to create and maintain schema mappings can be classified into four groups: machine learning techniques, common-agreement, semantic gossiping, and information retrieval (IR) approaches.

Machine Learning Techniques are typically used when the peers represent data based on ontologies and taxonomies as proposed in the Semantic Web [W3C 2001]. GLUE [Doan et al. 2003] is a system that applies machine learning techniques to semi-automatically find semantic mappings. Given a concept in one taxonomy, the focus of GLUE is to find the most similar concept in another taxonomy. The basic idea is to provide classifiers for the concepts. To decide the similarity between concepts `A` and `B`, data for concept `B` is classified using `A`'s classifier and vice versa. The amount of values that can be successfully classified into `A` and `B` represent the similarity between `A` and `B`. The system is capable of identifying an attribute "name" in one taxonomy, as related to the concatenation of attributes "first_name"

and "last_name" in another taxonomy. The developers of GLUE assume a large user base that collaborate to define the semantic mappings. GLUE helps in the process by asking precise questions that leverage the answers to improve accuracy [Doan 2005]. In Piazza [Tatarinov et al. 2003], a machine learning technique called *stacking* [Wolpert 1992], which attempts to reduce errors made in generalization processes, is used to combine different approaches to exploit diverse evidence of semantic similarity. Knowledge acquired in previous semantic matches is used to build a collection of metadata information. Statistics are collected over schema elements in the collection and are employed in future schema matchings.

A *Common-agreement* mapping, is a schema mapping done between peers with a common interest for sharing data. The mapping is generally done and maintained in a manual (non-automated) fashion by privileged or expert users. For example, the *APPA* P2P system makes the assumption that peers wishing to share data, are likely to agree on a Common Schema Description (CSD) [Valduriez and Pacitti 2004]. Peer schemas are specified as views on the CSD and queries are expressed in the terms of the local schema, not the CSD. APPA assumes that the schema mappings are maintained until the data sharing is no longer desired. In AutoMed [McBrien and Poulovassilis 2004] the definition of common-agreement mappings is done by using primitive bidirectional transformations defined in terms of a low level data model. The Local Relational Model [Bernstein et al. 2002] proposes coordination formulas that specify how the data in one peer relates to data in other peers. These coordination formulas are manually modified overtime as data sharing requirements between the peers evolve. Piazza [Tatarinov et al. 2003] is another system that assumes that peers interested in sharing data are willing to define and maintain the mappings between their schemas.

Semantic-gossiping is proposed in [Aberer et al. 2003] and assumes the existence of common-agreement mappings between peers, initially constructed by skilled experts. Relying on the transitivity of these mappings, queries are propagated towards nodes for which no direct mapping exist. Query propagation is to determine the semantic agreement with these semantically unmapped peers. The semantic agreement is measured by analyzing the results of the queries sent to the peers. Based on the degree of semantic agreement, new mappings may be created or existing mappings may be adjusted. The final goal is to incrementally derive an implicit global mapping among peers.

IR approaches are used in PeerDB [Ng et al. 2003] to decide, at query execution time, the mappings between two peer schemas. In PeerDB, descriptive words are associated to schema attributes. Queries are flooded to neighbouring peers which, by using IR matching techniques, decide if matching attributes for the query exist in the local schema. The user that submitted the query is then presented with these attributes and must confirm whether or not to proceed with the execution of the query at the remote peer where schema attributes match. The decision of the user is registered by the system and is used to route future queries to matching peers, when the queries refer to the same attributes as the initial query.

4.2 Data Mappings

Schema mappings work well when the schemas being integrated are semantically very close. In other words, when the differences between the schemas are mainly

structural: attribute *values* represent the same information, or can be transformed to be the same, and relations can be redefined using views to be structurally similar to relations or views in another (possibly mediated) schema. When the attribute values differ, but are nevertheless semantically related, data mappings are sometimes the only option for data sharing.

Data mappings are implemented as relations on the attributes being mapped. The mapping relations are called *mapping tables* and represent expert knowledge [Kementsietsidis et al. 2003b]. Tuples in the mapping tables specify the correspondence between values in the relations for which the data mapping is being defined. Actual tuples in the mapping tables are typically specified by domain specialists. For example, consider a mapping table relating gene data in one database to protein data in another database [Kementsietsidis et al. 2003a].

Assuming a data mapping between relations `A` and `B`, values of `A` not showing in the mapping table can be interpreted as not related to any value in `B` (*close world semantics*), or being related to every value in `B` (*open world semantics*). Interestingly, open world semantics are equivalent to closed world semantics when variables are allowed as values in tuples in the mapping tables as shown in [Kementsietsidis et al. 2003b].

Data mappings, like schema mappings, are transitive and sometimes one-to-one and reflexive. Data mappings define a semantic graph among peers as well.

4.3 Current Research in P2P Data Integration

Most P2P systems assume the existence of semantic mappings among peers, but do not specify how these mappings are created. The creation of the semantic mappings is probably the hardest aspect of the data sharing process. It is an expensive process that in most cases requires human intervention. Research related to the *Semantic Web* seems to have the most potential in this field [W3C 2001; Doan et al. 2003]. This is nevertheless a very hard problem that is aggravated by the need for collaboration among researchers from the AI and database fields.

Once the semantic mappings (either schema or data mappings) are created, several research issues arise in P2P systems. For example, assuming semantic mappings between UBC and UW in Example 4.1, and between UW and UT, it would be desirable to determine if there is information loss between UBC and UT. If there is information loss, identifying weak mappings that need to be improved would be desirable as well. Formally, given peers `A`, `B`, and `C`, and mappings `A` to `B`, and `B` to `C`, the goal is to compute a *mapping composition* between `A` and `C`. A mapping composition is a direct mapping between `A` and `C` equivalent to the original mappings [Madhavan and Halevy 2003]. When mapping compositions are expressed as formulas, another interesting problem is how to identify the *minimal composition* (*mapping cover* is the term used in [Kementsietsidis et al. 2003b] for data mappings), or the minimal formula sufficient to produce a mapping composition. Languages and formalisms to represent and manipulate mappings are presented in [Madhavan and Halevy 2003; Halevy et al. 2004; McBrien and Poulovassilis 2004].

Finally, another active area of research in P2P data sharing is the use of rule mechanisms (triggers) to enforce integration rules and to propagate data among peers. These rules allow to coordinate actions among peers when data are changed, inserted, deleted, or retrieved [Kantere et al. 2003; Arenas et al. 2003]. The semantic

differences between the schemas need to be considered when the data are propagated or the rule mechanisms are specified.

4.4 Summary

Schema and data mappings are the data integration methods commonly used in P2P systems to share structured data among peers. Schema mappings define structural and basic semantic relationships between schemas. Data mappings complement schema mappings and are used to define associations among data values in two different data sources. Both schema and data mappings define a semantic overlay network, where the transitivity of the semantic mappings can be exploited to share data with peers for which no direct mappings exist.

5. QUERY PROCESSING

In this section we look at query processing in systems sharing files, followed by systems sharing structured data. In file sharing systems, queries are routed to peers with the goal of locating files. Systems sharing structured data allow more expressive queries. Query processing in this case is done by peer selection, query routing, data retrieval, and merging retrieved results. We focus on the P2P-specific aspects of query processing, and do not examine the more traditional aspects of querying, such as query specification models/languages or user interfaces. We begin by discussing file sharing systems.

5.1 Query Processing in File Sharing P2P Systems

Many P2P systems aim to transfer files between peers. While many early P2P file-transfer systems such as Napster and eDonkey use a centralized indexing server, second generation systems use a much more distributed approach. Recaping Section 2.1, these file-sharing systems have different characteristics and requirements compared to other P2P systems. The purpose of query routing in these distributed systems is to locate peers that store the requested file. In contrast to DHTs, files or data items (such as keys) in file-transfer systems generally stay where they are and are not redistributed uniformly over the system. Furthermore, most frequently requested files are usually replicated at many peers. The topology of these systems are often highly dynamic and unstable. To conserve bandwidth and maintain scalability, propagation of routing tables is bounded by a fixed number of hops. As such, schemes used in large-scale stable environments that require significant maintenance overhead when peers join and leave are not appropriate in these systems. Queries are often routed to multiple peers in the overlay network according to some routing scheme. These schemes can be generalized into two main categories: blind searches and informed searches. Blind searches arbitrarily route queries to particular peers without any knowledge whether the query will be satisfied by these peers. These schemes are simple and peers do not need to store metadata related to other peers. Alternatively, informed searches route queries to peers that have a known higher probability of satisfying the queries. Because of the additional knowledge needed to route the queries, these peers store a considerable amount of metadata about other peers.

5.1.1 *Blind Searches.* These schemes do not store any information regarding data placement. Routing is done in a random fashion. Different optimizations can be made to achieve implementation simplicity, reduced traffic overhead, reduced resource utilization, reduced query latency and scalability.

Flooding is the simplest way to route queries. To propagate a query to a large number of peers, a node sends the query to all its neighbours. To limit the computational and network resources used, a node typically does not flood the neighbours again with the same query. A time-to-live (TTL) mechanism is also used to limit the radius of the flood. Systems such as Gnutella only uses flooding. This scheme has the advantage of propagating the query to a large number of peers quickly. However, this scheme is limited by the search radius and can overload the network quickly.

To optimize the flooding scheme, a two-level hierarchical approach based on peer heterogeneity was proposed. FastTrack extends simple flooding and introduces supernodes. Compared to regular peers, supernodes are powerful nodes that can handle more load. One or more supernodes act as proxies for the regular peers; flooding is performed only among supernodes. KaZaA implements the FastTrack protocol. The advantages are that flooding is reduced and less capable peers are not overwhelmed by messages. Although this scheme is more scalable than simple flooding, it does not scale well as the number of supernodes grows larger.

To further reduce network traffic in favour of scalability, a random walk technique can be used. By using a walker, a query is forwarded to exactly one neighbour at a time [Lv et al. 2002]. Propagation is bounded by TTL or a counter that gets decremented when a match is found. This reduces the traffic generated compared to flooding and FastTrack. However, search latency may be increased because the routes taken are longer. Because popular files are highly replicated in a P2P file-sharing systems, it is highly likely that these popular files can be found within a small number of hops [Sakaryan et al. 2004]. Biased random walk is a variation that takes peer heterogeneity into consideration. Instead of randomly choosing a peer, the GIA system [Chawathe et al. 2003] chooses a peer that is most capable in terms of network bandwidth. Along with one-hop replication, this creates a hierarchy where queries are propagated among highly capable peers and more capable peers act as proxies to less capable peers.

5.1.2 *Informed searches.* These schemes collect a large amount of information about data placement. More effective routing is done to reduce the number of hops required to locate a data item. The objective is to increase the probability of locating a data item as the hop count in the route increases. The schemes differ in the metadata collected about the peers and data placements.

In order to move from blind searching to informed searching, some form of routing table containing data placement information must be maintained at each peer. Query Routing Protocol (QRP) aims to replace blind flooding in Gnutella with informed routing [Rohrs 2002]. To achieve this, keywords describing the files that a peer offers are hashed and encoded in a Bloom Filter. These Bloom Filters make up the routing tables and are exchanged with the neighbours. The peer that has received the routing table merges it with its own and keeps propagating it. To conserve bandwidth and maintain scalability, propagation of routing tables are

bounded by a fixed number of hops. Queries are forwarded to the neighbour only if its routing table contains all the keywords. If none of the neighbours know about the requested file, this scheme degenerates to a blind search.

Building onto the Bloom Filter idea, the Exponentially Decaying Bloom Filter (EDBF) utilized by the Scalable Query Routing (SQR) scheme is used to construct a probabilistic routing table at each peer [Kumar et al. 2005]. The EDBF has the property that the amount of information encoded in the Bloom Filter about the location of a file decreases exponentially as the hop count increases. Query routing begins as a random walk because the EDBF encodes too little information about the peer hosting data at a far distance. However, as the query approaches the neighbourhood that is aware of the peer hosting the data, the search sharply hones into the peer.

Although following the same basic idea that queries are forwarded from peer to peer, Compound Routing Indices uses a different neighbour selection criteria [Crespo and Garcia-Molina 2002]. Files are categorized into topics and the locations of files are stored in terms of these topics. Neighbours are selected by the most files reachable by forwarding the query to that neighbour. The Hop-count Routing Indices incorporates hop-count information into the routing indices. The cost function for the neighbour selection criteria takes both number of files reachable and number of hop count into consideration. Lastly, the Exponentially Aggregated RI stores the result of applying the regular-tree cost formula to the hop-count RI. This is done to combat the fact that the hop-count RI does not contain information beyond a certain hop-count threshold.

The Adaptive Probabilistic Search (APS) uses a k-walker probabilistic approach to route queries [Tsoumakos and Roussopoulos 2003]. Instead of one random walker used in GIA, APS uses k simultaneous walkers each with its own probabilistic forwarding algorithm. Probability of forwarding to a neighbour is influenced by previous search results. Each peer maintains an index of neighbours for each file request seen. A query is forwarded to the neighbour that has the highest index value for the requested file. This index is updated as follows. In the optimistic approach, when a forwarding neighbour is selected, its index value is incremented. If the query is satisfied at some peer, the query terminates and the indices at the peers are left untouched. However, if the query cannot be satisfied before the TTL expires, a query-failed message propagates backwards and all peers in the route decrement their corresponding indices. The pessimistic approach leaves the index intact when a forwarding peer is selected and only increment the index when the query is known to have been satisfied at some peer. This scheme is self-learning and performs better when the peers stay in the system longer. However, this scheme can experience hot spots as one path gains popularity quickly, which leads to under utilization of other possible paths.

FreeNet [Clarke et al. 2002] uses the steepest-ascent hill-climbing search to locate a file. This is similar to the biased random walk. The decision of which peer to forward the query to depends on the routing table located at each peer. The routing table is constructed to indicate which neighbour the peer thinks has a greater chance of locating the requested file, which is represented by its globally unique identifier (GUID). When the file is located, it is passed backwards via the

query route and peers along the way update their routing table. Because FreeNet concerns itself with privacy, peers along the way may replicate the file and declare that the file is located at itself. The longer a peer stays in the system, the more information it gains about the potential location of files. Unlike other routing schemes mentioned in this paper, FreeNet requires that the file name is known in advance. The requested filename is converted into the GUID via the SHA-1 hash function before being forwarded to other peers.

5.1.3 *Conclusion.* We have presented the random search and informed search schemes in this subsection. These schemes target large scale, dynamic, file sharing P2P systems. Information about neighbours can be stored in variety of ways, such as Bloom Filters or Routing Indices. Both types of searches aim to locate files or data items, which are never relocated to other peers. Keyword-search queries are routed to peers in a random fashion or by some heuristics based on metadata collected.

5.2 Query Processing in Data Sharing P2P Systems

Traditionally, query processing for distributed data sharing environments is implemented by using a mediation layer that decomposes queries based on a global schema and sends appropriate sub-queries to dispersed sources, or by integrating and materializing data from diverse sources at a central place in a warehouse type setting. Such approaches are successful because of the availability of a global schema and the possibility to implement central control. However, in settings like large-scale P2P environments, where frequent changes in schemas or in source availability are very common, a central component that provides the schema knowledge as well as coordinates the execution of queries may become a bottleneck or a single point of failure. Therefore, decentralized query processing techniques are more suitable for P2P systems.

The suitability of decentralized query processing techniques has been demonstrated by the success of early file sharing systems like Napster and Gnutella, and more recently by DHTs. Decentralized P2P query processing implies formulating and executing queries at different peers with probably different characteristics and data models. Moreover, due to the rapid evolution of P2P applications, support for richer and more complex queries is highly desirable. Current trends seem to indicate that range queries, multi-attribute queries, join queries and aggregation queries will be integrated in most future P2P applications. This section presents previous and current research efforts and issues in query processing for P2P data sharing systems.

5.2.1 *Query Processing in Common Data Model P2P Systems.* Some P2P systems assume the existence of common schemas or data models that apply to all peers. Queries are distributed among peers based on query type and specifications like required attributes, predicates and data filters; which could be used to select peers to contact. Processing and filtering of results is done incrementally by different peers as queries are distributed. This is different from earlier P2P efforts where combination of results is normally done at the client side.

Under the Edutella framework [Edutella 2005; Löser et al. 2004; Nejdl et al. 2002b], each peer has a set of wrappers that translate the query results according to

specifications provided in a common query exchange language. A query is normally split into several subqueries that are transferred to different peers by consulting schema aware indices located at super peers. Peers process the subqueries and return combined results to the peer that initiated the query. What distinguishes this work is the proposed query shipping functionality, which pushes query descriptions and possibly user defined-code to peers participating in the query execution, and combines results on the remote peers.

Query plans can potentially be modified while moving through the network. In this setting, each peer resolves some portions of the plan it receives, and then forwards it to other peers. [Papadimos et al. 2003] introduce the notion of mutant query plans, where query plans are injected into the network by one of the peers based on a global predefined XML schema. Any peer can choose to mutate an incoming query plan in two ways: It can resolve enclosed resource names to their location(s), or it can supply the data of some resource directly, if it knows how to get it. Peers can also reduce the plan by evaluating a sub-graph of the plan and substituting the results in place of the sub-plan. If the plan is completely evaluated, it is sent back to the initiator, otherwise it is routed to another peer able to continue processing.

Query routing indices have also been widely used in common data model P2P systems. For instance, in [Galanis et al. 2003], each peer has a local XML search engine that enables document searches based on both, structure and content. Furthermore, each node maintains additional indices that facilitate the forwarding of queries to data providers that have relevant data. Each search engine indexes its local data using inverted lists, which map keywords to documents. These indices enable the processing of containment queries. In addition, each node maintains a peer index that maps keywords to peers. Queries are executed on local indices as well as peer indices stored at the querying peer. Using the peer index, a query can be sent directly to nodes that are likely to have results.

Some other systems like [Huebsch et al. 2003] assume the existence of a common data model as an inherent property for large distributed environments, where the data generated by popular software products follow the same schema allowing data sharing among peers. Data is assumed to be horizontally fragmented among peers. In this system, queries are distributed based on a DHT routing layer.

5.2.2 *Query Processing in Schema Heterogeneous P2P Systems.* As discussed in section 4, when no common data model or predefined schema(s) exist, peers have to apply data integration techniques that provide a common ground for query processing. The basic idea is to identify content or structure similarities among peers. *Semantic mappings* are then defined to specify these similarities, and based on the semantic mapping definitions, queries are reformulated for each specific peer.

Bernstein et al. [Bernstein et al. 2002] propose the Local Relational Model, where a Query Manager component located at each peer, uses data translation rules and semantic dependencies to reformulate queries submitted to the peer to match the schemas of other peers. In the Piazza system [Halevy et al. 2003], storage mappings are used to associate queries with suitable data relations, while description mappings are used to associate query results at one peer to results at other peers. Based on these description mappings, a reformulation algorithm

produces query expressions equivalent to a given query; where the given query could be answered by combining multiple peers mappings to reach the relevant data. In the Chatty Web system [Aberer et al. 2003], the authors propose appending schema mapping quality measures to a query, and updating these measures as the query roams around the network. Schema mapping quality is measured by syntactic and semantic similarity. Syntactic similarity refers to the extent of information lost from queries when attributes from one schema do not exist in another schema. Semantic similarity refers to the level of agreement on the meaning between schemas, and is measured by looking at the transformations a query suffers when expressed in terms of other schemas. What distinguishes this work is the lack of reliance on the existence of shared schemas among integrated peers. In the PeerDB system [Ng et al. 2003], queries are executed in two steps: in the first step, peers are selected based on the amount of metadata intersection between the query terms and the peer schemas; in the second step queries are submitted to the selected peers and results are sent back to query initiator. Users in this system are involved in selecting the potentially promising peers from the set of peer selected in the first step in the query execution.

5.2.3 *More Complex Query Types.* The support for a wider range of P2P applications motivate the evolution of current P2P technologies. In terms of query management, complex queries need to be supported by P2P architectures. A clear drawback in existing traditional DHT architectures is the fact that only single key queries and exact matches are supported. Special architectures have been proposed to support more complex P2P queries such as range queries, multi-attribute queries, join queries and aggregate queries. Some of these architectures are based on DHT-like methods, while others are based on unstructured network layouts

(1) **Range Queries**
The support for range queries has been seen as an important open research problem in P2P networks. Some of the efforts have been focused on trying to adapt existing DHT architectures to support range queries. The major problem these efforts face is that the randomizing hash function, which is used by DHT architectures to create a well distributed balance of load on different peers, works against range queries. This is because there is no relation between the values in a range after the hash function has been applied on the values. One possible solution is to hash ranges, but it requires a priori dividing the total range into partitions assigned to different peers. In addition, the size of the partitions can be a problem. If the partition size is too large, peers can get easily overloaded. On the other hand, if it is too small, many hops will be needed to locate the data.
a. DHT Based Solutions. Some techniques propose introducing variations on the way DHTs work in order to support range queries. These techniques are based on the basic DHT interface discussed in Section 3, however, they extend insertion and lookup operations to support ranges. [Gupta et al. 2003] propose an architecture for relational data sharing among P2P system based on Chord, and a hash-based method to provide approximate answers to range queries. Given a range, this approach considers the set of values that the range consists of, computes an integer for each one of these values by applying a min-

wise independent permutation hash function, and finally takes the minimum of the resulting integers as the identifier for that range. Lookup is performed in O(log(n)) hops, where N is the number of peers. However, the simulation results showed load balance problem for large networks.

Sahin et al. [Sahin et al. 2002] extend the CAN system for d=2. A virtual hash space is constructed for each attribute as a 2-dimensional square bounded by the lower and higher values of the attribute's domain. The space is further partitioned into zones, and each zone is assigned to a peer. That peer stores the results of the range queries whose ranges hash into the zone it owns, as well as routing information about adjacent zones. A query is routed towards the range's zone through the virtual space. Once the query reaches that zone, the stored results at this zone are checked. If the results are found locally, they are returned. Otherwise, the query is forwarded to the left and top neighbours that may contain potential results.

Another CAN extension is proposed in [Andrzejak and Xu 2002]; where nearby ranges map to nearby CAN zones. A subset of servers, called interval keepers, is responsible for subintervals of the attribute's domain. Each node reports its current attribute value to the appropriate interval keeper. When a range query is issued, it is routed to interval keepers by propagating in two waves: the first one to the neighbours that intersect the query and have higher interval than the current node, and the second wave to those that have lower interval.

b. Non DHT Based Solutions. Since DHTs are more suitable for exact match queries, adapting DHTs to support range queries often creates load balance problems that must be addressed. Therefore, many research efforts are proposed to implement range queries using other type of P2P indices.

In [Aspnes and Shah 2003], [Harvey et al. 2003], a solution is proposed based on skip graphs. However, some load balance problems were found to be inherent to these type of graphs. The load balance problem is considered to be a consequence of skewed insertions, deletions or query loads. Some solutions are proposed to the load balance problems. For instance, Gansen et al. [Gansen et al. 2004] propose using two skip graphs one to index data and the other to track load on nodes. Each node is able to determine the load on its neighbours in addition to the minimum and maximum load in the system. They devised an algorithm to balance load on neighbours so that empty nodes can take over heavy loaded nodes.

Bharambe et al. [Bharambe et al. 2004] introduce the Mercury system to support multi-attribute range queries with an add-on scheme for load balancing. In this framework, nodes are grouped into virtual hubs each of which is responsible for some query attribute. Nodes inside hubs are arranged in rings in a way similar to Chord. Random sampling is used to estimate the average load on nodes and find light loaded parts.

(2) **Multi-Attribute Queries**

There has been some work on multi-attribute P2P queries. Cai et al. [Cai et al. 2003] introduce the Multi-Attribute Addressable Network (MAAN). This system is built on Chord to provide multi-attribute and range queries. They use a locality preserving hash function to map attribute values to the Chord

identifier space, which is designed with the assumption that the data distribution could be known beforehand. Multi-attribute range queries are supported based on single-attribute resolution. However, the authors notice that there is a query selectivity breakpoint at which flooding becomes better than their scheme. The authors followed up with the RDFPeers system [Cai and Frank 2004] which allows heterogeneity in peers schemas where each peer contains RDF based data items described as triples <subject, predicate, object>. The load balance improves in this work but no test for skewed query loads was made.

(3) **Join Queries**

The need to support join queries for large scale P2P environments pushed researchers to adapt the older distributed join algorithms in P2P settings. Distributed data among peers could be seen, in some cases, as a set of large relational tables fragmented horizontally. Running efficient join queries over such massively dispersed fragments is a challenging task.

Huebsch, Hellerstein et al. [Huebsch et al. 2003] describe a three layer architecture in their PIER system: storage, DHT and query processing. They implement two equi-join algorithms. In their design, a key is constructed from a "namespace" (relation) and a "resourceID" (primary key by default). Queries are multicasted to all nodes in the namespaces to be joined. The first join algorithm is a version of symmetric hash join, where each node in the two namespaces finds the relevant tuples and hashes the tuples into a new namespace. In the second algorithm, called "Fetch Matches", one of the two relations is already hashed on the join attributes. Each node in the second namespace finds the matching tuples from the first namespace using DHT get operations.

[Triantafillou and Pitoura 2003] considered multicasting to a large number of peers inefficient. They propose using a set of dedicated peers called range guards to monitor partitions of join attributes. Join queries are therefore sent only to range guards which decide the peers that should be contacted to execute the query.

(4) **Aggregation/Ranking Queries**

Increasing the number of peers in a peer-to-peer network usually increases the number of answers to a given query as well. Users are not usually interested in arbitrarily large and un-ordered answer sets, but rather in a small set of "best" answers. Performing result aggregation and ranking according to some user defined scoring function is now being considered in P2P environments.

[Balke et al. 2005] propose an algorithm for distributed processing of top k queries with minimum possible data traffic relying on super peer HyperCuP topology. The algorithm first selects participating peers and super peers by using the super peer index where the query was initiated. Peers evaluate top k queries locally according to the design of their data layer and then they deliver objects and their scores to their super peers, which in turn select from the incoming objects the ones with maximum score. This framework assumed a global shared schema as well as consistent ranking methods employed at peers.

In PlanetP system [Cuenca-Acuna et al. 2003] a globally content-ranked view of the shared data is maintained. The approach is comprised of two major com-

ponents: a gossiping layer to support the replication of shared data structures across groups of peers, and a text-based search and rank algorithm. The gossiping layer enables peers to push any change that happens to a replicated data structure (e.g. peer index) to a randomly selected peer. A peer stops pushing the rumour if it has contacted n consecutive peers that have already heard the rumour. The content ranking scheme is based on the vector space model. For a given query, peers are ranked according to their relevance to queries. This is estimated as the number of documents they have carrying the query terms maintained by global index.

5.2.4 *Summary.* Providing robust query functionality for highly dynamic and heterogeneous P2P environments is challenging. Existing research efforts either assume the existence of a common P2P data model; or they build query processing techniques over a schema or data integration layer to work around peers heterogeneity. The more recent research efforts adapt more complex query types, e.g. range queries, to P2P settings. P2P query processing is evolving in the direction of migrating distributed database techniques, such as distributed query processing and optimization, to P2P data management systems.

6. DATA CONSISTENCY ISSUES

6.1 Overview

Data consistency issues arise in any scenario involving data duplication. Peer-to-peer data management systems are no exception. Maintaining data consistency is most prominent in two scenarios: caching and replication. Caching is often done solely for the purposes of improved performance. A cache implies that there is a single authoritative source for the document, such as the original web server serving a document in the case of a web cache. Replication is commonly used for improving availability and reliability of data. In this scenario, all replicas are regarded as equally authoritative and changes to different replicas must somehow be reconciled. Distributed file systems often makes use of replication.

This section outlines the data consistency issues which have arisen in P2P systems. First, classical issues are briefly discussed in order to give a framework for understanding the P2P specific challenges. Following that, solutions to P2P data consistency issues will be discussed. In our peer reference architecture, these issues are addressed by the cache manager and the update manager, see Figure 1.

6.2 Classical Issues

6.2.1 *Caching.* Keeping data consistent has long been a problem in many areas of study. For example, operating systems must address cache coherency when maintaining a memory cache of disk data. Similarly, distributed file systems such as AFS [Howard et al. 1988], CODA [Satyanarayanan et al. 1990], and NFS [Sandberg et al. 1985] take measures to ensure a certain level of data consistency. Web proxy caches pose problems with data consistency as well. A big problem in any caching system is: *how does the system ensure that the user is receiving the latest copy of the data?*

Variants of strong and weak consistency can be seen in all of these examples; choosing one over the other entails choosing between different tradeoffs, depending

on the application. A web proxy cache that *checks-every-time*, through HTTP If-modified-since requests, is ensuring strong consistency, as a user will never see a stale copy of the document. A less resource intensive approach would be for the web proxy cache to rely on an *expiry time* (time-to-live) attached to the page by the document source. This method curtails resource usage, but the user may receive an out-of-date copy of the document. In this case we are trading freshness for resource savings.

Another common approach to ensure consistency is *cache invalidation notification*, as done in AFS and CODA. In these systems, clients locally cache copies of documents and when the document becomes updated at the server, the server sends a callback message to the appropriate clients. A tradeoff is made: increased resource usage for eliminating staleness.

6.2.2 *Replica Management.* Consider a distributed file systems that aims to maintain consistency between replicas of documents. In order to ensure no update conflicts, a pessimistic approach could be taken, such as locking. In AFS and CODA, consistency is relaxed in favour of performance and lower overhead. These systems take a more optimistic approach and allow multiple clients to modify the same file simultaneously. This opens the possibility for updates to overlap each other. Should two updates conflict, some form of resolution needs to take place. If the updates are in different parts of the file, automatic and application-specific techniques can often be used to merge the changes. If the changes overlap within the file, the user must manually merge the changes, often through a conflict resolution interface provided by the system.

In systems like AFS and CODA, strict consistency is traded for increased simplicity and functionality.

6.3 New Data Consistency Issues in P2P

P2P systems pose new data consistency challenges for which classical solutions do not directly apply. We first discuss the characteristics of P2P systems which give rise to these challenges.

6.3.1 *P2P Characteristics.* Peer-to-Peer systems are fully decentralized which have the potential for many natural benefits over their centralized alternatives. [Iyer et al. 2002] point out some of these advantages:

—Organic scaling
—No costly infrastructure
—Self-organizing structure
—Resilient to node failures
—Low overhead on participating nodes

In contrast, centralized solutions tend to be expensive both in terms of infrastructure costs (dedicated hardware) and administrative costs. However, the benefits of decentralization do not come without a cost. In the context of data consistency, P2P solutions trade decreased centralization for heightened consistency challenges.

6.3.2 *P2P Specific Challenges.* With the benefits of a decentralized P2P approach come specific challenges that must be overcome in order to maintain data

Table III. P2P specific challenges and their implications on data consistency

Challenge	Implications
High churn rate: nodes frequently joining, leaving, and failing	Must have ways of maintaining the network structure, e.g. using a DHT
Lack of global knowledge	Must act on partial knowledge, such as probabilistic measures
Low online probability	Peers are offline most of the time and cannot be relied on to keep data intact
Unknown and varying node capacity	Can't assume well connected and powerful infrastructure; must be sensitive to individual capacity
Overlay topology is independent of physical topology	One hop in the overlay may be a large physical distance; must be aware of underlying topology

consistency. [Datta et al. 2003] and [Rowstron and Druschel 2001b] point of some of these challenges, which are summarized in Table III.

P2P systems must address classical issues such as update propagation and availability in the face of novel challenges. For example, maintaining a minimum number of replicas throughout the network in the face of frequent node failures, low peer online probability, and lack of global knowledge appears to be an insurmountable task. Typical distributed systems approaches simply do not account for these possibilities and often make assumptions to the contrary, such as high node availability and global knowledge. In a P2P system however, nodes being offline is the norm rather than the exception. A P2P replication system must therefore take into account these important characteristics of the participating peers.

P2P systems typically make use of an overlay network in order to impose an interconnection structure, e.g. a DHT. This abstracts away the underlying network characteristics which physically connect the peers. While convenient to use, the overlay may fail to exploit locality properties and optimizations that would be evident had the physical network been considered. Furthermore, P2P systems often aim for distribution of data in a uniform and random fashion. This fails to consider the capacities of individual nodes. A slow machine connected to a home network cannot handle nearly as much load as a high end server connected to an Internet backbone. Since data replication involves load distribution, two important characteristics of a P2P system are that it be both topology-aware and capacity-aware.

Having given a framework for the many aspects of the problem, we proceed to discuss possible solutions.

6.4 Solutions to Data Consistency Issues in P2P

6.4.1 *High Churn Rate.* The use of a DHT is an effective solution to the problem of maintaining lookup capabilities in the face of high churn rate. The underlying DHT maintenance algorithms will maintain the structure of the overlay network in the face of node joins and failures. This approach is taken by many systems, such as Oceanstore [Kubiatowicz et al. 2000], PAST [Rowstron and Druschel 2001b] Ivy [Muthitacharoen et al. 2002], CFS [Dabek et al. 2001] Squirrel [Iyer et al. 2002], Backslash [Stading et al. 2002], [Datta et al. 2003], [M. Waldvogel 2003]. It should be noted that while DHTs effectively deal with high churn rate, they only deal with it at the lookup level. That is, the DHT will guarantee that a key will map to some node in the face of nodes joining and failing. Ensuring the actual data survives is

Table IV. Data consistency challenges in P2P and their solutions

Challenge	Solutions in Replication	Solutions in Caching
High churn rate	Must maintain k online replicas: (1) Using estimated global information and probabilistic methods, (2) Store k replicas at k successors in a Chord ring, (3) Maintain replicas at nodes with k closest numericIds in the DHT	No need to maintain availability of cached data, use DHT to maintain lookup
Lack of global knowledge	(1) Use estimated global information based on rumor spreading, (2) Not an issue if using a DHT	Global knowledge would help optimize cache placement, but not necessary
Low online probability	Assume peers won't rejoin and maintain k replicas across the set of online peers; use pull techniques to update peers upon rejoining	Not a concern with caching, peers share/cache whatever is available
Unknown and varying node capacity	(1) Often ignored, (2) Nodes advertise their capacity, replica placement is based on these capacities	Cache and share only what you choose
Overlay topology is independent of physical topology	(1) Often ignored, (2) Use a DHT with locality properties	(1) Often ignored, (2) Use a DHT with locality properties
Updating replicas / Reducing staleness	(1) Push updates to all active replias, (2) Pull updates from most recent replica when required (3) Store updates as node-specific log entries (4) Update replicas using DHT index	(1) Assume data doesn't go stale, (2) Web: Expiry (TTL), Conditional GETs (If-modified-since)

a separate problem, and is addressed below.

Unstructured P2P system can be used for data consistency applications, but are far less common. They are used in situations where less infrastructure is desired, such as sharing documents among nodes when moving data around to create strict structure is not desired. Peer-OLAP [Kalnis et al. 2002] uses an unstructured network to share cached results between peers engaging in On-Line Analytical Processing (OLAP) queries. In this system, a peer searching for a cached result is not guaranteed to locate all documents. Although it is possible to use unstructured networks, structured overlays like DHTs are a natural approach for data consistency based applications.

Recall that we are dealing with applications involving either of two primitives: caching and replication. We will discuss the challenges specific to each of these next. Table IV summarizes the information presented.

6.4.2 *Replication.* Keeping replicas consistent in a P2P system is a challenging task. One major challenge is maintaining *data continuity*. In the context of P2P systems, this reduces to the problem of maintaining a sufficient number of replicas in order to ensure that data is never lost.

[M. Waldvogel 2003] addresses this problem using a DHT which takes the pair (replica number, document id) as its key into the DHT. Therefore, in order to perform a lookup for a particular document, a particular replica number must be specified. For example, if a lookup for replica one fails, the user can perform a lookup for replica two. PAST [Rowstron and Druschel 2001b], a P2P persistent storage utility, performs the replication closer to the DHT layer. The k replicas are

placed at the nodes with nodeIDs closest to the docID in the underlying Pastry DHT. Other approaches, such as Ivy [Muthitacharoen et al. 2002], CFS [Dabek et al. 2001] and OceanStore [Kubiatowicz et al. 2000], rely on the underlying DHT to provide the necessary replication. In the case of Ivy and CFS, they rely on the DHash [Dabek et al. 2001] DHT, which places the k replicas on the k successors in the Chord ring of the node storing the document. If the primary replica fails, the successor immediately takes over.

Another major problem which must be addressed in replication systems is the issue of *updating replicas*. Some distributed file systems, such as CFS, provide only read-only access, which avoids this problem. Other systems, such as PAST, store immutable data, which implies that once data is added to the network, it cannot be modified. These systems sidestep the problem of updating replicas.

The method for updating replicas is highly dependant on the mechanism used to store replicas. In [M. Waldvogel 2003], updating the replicas is simply a matter of updating the all of the (replica number, document name) pairs. In OceanStore, updates are disseminated on the spanning tree built by the Tapestry DHT. Ivy stores all updates as per-node log entries. These logs are then stored as data in the DHT and if a node wishes to retrieve the most recent data, it must apply all updates in the log since its last snapshot. In the work done by [Datta et al. 2003], a push/pull technique is used to update replicas. When a node updates a document, it performs a push of that update to the other nodes storing the replicas. If a node comes back online and wishes to update its copy of the replica, it performs a pull whereby it contacts the other nodes storing the replica and retrieves the latest copy.

Node capacity is often ignored when deciding where to place replicas. One system which considers node capacity is PAST, where nodes advertise their capacity (e.g. disk space free), and the network performs some intelligent distribution process based on this information. If space is limited, the network will choose to replicate more popular files rather than less popular files in order to balance load.

6.4.3 *Caching.* Dealing with *churn* and *low online probability* in a P2P system is not particularly important for caching applications, as the goal is not to maintain availability of the cached data, but simply to use a cache to provide improved performance. If a node containing a specific document leaves the network, the overhead required to move its cache to a neighbouring node is likely greater than the effort required to retrieve the specific document from the original source, such as the original web server in the case of a web cache. Squirrel [Iyer et al. 2002] takes this approach, but allows the option of a leaving node to transfer some of its cache to two of its neighbours. Peer-OLAP and the work done by [Sahin et al. 2004] both make no attempt to maintain cached data of a departing node.

Having a *lack of global knowledge* in a caching application does not pose a significant problem as long as lookups for a particular data item can be performed. Again, since the goal is performance, determining the number of cached copies of the document throughout the network is not critical to the functioning of the system. Knowing such information, caching algorithms could perhaps be improved for load-balancing and efficiency purposes, however global knowledge to achieve this is not required. For example, Squirrel [Iyer et al. 2002], when using its directory scheme, maintains a list of nodes which contain a cached copy of a particular docu-

ment. A DHT lookup for the document resolves to the node (the home node) which stores this list of cached copies. This list is kept to a fixed length in order to limit the amount of redundant caches in the network. Global knowledge in Peer-OLAP would help the lookup procedure locate the desired document, but this cannot be reasonably achieved in an unstructured network. Therefore, as with other unstructured networks, a flooding algorithm is employed for search and results are often limited to local nodes in the overlay.

The major challenge in both P2P and non-P2P caching applications is *reducing the amount of stale data* returned to the requestor. In a traditional web proxy cache, expiry information attached to the page is used in conjunction with conditional If-modified-since HTTP gets, as discussed in section 6.2. P2P web caching strategies also employ these weak consistency semantics for reducing staleness. Squirrel [Iyer et al. 2002] uses this approach and when a stale copy of a document is detected, all stale copies are effectively removed, and any future requests are re-cached across k nodes (where k is the number of caches maintained per document). Other cooperative caching schemes use the same consistency approach, such as Hier-GD [Zhu 2003], which takes the idea of Squirrel one step further by sharing cached objects between organizations.

6.5 Summary

P2P systems provide a decentralized approach to solving classical problems in replication and caching. They offer many advantages over traditional centralized approaches: e.g. organic scaling, no costly infrastructure, and fault tolerance. However, these benefits do not come without a cost. P2P approaches must address the two big challenges of: maintaining data continuity in the face of nodes frequently joining and leaving, and keeping replicas consistent when updates to the data occur. While these problems have been addressed in the literature, more work must be done in order to take these ideas into the implementation phase.

7. CONCLUSIONS

In this survey, we have examined data management issues in the context of P2P data sharing systems. In particular, we have identified an abstract reference architecture for these systems. In our architecture a peer contains a query interface, a data management layer, and a P2P network sublayer. We have identified four main issues which must be addressed in these systems: locating data, query processing, data integration, and data consistency. Within each component we have classified the problems and solutions addressed by current work. We summarize the main classifications, open problems, and future directions below.

In file sharing systems, user satisfaction is of utmost importance. In these systems, there is often a tradeoff between user satisfaction and incentive to share. However, the nature of these systems often entails that they can easily be abused by selfish peers. The existence of these systems relies on contributions made by individual peers. Striking a balance between user satisfaction and contribution is difficult. We believe that future file sharing systems can only truly succeed when an incentive to share is built into the system. Other characteristics of unstructured file sharing systems that we believe will continue to be successful into the future are the use of supernodes, global file IDs, automatic neighbour selection, and anonymity.

Finding a model which encourages users to share is an open problem in this area. Another problem observed in current file sharing systems like KaZaa is the high pollution rate [Liang et al. 2004]. That is, individuals purposely name files different than what is actually contained in the file. An open problem in this area is to build trust and reputation elements into the system to reduce this problem.

Overlay networks have become an extremely hot area of research, particularly the study of DHTs. Although DHTs have many advantages such as their ability to organically scale, there remains many issues which still need to be adequately addressed. DHTs, as with any overlay network, have been criticized for their topology-oblivious view of the underlying network. Systems, such as SkipNet, take a step toward solving this problem. However, many of the attractive properties of DHTs come from their uniform distribution of data throughout the network. Balancing data locality with fast lookup times and robustness is an issue that remains to be addressed in future work.

Two outstanding issues have been preventing DHTs from being widely adopted: lack of user incentive, and limited API flexibility. In a structured overlay like a DHT, a participating node must manage keys and potentially data for other random users in the system. The overhead and buy-in required to do this at a single peer has left users wary. The other issue which needs to be addressed is building application flexibility into DHTs. Application development on top of DHTs has been limited due to lack of a rich interface. Extending the basically key-value lookup provided by DHTs remains as an open issue in the area.

Heterogeneity in P2P systems sharing structured data is another challenging issue that has received a lot of attention. As P2P technologies become more popular in different domains, they are required to deal with the inherent differences in the way peers represent their data. Schema and data mappings are the data integration methods commonly used in P2P systems to share structured data between peers. Current trends are pushing toward representing data in well-defined structures which allow for easier mapping of data between peers. However, creating semantic mappings for legacy data remains an area which requires more research attention.

In cases where semantic mappings exist, there still remains open research problems. One research direction that has been taken by P2P database systems is to compute mapping compositions between data sets. e.g. Suppose a mapping exists from peer A to peer B and B to C, we wish to compute a mapping from A to C for which there is no information loss.

P2P query processing has evolved from supporting only simple file search based on keywords to more sophisticated queries that operate on structured data. Integrating semantic and structure mapping techniques with distributed query processing in P2P systems is a hot research topic. Recent query processing research attempts to adapt complex query types, e.g. range queries, aggregate queries and multi-attribute queries, to P2P settings. Estimating the goodness of query results remains an open issue in P2P query processing systems. The assumption that P2P users are usually not interested in accurate results might not hold for more sophisticated data sharing applications.

P2P applications involving replication and caching must data consistency issues

in the face of P2P specific challenges. For example, maintaining a minimum number of replicas to ensure data continuity becomes a difficult problem as peers are frequently joining and leaving the network. Another challenge is propagating updates throughout the system in order to maintain replica and cache consistency. Research in P2P data consistency has been on developing working systems which address these issues. In particular, many systems aim to provide probabilistic guarantees rather than strict consistency. Implementing caching schemes and distributed file systems overtop of DHTs has become a major focus of recent research in this area. Much work needs to be done in order to take these ideas into the implementation phase.

We have provided a reference architecture for P2P data sharing systems and identified the major challenges faced in each component. P2P data sharing systems are a hot area of research and will continue to be until these challenges are adequately addressed.

ACKNOWLEDGMENTS

We would like to acknowledge Tamer Özsu for his guidance and input in preparing this work.

REFERENCES

Aberer, K., Cudr-Mauroux, P., and Hauswirth, M. 2003. The chatty web: Emergent semantics through gossiping. In *WWW '03: Proceedings of the twelfth international conference on World Wide Web*. ACM Press, 197–206.

Aberer, K., Cudre-Mauroux, P., and Hauswirth, M. 2003. Start making sense: The Chatty Web approach for global semantic agreements. *Journal of Web semantics 1(1), December 1,* 1.

Adar, E. and Huberman., B. A. 2000. Free riding on gnutella. Tech. rep., Xerox PARC. August.

Andersen, D. 2005. Overlay networks: Networking on top of the network. http://dps.uibk.ac.at/index.pl/publications?proxiedUrl=httpews.com

Andrzejak, A. and Xu, Z. 2002. Scalable, efficient range queries for grid information services. In *P2P '02: Proceedings of the Second International Conference on Peer-to-Peer Computing*. IEEE Computer Society, Washington, DC, USA, 33.

Arenas, M., Kantere, V., Kementsietsidis, A., Kiringa, I., Miller, R. J., and Mylopoulos, J. 2003. The hyperion project: From data integration to data coordination. *ACM SIGMOD Record 32,* 3.

Aspnes, J. and Shah, G. 2003. Skip graphs. In *SODA '03: Proceedings of the fourteenth annual ACM-SIAM symposium on Discrete algorithms*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 384–393.

Balke, W.-T., Nejdl, W., Siberski, W., and Thaden, U. 2005. Progressive distributed top-k retrieval in peer-to-peer networks. In *ICDE '05: Proceedings of the 21st International Conference on Data Engineering (ICDE'05)*. IEEE Computer Society, Washington, DC, USA, 174–185.

Bernstein, P., Giunchiglia, F., Kementsietsidis, A., Mylopoulos, J., Serafini, L., and Zaihrayeu, I. 2002. Data management for peer-to-peer computing: A vision. In *Workshop on the Web and Databases, WebDB*.

Bharambe, A. R., Agrawal, M., and Seshan, S. 2004. Mercury: supporting scalable multi-attribute range queries. *SIGCOMM Comput. Commun. Rev. 34,* 4, 353–366.

Bickson, D. and Malkhi, D. 2003. A study of privacy in file sharing networks. Tech. rep., Leibniz Research Center, the Hebrew University of Jerusalem, Israel.

Birman, K. 2000. Technology challenges for virtual overlay networks. In *Proceedings of the 2000 IEEE Workshop on Information Assurance and Security*. West Point, NY, USA, 200–206.

BitTorrent 2005. BitTorrent. http://www.bittorrent.com/.

Cai, M. and Frank, M. 2004. Rdfpeers: a scalable distributed rdf repository based on a structured peer-to-peer network. In *WWW '04: Proceedings of the 13th international conference on World Wide Web*. ACM Press, New York, NY, USA, 650–657.

Cai, M., Frank, M., Chen, J., and Szekely, P. 2003. Maan: A multi-attribute addressable network for grid information services. In *GRID '03: Proceedings of the Fourth International Workshop on Grid Computing*. IEEE Computer Society, Washington, DC, USA, 184.

Castro, M., Druschel, P., Kermarrec, A., Nandi, A., Rowstron, A., and Singh, A. 2003. Splitstream: High-bandwidth content distribution in cooperative environments. In *Proceedings of the 2nd International Workshop on Peer-to-Peer Systems (IPTPS '03)*. Berkeley, California, USA.

Chawathe, Y., Ratnasamy, S., Breslau, L., Lanham, N., and Shenker, S. 2003. Making gnutella-like p2p systems scalable. In *SIGCOMM '03: Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications*. ACM Press, New York, NY, USA, 407–418.

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

Clarke, I., Sandberg, O., Wiley, B., and Hong, T. W. 2001. Freenet: a distributed anonymous information storage and retrieval system. In *International workshop on Designing privacy enhancing technologies*. Springer-Verlag New York, Inc., 46–66.

Cohen, B. 2003. Incentives build robustness in bittorrent. In *Proceedings of the 1st Workshop on Economics of Peer-to-Peer Systems*. Berkeley, California, USA.

Cox, R., Muthitacharoen, A., and Morris, R. 2002. Serving dns using a peer-to-peer lookup service. In *IPTPS '01: Revised Papers from the First International Workshop on Peer-to-Peer Systems*. Springer-Verlag, London, UK, 155–165.

Crespo, A. and Garcia-Molina, H. 2002. Routing indices for peer-to-peer systems. In *ICDCS '02: Proceedings of the 22 nd International Conference on Distributed Computing Systems (ICDCS'02)*. IEEE Computer Society, Washington, DC, USA, 23.

Cuenca-Acuna, F. M., Peery, C., Martin, R. P., and Nguyen, T. D. 2003. Planetp: Using gossiping to build content addressable peer-to-peer information sharing communities. In *HPDC '03: Proceedings of the 12th IEEE International Symposium on High Performance Distributed Computing (HPDC'03)*. IEEE Computer Society, Washington, DC, USA, 236.

Dabek, F., Brunskill, E., Kaashoek, M., Karger, D., Morris, R., Stoica, I., and Balakrishnan, H. 2001. Building peer-to-peer systems with chord, a distributed lookup service. In *HOTOS '01: Proceedings of the Eighth Workshop on Hot Topics in Operating Systems*. IEEE Computer Society, Washington, DC, USA, 81.

Dabek, F., Kaashoek, M. F., Karger, D., Morris, R., and Stoica, I. 2001. Wide-area cooperative storage with cfs. In *SOSP '01: Proceedings of the eighteenth ACM symposium on Operating systems principles*. ACM Press, New York, NY, USA, 202–215.

Datta, A., Hauswirth, M., and Aberer, K. 2003. Updates in highly unreliable, replicated peer-to-peer systems. In *ICDCS '03: Proceedings of the 23rd International Conference on Distributed Computing Systems*. IEEE Computer Society, Washington, DC, USA, 76.

Dhamija, R., Feldman, M., and Garcia-Martinez, A. 2002. Peer-to-peer privacy and anonymity. http://www.sims.berkeley.edu/~mfeldman/research/p2p_privacy.ps.

Doan, A. 2005. Learning to live with semantic heterogeneity. In *Dagstuhl Seminar, "Machine Learning for the Semantic Web"*. http://www.smi.ucd.ie/Dagstuhl-MLSW/proceedings/.

Doan, A., Madhavan, J., Dhamankar, R., Domingos, P., and Halevy, A. 2003. Learning to match ontologies on the semantic web. *The VLDB Journal 12,* 4, 303–319.

edonkey2000 2005. eDonkey2000 - Overnet. http://www.edonkey2000.com/.

Edutella 2005. Edutella Project. http://edutella.jxta.org.

eXeem 2005. eXeem. http://www.exeem.com/.

Faloutsos, M., Faloutsos, P., and Faloutsos, C. 1999. On power-law relationships of the internet topology. In *SIGCOMM '99: Proceedings of the conference on Applications, technologies, architectures, and protocols for computer communication*. ACM Press, New York, NY, USA, 251–262.

FastTrack 2002. Fasttrack peer-to-peer technology company. http://www.fasttrack.nu/.

FasttrackSuperNode 2005. Peer-to-peer technologies and protocols. http://matrix.netsoc.tcd.ie/~neo/4ba2/p2p/#fasttrack.

Galanis, L., Wang, Y., Jeffery, S. R., and DeWitt, D. J. 2003. *Processing Queries in a Large Peer-to-Peer System*. Lecture Notes in Computer Science, vol. 2681. Springer-Verlag GmbH, 273–288.

Ganesan, P., Gummadi, K., and Garcia-Molina, H. 2004. Canon in g major: Designing dhts with hierarchical structure. In *ICDCS '04: Proceedings of the 24th International Conference on Distributed Computing Systems (ICDCS'04)*. IEEE Computer Society, Washington, DC, USA, 263–272.

Gansen, P., Bawa, M., and Garcia-Molina, H. 2004. Online Balancing of Range Partitioned Data with Applications to Peer-to-Peer Systems. In *Proceedings of the 30th International Conference on Very Large Databases VLDB*. 444–455.

Gnutella 2005. Gnutella. http://rfc-gnutella.sourceforge.net/.

Good, N. S. and Krekelberg, A. 2003. Usability and privacy: a study of kazaa p2p file-sharing. In *CHI '03: Proceedings of the conference on Human factors in computing systems*. ACM Press, 137–144.

Gummadi, K., Gummadi, R., Gribble, S., Ratnasamy, S., Shenker, S., and Stoica, I. 2003. The impact of dht routing geometry on resilience and proximity. In *SIGCOMM '03: Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications*. ACM Press.

Gummadi, K. P., Dunn, R. J., Saroiu, S., Gribble, S. D., Levy, H. M., and Zahorjan, J. 2003. Measurement, modeling, and analysis of a peer-to-peer file-sharing workload. In *SOSP '03: Proceedings of the nineteenth ACM symposium on Operating systems principles*. ACM Press, 314–329.

Gupta, A., Agrawal, D., and Abbadi, A. 2003. Approximate Range Selection Queries in Peer-to-Peer Systems. In *Proceedings of the First Biennial Conference on Innovative Data Systems Research (CIDR)*.

Halevy, A. Y., Ives, Z. G., Suciu, D., and Tatarinov, I. 2003. Schema Mediation in Peer Data Management Systems . In *Proceedings of the International Conference on Data Engineering, ICDE*. 505.

Halevy, A. Y., Ives, Z. G., Suciu, D., and Tatarivov, I. 2004. Schema mediation for large-scale semantic data sharing. *VLDB Journal*.

Harvey, N., Jones, M., Saroiu, S., Theimer, M., and Wolman, A. 2003. Skipnet: A scalable overlay network with practical locality properties. In *Proceedings of 4th USENIX Symp. on Internet Tech. and Syst. (USITS), 2003.* ACM Press.

Harvey, N. J. A., Jones, M. B., Theimer, M., and Wolman, A. 2003. *Efficient Recovery from Organizational Disconnects in SkipNet*. Lecture Notes in Computer Science, vol. 2735. Springer-Verlag GmbH, 183–196.

Heckmann, O. and Bock, A. 2002. The eDonkey 2000 Protocol. Tech. Rep. KOM-TR-08-2002, Multimedia Communications Lab, Darmstadt University of Technology. December.

Heckmann, O., Liebau, N., Darlagiannis, V., Bock, A., Mauthe, A., and Steinmetz, R. 2005. *From Integrated Publication and Information Systems to Information and Knowledge Environments: Essays Dedicated to Erich J. Neuhold on the Occasion of His 65th Birthday*. Lecture Notes in Computer Science, vol. 3379. Springer-Verlag GmbH, Chapter A Peer-to-Peer Content Distribution Network, 69–78.

Hellerstein, J. M. 2003. Toward network data independence. *SIGMOD Rec. 32,* 3, 34–40.

Howard, J. H., Kazar, M. L., Menees, S. G., Nichols, D. A., Satyanarayanan, M., Sidebotham, R. N., and West, M. J. 1988. Scale and performance in a distributed file system. *ACM Trans. Comput. Syst. 6,* 1, 51–81.

Huebsch, R., Hellerstein, J., Lanham, B., S., L., and Stoica, I. 2003. Querying the Internet with PIER . In *Proceedings of the 29th Int. Conf. on Very Large Data Bases (VLDB)*.

Hughes, D., Coulson, G., and Walkerdine, J. 2005. Free riding on gnutella revisited: the bell tolls? Draft at http://www.comp.lancs.ac.uk/computing/users/hughesdr/papers/freeriding.pdf.

IRIS 2005. Project IRIS (Infrastructure for Resilient Internet Systems) Web-site . http://project-iris.net/.

Iyer, S., Rowstron, A., and Druschel, P. 2002. Squirrel: a decentralized peer-to-peer web cache. In *PODC '02: Proceedings of the twenty-first annual symposium on Principles of distributed computing*. ACM Press, 213–222.

Jain, S., Mahajan, R., and Wetherall, D. 2003. A study of the performance potential of dht-based overlays. In *Proceedings of the 4th Usenix Symposium on Internet Technologies (USITS), March 2003*.

Kaashoek, M. F. and Karger, D. R. 2003. *Koorde: A Simple Degree-Optimal Distributed Hash Table*. Lecture Notes in Computer Science, vol. 2735. Springer-Verlag GmbH, 98–107.

Kalnis, P., Ng, W. S., Ooi, B. C., Papadias, D., and Tan, K.-L. 2002. An adaptive peer-to-peer network for distributed caching of olap results. In *SIGMOD '02: Proceedings of the 2002 ACM SIGMOD international conference on Management of data*. ACM Press, New York, NY, USA, 25–36.

Kantere, V., Mylopoulos, J., and Kiringa, I. 2003. A distributed rule mechanism for multi-database systems. Springer-Verlag LNCS, 56–73.

Karp, B., Ratnasamy, S., Rhea, S., and Shenker, S. 2004. Spurring adoption of dhts with openhash, a public dht service. In *3rd International Peer To Peer Systems Workshop (IPTPS 2004)*. ACM Press.

KaZaAFileFormat 2003. KaZaA P2P FastTrack File Formats - 2003-10-12. http://home.hetnet.nl/~frejon55/.

KazaaMediaDesktop 2005. KaZaA media desktop. http://www.kazaa.com/.

Kementsietsidis, A., Arenas, M., and Miller, R. J. 2003a. Managing data mappings in the hyperion project. In *Proceedings of the International Conference on Data Engineering (ICDE)*. 732–734.

Kementsietsidis, A., Arenas, M., and Miller, R. J. 2003b. Mapping data in peer-to-peer systems: Semantics and algorithmic issues. In *SIGMOD '03: Proceedings of the 2003 ACM SIGMOD international conference on Management of data*. ACM Press, 325–336.

Kubiatowicz, J., Bindel, D., Chen, Y., Czerwinski, S., Eaton, P., Geels, D., Gummadi, R., Rhea, S., Weatherspoon, H., Wells, C., and Zhao, B. 2000. Oceanstore: an architecture for global-scale persistent storage. In *ASPLOS-IX: Proceedings of the ninth international conference on Architectural support for programming languages and operating systems*. ACM Press, New York, NY, USA, 190–201.

Kulbak, Y. and Bickson, D. 2005. The emule protocol specification. Tech. rep., Leibniz Research Center, the Hebrew University of Jerusalem, Israel.

Kumar, A., Xu, J. J., and Zegura, E. W. 2005. Efficient and scalable query routing for unstructured peer-to-peer networks. Unpublished Work. Available at http://www.cc.gatech.edu/ akumar/sqr.pdf.

Lenzerini, M. 2002. Data integration: A theoretical perspective. In *Proceedings of the 21st Symposium on Principles of Database Systems (PODS)*. 233–246.

Liang, J., Kumar, R., and Ross, K. 2004a. Understanding KaZaA. http://cis.poly.edu/~ross/papers/UnderstandingKaZaA.pdf.

Liang, J., Kumar, R., and Ross, K. W. 2004b. The KaZaA Overlay: A Measurement Study. In *Proceedings of the 19th IEEE Annual Computer Communications Workshop, 2004*. Bonita Springs, Florida.

Liang, J., Kumar, R., Xi, Y., and Ross, K. W. 2004. Pollution in P2P File Sharing Systems. http://cis.poly.edu/~ross/papers/pollution.pdf.

Löser, A., Naumann, F., Siberski, W., Nejdl, W., and Thaden, U. 2004. *Semantic Overlay Clusters within Super-Peer Networks*. Lecture Notes in Computer Science, vol. 2944. Springer-Verlag GmbH, 33–47.

Löser, A., Siberski, W., Wolpers, M., and Nejdl, W. 2003. Information integration in schema-based peer-to-peer networks. In *Proceedings of Conference of Advanced Information Systems Engieering (CAiSE)*. 258–272.

Lv, Q., Cao, P., Cohen, E., Li, K., and Shenker, S. 2002. Search and replication in unstructured peer-to-peer networks. In *ICS '02: Proceedings of the 16th international conference on Supercomputing*. ACM Press, New York, NY, USA, 84–95.

M. Waldvogel, P. Hurley, D. B. 2003. Dynamic replica management in distributed hash tables. Tech. Rep. RZ 3502, IBM Technical Report.

Madhavan, J. and Halevy, A. Y. 2003. Composing mappings among data sources. In *Proceedings of the 29th VLDB Conference*.

Malkhi, D., Noar, M., and Ratajczak, D. 2002. Viceroy: A scalable and dynamic emulation of the butterfly. In *Proceedings of Symposium on Principles of Distributed Computing (PODC 2002)*. ACM Press.

Maymounkov, P. and Mazieres, D. 2002. Kademlia: A peer-to-peer information system based on the xor metric. In *1st International Peer To Peer Systems Workshop (IPTPS 2002)*. ACM Press.

McBrien, P. and Poulovassilis, A. 2004. Defining peer-to-peer data integration using both as view rules. Springer Verlag LNCS, 91–107.

Muthitacharoen, A., Morris, R., Gil, T. M., and Chen, B. 2002. Ivy: a read/write peer-to-peer file system. *SIGOPS Oper. Syst. Rev. 36,* SI, 31–44.

My'zrak, A. T., Cheng, Y., Kumar, V., and Savage, S. 2003. Structured superpeers: Leveraging heterogeneity to provide constant-time lookup. In *WIAPP '03: Proceedings of the The Third IEEE Workshop on Internet Applications*. IEEE Computer Society, Washington, DC, USA, 104.

Napster 2001. Napster. http://www.napster.com/.

Nejdl, W., Wolf, B., Qu, C., Decker, S., Sintek, M., Naeve, A., Nilsson, M., Palmér, M., and Risch, T. 2002a. Edutella: a p2p networking infrastructure based on rdf. In *WWW '02: Proceedings of the eleventh international conference on World Wide Web*. ACM Press, New York, NY, USA, 604–615.

Nejdl, W., Wolf, B., Qu, C., Decker, S., Sintek, M., Naeve, A., Nilsson, M., Palmér, M., and Risch, T. 2002b. Edutella: a p2p networking infrastructure based on rdf. In *WWW '02: Proceedings of the 11th international conference on World Wide Web*. ACM Press, New York, NY, USA, 604–615.

Ng, W. S., Ooi, B. C., Tan, K.-L., and Zhou, A. 2003. Peerdb: A p2p-based system for distributed data sharing. In *Intl. Conf. on Data Engineering (ICDE)*.

Papadimos, V., Maier, D., and Tufte, K. 2003. Distributed Query Processing and Catalogs for Peer-to-Peer Systems. In *Proceedings of Conference on Innovative Data Systems Research (CIDR)*.

Plaxton, C. G., Rajaraman, R., and Richa, A. W. 1997. Accessing nearby copies of replicated objects in a distributed environment. In *SPAA '97: Proceedings of the ninth annual ACM symposium on Parallel algorithms and architectures*. ACM Press, New York, NY, USA, 311–320.

Pouwelse, J., Garbacki, P., Epema, D., and Sips, H. 2005. The bittorrent p2p file-sharing system: Measurements and analysis. In *Proceedings of the 4th International Workshop on Peer-to-Peer Systems (IPTPS'05), Feb 2005*.

Pugh, W. 1990. Skip lists: a probabilistic alternative to balanced trees. *Commun. ACM 33,* 6, 668–676.

Rahm, E. and Bernstein, P. A. 2001. A survey of approaches to automatic schema matching. *VLDB Journal 10,* 4, 334–350.

Ratnasamy, S., Francis, P., Handley, M., and Karp, R. 2001. A scalable content-addressable network. In *SIGCOMM '01: Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications*. ACM Press.

Ritter, J. 2001. Why gnutella can't scale, no, really. http://www.darkridge.com/~jpr5/doc/gnutella.html.

Rohrs, C. 2002. Query routing for the gnutella network. Unpublished Work. Available at http://rfc-gnutella.sourceforge.net/src/qrp.html.

Rowstron, A. and Druschel, P. 2001a. Pastry: Scalable, decentralized object location and routing for large-scale peer-to-peer systems. In *"Proceedings of the 18th IFIP/ACM Int'l. Conf. Distributed Systems Platforms (Middleware'01), LNCS 2218, 2001"*. ACM Press.

Rowstron, A. and Druschel, P. 2001b. Storage management and caching in past, a large-scale, persistent peer-to-peer storage utility. In *SOSP '01: Proceedings of the eighteenth ACM symposium on Operating systems principles*. ACM Press, New York, NY, USA, 188–201.

Sahin, O., Gupta, A., Agrawal, D., and Abbadi, A. 2002. Query Processing Over Peer-to-Peer Data Sharing Systems. Tech. rep., University of California at Santa Barbara.

Sahin, O. D., Gupta, A., Agrawal, D., and Abbadi, A. E. 2004. A peer-to-peer framework for caching range queries. In *ICDE '04: Proceedings of the 20th International Conference on Data Engineering*. IEEE Computer Society, 165.

Sakaryan, G., Wulff, M., and Unger, H. 2004. Search methods in p2p networks: a survey. In *I2CS-Innovative Internet Community Systems (I2CS 2004)*.

Sandberg, R., Goldberg, D., Kleiman, S., Walsh, D., and Lyon, B. 1985. Design and implementation of the Sun Network Filesystem. In *Proceedings of Summer 1985 USENIX Conference*. Portland, OR, USA, 119–130.

Saroiu, S., Gummadi, K. P., Dunn, R. J., Gribble, S. D., and Levy, H. M. 2002. An analysis of internet content delivery systems. *SIGOPS Oper. Syst. Rev. 36,* SI, 315–327.

Saroiu, S., Gummadi, P. K., and Gribble, S. D. 2002. A measurement study of peer-to-peer file sharing systems. In *Proceedings of Multimedia Computing and Networking, 2002*.

Satyanarayanan, M., Kistler, J. J., Kumar, P., Okasaki, M. E., Siegel, E. H., and Steere, D. C. 1990. Coda: A highly available file system for a distributed workstation environment. *IEEE Trans. Comput. 39,* 4, 447–459.

Stading, T., Maniatis, P., and Baker, M. 2002. Peer-to-peer caching schemes to address flash crowds. In *IPTPS '01: Revised Papers from the First International Workshop on Peer-to-Peer Systems*. Springer-Verlag, London, UK, 203–213.

Stoica, I., Adkins, D., Ratnasamy, S., Shenker, S., Surana, S., and Zhuang, S. 2002. Internet indirection infrastructure. In *IPTPS '01: Revised Papers from the First International Workshop on Peer-to-Peer Systems*. Springer-Verlag, London, UK, 191–202.

Stoica, I., Morris, R., Liben-Nowell, D., Karger, D., Kaashoek, M., Dabek, F., and Balakrishnan, H. 2001. Chord: A scalable peer-to-peer lookup protocol for internet applications. In *SIGCOMM '01: Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications*. ACM Press.

Tatarinov, I., Ives, Z., amd, J., Halevy, A., Suciu, D., Dalvi, N., Dong, X., Kadiyaska, Y., Miklau, G., and Mork, P. 2003. The piazza peer data management project. *ACM SIGMOD Record 32,* 3.

Triantafillou, P. and Pitoura, T. 2003. Towards a unifying framework for complex query processing over structured peer-to-peer data networks. In *DBISP2P*, K. Aberer, V. Kalogeraki, and M. Koubarakis, Eds. Lecture Notes in Computer Science, vol. 2944. Springer, 169–183.

Tsoumakos, D. and Roussopoulos, N. 2003. Adaptive probabilistic search for peer-to-peer networks. In *P2P '03: Proceedings of the 3rd International Conference on Peer-to-Peer Computing*. IEEE Computer Society, Washington, DC, USA, 102.

Ullman, J. D. 1997. Information integration using logical views. In *Proceedings of the 6th Int. Conf. on Database Theory (ICDT-97)*. Lecture Notes in Computer Science, pages 19–40. Springer-Verlag, 1997.

Valduriez, P. and Pacitti, E. 2004. Data management in large-scale p2p systems. In *Proceedings 6th Int. Conf. on High Performance Computing in Computational Sciences (VECPAR)*.

W3C 2001. World wide web consortium semantic web activities. http://www.w3.org/2001/sw.

Wiederhold, G. 1992. Mediators in the architecture of future information systems. *Computer 25,* 3, 38–49.

Wolpert, D. H. 1992. Stacked generalization. *Neural Networks 5*, 241–259.

Xu, L. and Embley, D. W. 2004. Combining the best of global-as-view and local-as-view for data integration. In *Information Systems Technologies and Its Applications - ISTA*. 123–136.

Zhao, B., Huang, L., Stribling, J., Rhea, S., Joseph, A. D., and Kubiatowicz, J. 2004. Tapestry: A resilient global-scale overlay for service deployment. *"IEEE Journal on Selected Areas in Communications, 22(1), January 2004" 22,* 1 (January).

Zhu, Y. Hu, Y. 2003. Exploiting client caches: An approach to building large web caches. In *Proceedings of the 2003 International Conference on Parallel Processing*. 419–426.

A Survey of Data Management in Peer-to-Peer Systems.
University of Waterloo Technical Report CS-2006-18.
Submitted June, 2006.