A Survey of Peer-to-Peer Content Distribution Technologies

STEPHANOS ANDROUTSELLIS-THEOTOKIS AND DIOMIDIS SPINELLIS

Athens University of Economics and Business

Distributed computer architectures labeled "peer-to-peer" are designed for the sharing of computer resources (content, storage, CPU cycles) by direct exchange, rather than requiring the intermediation or support of a centralized server or authority. Peer-to-peer architectures are characterized by their ability to adapt to failures and accommodate transient populations of nodes while maintaining acceptable connectivity and performance.

Content distribution is an important peer-to-peer application on the Internet that has received considerable research attention. Content distribution applications typically allow personal computers to function in a coordinated manner as a distributed storage medium by contributing, searching, and obtaining digital content.

In this survey, we propose a framework for analyzing peer-to-peer content distribution technologies. Our approach focuses on nonfunctional characteristics such as security, scalability, performance, fairness, and resource management potential, and examines the way in which these characteristics are reflected in—and affected by—the architectural design decisions adopted by current peer-to-peer systems.

We study current peer-to-peer systems and infrastructure technologies in terms of their distributed object location and routing mechanisms, their approach to content replication, caching and migration, their support for encryption, access control, authentication and identity, anonymity, deniability, accountability and reputation, and their use of resource trading and management schemes.

Categories and Subject Descriptors: C.2.1 [**Computer-Communication Networks**]: Network Architecture and Design—*Network topology*; C.2.2 [**Computer-Communication Networks**]: Network Protocols—*Routing protocols*; C.2.4 [**Computer-Communication Networks**]: Distributed Systems—*Distributed databases*; H.2.4 [**Database Management**]: Systems—*Distributed databases*; H.3.4 [**Information Storage and Retrieval**]: Systems and Software—*Distributed systems*

General Terms: Algorithms, Design, Performance, Reliability, Security

Additional Key Words and Phrases: Content distribution, DOLR, DHT, grid computing, p2p, peer-to-peer

1. INTRODUCTION

A new wave of network architectures labeled *peer-to-peer* is the basis of operation of distributed computing systems such as [Gnutella 2003], Seti@ Home [SetiAtHome 2003], OceanStore [Kubiatowicz et al. 2000], and many others. Such architectures are generally characterized by the direct sharing

Authors' address: Athens University of Economics and Business, 76 Patission St., GR-104 34, Athens, Greece; email: stheotok@aueb.gr.

of computer resources (CPU cycles, storage, content) rather than requiring the intermediation of a centralized server.

The motivation behind basing applications on peer-to-peer architectures derives to a large extent from their ability to function, scale, and self-organize in the presence of a highly transient population of nodes, network, and computer failures, without the need of a central server and the overhead of its administration. Such architectures typically have as inherent characteristics scalability, resistance to censorship and centralized control, and increased access to resources. Administration, maintenance, responsibility for the operation, and even the notion of "ownership" of peer-to-peer systems are also distributed among the users, instead of being handled by a single company, institution or person (see also Agre [2003] for an interesting discussion of institutional change through decentralized architectures). Finally, peer-to-peer architectures have the potential to accelerate communication processes and reduce collaboration costs through the ad hoc administration of working groups [Schoder and Fischbach 2003].

This report surveys peer-to-peer content distribution technologies, aiming to provide a comprehensive account of applications, features, and implementation techniques. As this is a new and (thankfully) rapidly evolving field, and advances and new capabilities are constantly being introduced, this article will be presenting what essentially constitutes a "snapshot" of the state of the art around the time of its writing—as is unavoidably the case for any survey of a thriving research field. We do, however, believe that the core information and principles presented will remain relevant and useful for the reader.

In the next section, we define the basic concepts of peer-to-peer computing. We classify peer-to-peer systems into three categories (communication and collaboration, distributed computation, and content distribution). Content distribution systems are further discussed and categorized.

We then present the main attributes of peer-to-peer content distribution systems, and the aspects of their architectural design which affect these attributes are analyzed with reference to specific existing peer-to-peer content distribution systems and technologies.

Throughout this report the terms "node", "peer" and "user" are used interchangeably, according to the context, to refer to the entities that are connected in a peer-to-peer network.

1.1. Defining Peer-to-Peer Computing

A quick look at the literature reveals a considerable number of different definitions of "peer-to-peer", mainly distinguished by the "broadness" they attach to the term.

The strictest definitions of "pure" peer-to-peer refer to totally distributed systems, in which all nodes are completely equivalent in terms of functionality and tasks they perform. These definitions fail to encompass, for example, systems that employ the notion of "supernodes" (nodes that function as dynamically assigned localized mini-servers) such as Kazaa [2003], which are, however, widely accepted as peer-to-peer, or systems that rely on some centralized server infrastructure for a subset of noncore tasks (e.g. bootstrapping, maintaining reputation ratings, etc).

According to a broader and widely accepted definition in Shirky [2000], "peer-to-peer is a class of applications that take advantage of resources—storage, cycles, content, human presence—available at the edges of the internet". This definition, however, encompasses systems that completely rely upon centralized servers for their operation (such as seti@home, various instant messaging systems, or even the notorious Napster), as well as various applications from the field of Grid computing.

Overall, it is fair to say that there is no general agreement about what "is" and what "is not" peer-to-peer.

We feel that this lack of agreement on a definition—or rather the acceptance of various different definitions—is, to a large extent, due to the fact that systems or applications are labeled "peer-to-peer" not because of their internal operation or architecture, but rather as a result of how they are perceived "externally", that is, whether they give the impression of providing direct interaction between computers. As a result, different definitions of "peer-to-peer" are applied to accommodate the various different cases of such systems or applications.

From our perspective, we believe that the two defining characteristics of peer-to-peer architectures are the following:

—The sharing of computer resources by direct exchange, rather than requiring the intermediation of a centralized server. Centralized servers can sometimes be used for specific tasks (system bootstrapping, adding new nodes to the network, obtain global keys for data encryption), however, systems that rely on one or more global centralized servers for their basic operation (e.g. for maintaining a global index and searching through it—Napster, Publius) are clearly stretching the definition of peer-to-peer.

As the nodes of a peer-to-peer network cannot rely on a central server coordinating the exchange of content and the operation of the entire network, they are required to actively participate by independently and unilaterally performing tasks such as searching for other nodes, locating or caching content, routing information and messages, connecting to or disconnecting from other neighboring nodes, encrypting, introducing, retrieving, decrypting and verifying content, as well as others.

—Their ability to treat instability and variable connectivity as the norm, automatically adapting to failures in both network connections and computers, as well as to a transient population of nodes.

This fault-tolerant, self-organizing capacity suggests the need for an adaptive network topology that will change as nodes enter or leave and network connections fail or recover, in order to maintain its connectivity and performance.

We therefore propose the following definition:

> *Peer-to-peer systems are distributed systems consisting of interconnected nodes able to self-organize into network topologies with the purpose of sharing resources such as content, CPU cycles, storage and bandwidth, capable of adapting to failures and accommodating transient populations of nodes while maintaining acceptable connectivity and performance, without requiring the intermediation or support of a global centralized server or authority.*

This definition is meant to encompass "degrees of centralization" ranging from the pure, completely decentralized systems such as Gnutella, to "partially centralized" systems[1] such as Kazaa. However, for the purposes of this survey, we shall not restrict our presentation and discussion of architectures and systems to our own proposed definition, and we will take into account systems that are considered peer-to-peer by other definitions as well, including systems that employ a centralized server (such as Napster, instant messaging applications, and others).

The focus of our study is content distribution, a significant area of peer-to-peer systems that has received considerable research attention.

1.2. Peer-to-Peer and Grid Computing

Peer-to-peer and Grid computing are two approaches to distributed computing, both concerned with the organization of resource sharing in large-scale computational societies.

Grids are distributed systems that enable the large-scale coordinated use and sharing of geographically distributed resources, based on persistent, standards-based service infrastructures, often with a high-performance orientation [Foster et al. 2001].

[1] In our definition, we refer to "global" servers, to make a distinction from the dynamically assigned "supernodes" of partially centralized systems (see also Section 3.3.3)

As Grid systems increase in scale, they begin to require solutions to issues of self-configuration, fault tolerance, and scalability, for which peer-to-peer research has much to offer.

Peer-to-peer systems, on the other hand, focus on dealing with instability, transient populations, fault tolerance, and self-adaptation. To date, however, peer-to-peer developers have worked mainly on vertically integrated applications, rather than seeking to define common protocols and standardized infrastructures for interoperability.

In summary, one can say that "Grid computing addresses infrastructure, but not yet failure, while peer-to-peer addresses failure, but not yet infrastructure" [Foster and Iamnitchi 2003].

In addition to this, the form of sharing initially targeted by peer-to-peer has been of limited functionality, providing a global content distribution and filesharing space lacking any form of access control.

As peer-to-peer technologies move into more sophisticated and complex applications, such as structured content distribution, desktop collaboration, and network computation, it is expected that there will be a strong convergence between peer-to-peer and Grid computing [Foster 2000]. The result will be a new class of technologies combining elements of both peer-to-peer and Grid computing, which will address scalability, self-adaptation, and failure recovery, while, at the same time, providing a persistent and standardized infrastructure for interoperability.

1.3. Classification of Peer-to-Peer Applications

Peer-to-peer architectures have been employed for a variety of different application categories, which include the following.

Communication and Collaboration. This category includes systems that provide the infrastructure for facilitating direct, usually real-time, communication and collaboration between peer computers. Examples include chat and instant messaging applications, such as Chat/Irc, Instant Messaging (Aol, Icq, Yahoo, Msn), and Jabber [Jabber 2003].

Distributed Computation. This category includes systems whose aim is to take advantage of the available peer computer processing power (CPU cycles). This is achieved by breaking down a computer-intensive task into small work units and distributing them to different peer computers, that execute their corresponding work unit and return the results. Central coordination is invariably required, mainly for breaking up and distributing the tasks and collecting the results. Examples of such systems include projects such as Seti@home [Sullivan III et al. 1997; SetiAtHome 2003], genome@home [Larson et al. 2003; GenomeAtHome 2003], and others.

Internet Service Support. A number of different applications based on peer-to-peer infrastructures have emerged for supporting a variety of Internet services. Examples of such applications include peer-to-peer multicast systems [VanRenesse et al. 2003; Castro et al. 2002], Internet indirection infrastructures [Stoica et al. 2002], and security applications, providing protection against denial of service or virus attacks [Keromytis et al. 2002; Janakiraman et al. 2003; Vlachos et al. 2004].

Database Systems. Considerable work has been done on designing distributed database systems based on peer-to-peer infrastructures. Bernstein et al. [2002] propose the Local Relational Model (LRM), in which the set of all data stored in a peer-to-peer network is assumed to be comprised of inconsistent local relational databases interconnected by sets of "acquaintances" that define translation rules and semantic dependencies between them. PIER [Huebsch et al. 2003] is a scalable distributed query engine built on top of a peer-to-peer overlay network topology that allows relational queries to run across thousands of computers. The Piazza system [Halevy et al. 2003] provides an infrastructure for building semantic Web [Berners-Lee et al. 2001] applications, consisting of nodes that can supply either source data (e.g. from a relational database), schemas (or ontologies) or both. Piazza nodes are transitively connected by chains of *mappings* between pairs of nodes, allowing queries to be distributed across the Piazza network. Finally, Edutella [Nejdl et al. 2003] is an open-source project that builds on the W3C metadata standard RDF, to provide a metadata infrastructure and querying capability for peer-to-peer applications.

Content Distribution. Most of the current peer-to-peer systems fall within the category of content distribution, which includes systems and infrastructures designed for the sharing of digital media and other data between users. Peer-to-peer content distribution systems range from relatively simple direct filesharing applications, to more sophisticated systems that create a distributed storage medium for securely and efficiently publishing, organizing, indexing, searching, updating, and retrieving data. There are numerous such systems and infrastructures. Some examples are: the late Napster, Publius [Waldman et al. 2000], Gnutella [Gnutella 2003], Kazaa [Kazaa 2003], Freenet [Clarke et al. 2000], MojoNation [MojoNation 2003], Oceanstore [Kubiatowicz et al. 2000], PAST [Druschel and Rowstron 2001], Chord [Stoica et al. 2001], Scan [Chen et al. 2000], FreeHaven [Dingledine et al. 2000], Groove [Groove 2003], and Mnemosyne [Hand and Roscoe 2002].

This survey will focus on content distribution, one of the most prominent application areas of peer-to-peer systems.

1.4. Peer-to-Peer Content Distribution

In its most basic form, a peer-to-peer content distribution system creates a distributed storage medium that allows for the publishing, searching, and retrieval of files by members of its network. As systems become more sophisticated, nonfunctional features may be provided, including provisions for security, anonymity, fairness, increased scalability and performance, as well as resource management and organization capabilities. All of these will be discussed in the following sections.

An examination of current peer-to-peer technologies in this context suggests that they can be grouped as follows (with examples in Tables I and II).

—*Peer-to-Peer Applications.* This category includes content distribution systems that are based on peer-to-peer technology. We attempt to further subdivide them into the following two groups, based on their application goals and perceived complexity:
Peer-to-peer "file exchange" systems. These systems are targeted towards simple, one-off file exchanges between peers. They are used for setting up a network of peers and providing facilities for searching and transferring files between them. These are typically light-weight applications that adopt a best-effort approach without addressing security, availability, and persistence. It is mainly systems in this category that are responsible for spawning the reputation (and in some cases notoriety) of peer-to-peer technologies.
Peer-to-peer content publishing and storage systems. These systems are targeted towards creating a distributed storage medium in—and through—which users will be able to publish, store, and distribute content in a secure and persistent manner. Such content is meant to be accessible in a controlled manner by peers with appropriate privileges. The main focus of such systems is security and persistence, and often the aim is to incorporate provisions for accountability, anonymity and censorship resistance, as well as persistent content management (updating, removing, version control) facilities.

—*Peer-to-Peer Infrastructures.* This category includes peer-to-peer based infrastructures that do not constitute working applications, but provide peer-to-peer based services and application frameworks. The following infrastructure services are identified:
Routing and location. Any peer-to-peer content distribution system relies on a network of peers within which requests and messages must be routed

Table I. A Classification of Current Peer-to-Peer Systems
(RM: Resource Management; CR: Censorship Resistance; PS: Performance and Scalability; SPE: Security, Privacy and Encryption; A: Anonymity; RA: Reputation and Accountability; RT: Resource Trading.)

Peer-to-Peer File Exchange Systems

System	Brief Description
Napster	Distributed file sharing—hybrid decentralized.
Kazaa [2003]	Distributed file sharing—partially centralized.
Gnutella [2003]	Distributed file sharing—purely decentralized.

Peer-to-Peer Content Publishing and Storage Systems

System	Brief Description	Main Focus
Scan [Chen et al. 2000]	A dynamic, scalable, efficient content distribution network. Provides dynamic content replication.	PS
Publius [Waldman et al. 2000]	A censorship-resistant system for publishing content. Static list of servers. Enhanced content management (update and delete).	RM
Groove [Groove 2003]	Internet communications software for direct real-time peer-to-peer interaction.	RM,PS,SPE
FreeHaven [Dingledine et al. 2000]	A flexible system for anonymous storage.	A,RA
Freenet [Clarke et al. 2000]	Distributed anonymous information storage and retrieval system.	A,RA
MojoNation [MojoNation 2003]	Distributed file storage. Fairness through the use of currency mojo.	SPE,RT
Oceanstore [Kubiatowicz et al. 2000]	An architecture for global scale persistent storage. Scalable, provides security and access control.	RM,PS,SPE
Intermemory [Chen et al. 1999]	System of networked computers. Donate storage in exchange for the right to publish data.	RT
Mnemosyne [Hand and Roscoe 2002]	Peer-to-peer steganographic storage system. Provides privacy and plausible deniability.	SPE
PAST [Druschel and Rowstron 2001]	Large scale persistent peer-to-peer storage utility.	PS,SPE
Dagster [Stubblefield and Wallach 2001]	A censorship-resistant document publishing system.	CR,SPE
Tangler [Waldman and Mazi 2001]	A content publishing system based on document entanglements.	CR,SPE

with efficiency and fault tolerance, and through which peers and content can be efficiently located. Different infrastructures and algorithms have been developed to provide such services.

Anonymity. Peer-to-peer based infrastructure systems have been designed with the explicit aim of providing user anonymity.

Reputation Management. In a peer-to-peer network, there is no central organization to maintain reputation information for users and their behavior. Reputation information is, therefore, hosted in the various network nodes. In order for such reputation information to be kept secure, up-to-date, and available throughout the network, complex reputation management infrastructures need to be employed.

2. ANALYSIS FRAMEWORK

In this survey, we present a description and analysis of applications, systems, and infrastructures that are based on peer-to-peer architectures and aim at either offering content distribution solutions, or supporting content distribution related activities.

Our approach is based on:

—identifying the feature space of nonfunctional properties and characteristics of content distribution systems,

—determining the way in which the nonfunctional properties depend on, and can be affected by, various design features, and

—providing an account, analysis, and evaluation of the design features and solutions that have been adopted by

Table II. A Classification of Current Peer-to-Peer Infrastructures

Infrastructures for routing and location	
Chord [Stoica et al. 2001]	A scalable peer-to-peer lookup service. Given a key, it maps the key to a node.
CAN [Ratnasamy et al. 2001]	Scalable content addressable network. A distributed infrastructure that provides hash-table functionality for mapping file names to their locations.
Pastry [Rowstron and Druschel 2001]	Infrastructure for fault-tolerant wide-area location and routing.
Tapestry [Zhao et al. 2001]	Infrastructure for fault-tolerant wide-area location and routing.
Kademlia [Mayamounkov and Mazieres 2002]	A scalable peer-to-peer lookup service based on the XOR metric.
Infrastructures for anonymity	
Anonymous remailer mixnet [Berthold et al. 1998]	Infrastructure for anonymous connection.
Onion Routing [Goldschlag et al. 1999]	Infrastructure for anonymous connection.
ZeroKnowledge Freedom [Freedom 2003]	Infrastructure for anonymous connection.
Tarzan [Freedman et al. 2002]	A peer-to-peer decentralized anonymous network layer.
Infrastructures for reputation management	
Eigentrust [Kamvar et al. 2003]	A Distributed reputation management algorithm.
A Partially distributed reputation management system [Gupta et al. 2003]	A partially distributed approach based on a debit/credit or debit-only scheme.
PeerTrust [Xiong and Liu 2002]	A decentralized, feedback-based reputation management system using satisfaction and number of interaction metrics.

current peer-to-peer systems, as well as their shortcomings, potential improvements, and proposed alternatives.

For the identification of the nonfunctional characteristics on which our study is based, we used the work of Shaw and Garlan [1995], which we adapted to the field of peer-to-peer content distribution.

The various design features and solutions that we examine, as well as their relationship to the relevant nonfunctional characteristics, were assembled through a detailed analysis and study of current peer-to-peer content distribution systems that are either deployed, researched, or proposed.

The resulting analysis framework is illustrated in Figure 1. The boxes in the periphery show the most important attributes of peer-to-peer content distribution systems, described as follows.

Security. Further analyzed in terms of:

Integrity and authenticity. Safeguarding the accuracy and completeness of data and processing methods. Unauthorized entities cannot change data; adversaries cannot substitute a forged document for a requested one.

Privacy and confidentiality. Ensuring that data is accessible only to those authorized to have access, and that there is control over what data is collected, how it is used, and how it is maintained.

Availability and persistence. Ensuring that authorized users have access to data and associated assets when required. For a peer-to-peer content distribution system this often means always. This property entails stability in the presence of failure, or changing node populations.

Scalability. Maintaining the system's performance attributes independent of the number of nodes or documents in its network. A dramatic increase in the number of nodes or documents will have minimal effect on performance and availability.

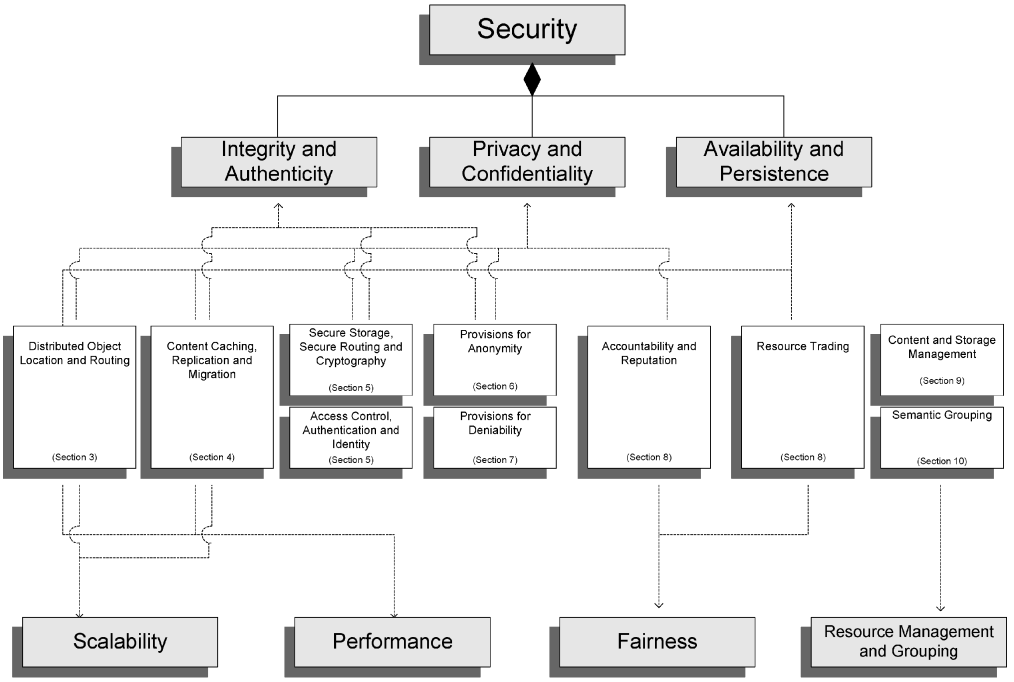
Security
Integrity and Authenticity
Privacy and Confidentiality
Availability and Persistence
Distributed Object Location and Routing (Section 3)
Content Caching, Replication and Migration (Section 4)
Secure Storage, Secure Routing and Cryptography (Section 5)
Access Control, Authentication and Identity (Section 5)
Provisions for Anonymity (Section 6)
Provisions for Deniability (Section 7)
Accountability and Reputation (Section 8)
Resource Trading (Section 8)
Content and Storage Management (Section 9)
Semantic Grouping (Section 10)
Scalability
Performance
Fairness
Resource Management and Grouping

Fig. 1. Illustration of the way in which various design features affect the main characteristics of peer-to-peer content distribution systems.

Performance. The time required for performing the operations allowed by the system, typically publication, searching, and retrieval of documents.

Fairness. Ensuring that users offer and consume resources in a fair and balanced manner. May rely on accountability, reputation, and resource trading mechanisms.

Resource Management Capabilities. In their most basic form, peer-to-peer content distribution systems allow the publishing, searching, and retrieval of documents. More sophisticated systems may afford more advanced resource management capabilities, such as editing or removal of documents, management of storage space, and operations on metadata.

Semantic Grouping of Information. An area of research that has attracted considerable attention recently is the semantic grouping and organization of content and information in peer-to-peer networks. Various grouping schemes are encountered, such as semantic grouping, based on the content itself; grouping, based on locality or network distance; grouping, based on organization ties, as well as others.

The relationships between nonfunctional characteristics are depicted as a UML diagram in Figure 1. The Figure schematically illustrates the relationship between the various design features and the main characteristics of peer-to-peer content distribution systems.

The boxes in the center show the design decisions that affect these attributes. We note that these design decisions are mostly independent and orthogonal.

We see, for example, that the performance of a peer-to-peer content distribution system is affected by the distributed object location and routing mechanisms, as well as by the data replication, caching, and migration algorithms. Fairness, on the other hand, depends on the system's provisions for accountability and reputation, as well as on any resource trading mechanisms that may be implemented.

In the following sections of this survey, the different design decisions and features will be presented and discussed, based on an examination of existing peer-to-peer content distribution systems, and with reference to the way in which they affect the main attributes of these systems, as presented. A relevant presentation and discussion of various desired properties of peer-to-peer systems can also be found in Kubiatowics [2003], while an analysis of peer-to-peer systems from the end-user perspective can be found in Lee [2003].

3. PEER-TO-PEER DISTRIBUTED OBJECT LOCATION AND ROUTING

The operation of any peer-to-peer content distribution system relies on a network of peer computers (nodes), and connections (edges) between them. This network is formed on top of—and independently from—the underlying physical computer (typically IP) network, and is thus referred to as an "overlay" network. The topology, structure, and degree of centralization of the overlay network, and the routing and location mechanisms it employs for messages and content are crucial to the operation of the system, as they affect its fault tolerance, self-maintainability, adaptability to failures, performance, scalability, and security; in short almost all of the system's attributes as laid out in Section 2 (see also Figure 1).

Overlay networks can be distinguished in terms of their centralization and structure.

3.1. Overlay Network Centralization

Although in their purest form peer-to-peer overlay networks are supposed to be totally decentralized, in practice this is not always true, and systems with various degrees of centralization are encountered. Specifically, the following three categories are identified.

Purely Decentralized Architectures. All nodes in the network perform exactly the same tasks, acting both as servers and clients, and there is no central coordination of their activities. The nodes of such networks are often termed "servents" (SERVers+clieENTS).

Partially Centralized Architectures. The basis is the same as with purely decentralized systems. Some of the nodes, however, assume a more important role, acting as local central indexes for files shared by local peers. The way in which these *supernodes* are assigned their role by the network varies between different systems. It is important, however, to note that these supernodes do not constitute single points of failure for a peer-to-peer network, since they are dynamically assigned and, if they fail, the network will automatically take action to replace them with others.

Hybrid Decentralized Architectures. In these systems, there is a central server facilitating the interaction between peers by maintaining directories of metadata, describing the shared files stored by the peer nodes. Although the end-to-end interaction and file exchanges may take place directly between two peer nodes, the central servers facilitate this interaction by performing the lookups and identifying the nodes storing the files. The terms "peer-through-peer" or "broker mediated" are sometimes used for such systems [Kim 2001].

Obviously, in these architectures, there is a single point of failure (the central server). This typically renders them inherently unscalable and vulnerable to censorship, technical failure, or malicious attack.

3.2. Network Structure

By structure, we refer to whether the overlay network is created nondeterministically (ad hoc) as nodes and content are added, or whether its creation is based on specific rules. We categorize peer-to-peer networks as follows, in terms of their structure:

Unstructured. The placement of content (files) is completely unrelated to the overlay topology.

In an unstructured network, content typically needs to be located. Searching mechanisms range from brute force methods, such as flooding the network with propagating queries in a breadth-first or depth-first manner until the desired content is located, to more sophisticated and resource-preserving strategies that include the use of random walks and routing indices (discussed in more detail in Section 3.3.4). The searching mechanisms employed in unstructured networks have obvious implications, particularly in regards to matters of availability, scalability, and persistence.

Unstructured systems are generally more appropriate for accommodating highly-transient node populations. Some representative examples of unstructured systems are Napster, Publius [Waldman et al. 2000], Gnutella [Gnutella 2003], Kazaa [Kazaa 2003], Edutella [Nejdl et al. 2003], FreeHaven [Dingledine et al. 2000], as well as others.

Structured. These have emerged mainly in an attempt to address the scalability issues that unstructured systems were originally faced with. In structured networks, the overlay topology is tightly controlled and files (or pointers to them) are placed at precisely specified locations. These systems essentially provide a mapping between content (e.g. file identifier) and location (e.g. node address), in the form of a distributed routing table, so that queries can be efficiently routed to the node with the desired content [Lv et al. 2002].

Structured systems offer a scalable solution for exact-match queries, that is, queries where the exact identifier of the requested data object is known (as compared to keyword queries). Using exact-match queries as a substrate for keyword queries remains an open research problem for distributed environments [Witten et al. 1999].

A disadvantage of structured systems is that it is hard to maintain the structure required for efficiently routing messages in the face of a very transient node population, in which nodes are joining and leaving at a high rate [Lv et al. 2002].

Typical examples of structured systems include Chord [Stoica et al. 2001], CAN [Ratnasamy et al. 2001], PAST [Druschel and Rowstron 2001], Tapestry [Zhao et al. 2001] among others.

Table III. A classification of Peer-to-Peer Content Distribution Systems and Location and Routing Infrastructures in Terms of Their Network Structure, With Some Typical Examples

	Centralization		
	Hybrid	Partial	None
Unstructured	Napster, Publius	Kazaa, Morpheus, Gnutella, Edutella	Gnutella, FreeHaven
Structured Infrastructures			Chord, CAN, Tapestry, Pastry
Structured Systems			OceanStore, Mnemosyne, Scan, PAST, Kademlia, Tarzan

A category of networks that are in between structured and unstructured are referred to as *loosely structured* networks. Although the location of content is not completely specified, it is affected by routing hints. A typical example is Freenet [Clarke et al. 2000; Clake et al. 2002].

Table III summarizes the categories we outlined, with examples of peer-to-peer content distribution systems and architectures. Note that all structured and loosely structured systems are inherently purely decentralized; form follows function.

In the following sections, the overlay network topology and operation of different peer-to-peer systems is discussed, following the above classification, according to degree of centralization and network structure.

3.3. Unstructured Architectures

3.3.1. Hybrid Decentralized. Figure 2 illustrates the architecture of a typical hybrid decentralized peer-to-peer system.

Each client computer stores content (files) shared with the rest of the network. All clients connect to a central directory server that maintains:

—A table of registered user connection information (IP address, connection bandwidth etc.)

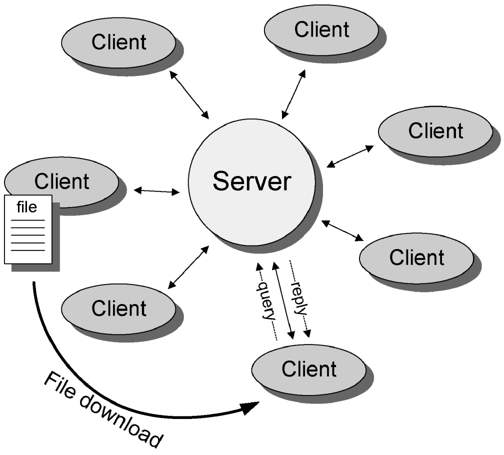

Fig. 2. Typical hybrid decentralized peer-to-peer architecture. A central directory server maintains an index of the metadata for all files in the network.

—A table listing the files that each user holds and shares in the network, along with metadata descriptions of the files (e.g. filename, time of creation, etc.)

A computer that wishes to join the network contacts the central server and reports the files it maintains. Client computers send requests for files to the server. The server searches for matches in its index, returning a list of users that hold the matching file. The user then opens direct connections with one or more of the peers that hold the requested file, and downloads it (see Figure 2).

The advantage of hybrid decentralized systems is that they are simple to implement, and they locate files quickly and efficiently. Their main disadvantage is that they are vulnerable to censorship, legal action, surveillance, malicious attack, and technical failure, since the content shared, or at least descriptions of it, and the ability to access it are controlled by the single institution, company, or user maintaining the central server. Furthermore, these systems are considered inherently unscalable, as there are bound to be limitations to the size of the server database and its capacity to respond to queries. Large Web search engines have, however, repeatedly provided counterexamples to this notion.

Examples of hybrid decentralized content distribution systems include the notorious Napster and Publius systems [Waldman et al. 2000] that rely on a static, system-wide list of servers. Their architecture does not provide any smooth, decentralized support for adding a new server, or removing dead or malicious servers. A comprehensive study of the behavior of hybrid peer-to-peer systems and a comparison of their query characteristics and their performance in terms of bandwidth and CPU cycle consumption is presented in Yang and Garcia-Molina [2001].

It should be noted that systems that do not fall under the hybrid decentralized category may still use some central administration server to a limited extent, for example, for initial system bootstrapping (e.g. Mojonation [MojoNation 2003]), or for allowing new users to join the network by providing them with access to a list of current users (e.g. gnutellahosts.com for the gnutella network).

3.3.2. Purely Decentralized. In this section, we examine the Gnutella network [Gnutella 2003], an interesting and representative member of purely decentralized peer-to-peer architectures, due to its open architecture, achieved scale, and self-organizing structure. FreeHaven [Dingledine et al. 2000] is another system using routing and search mechanisms similar to those of Gnutella.

Like most peer-to-peer systems, Gnutella builds a virtual overlay network with its own routing mechanisms [Ripeanu and Foster 2002], allowing its users to share files with other peers.

There is no central coordination of the activities in the network and users connect to each other directly through a software application that functions both as a client and a server (users are referred to as a *servents*).

Gnutella uses IP as its underlying network service, while the communication between servents is specified in a form of application level protocol supporting four types of messages [Jovanovich et al. 2001]:

Ping. A request for a certain host to announce itself.

Pong. Reply to a Ping message. It contains the IP and port of the responding host and number and size of the files being shared.

Query. A search request. It contains a search string and the minimum speed requirements of the responding host.

Query Hits. Reply to a Query message. It contains the IP, port, and speed of the responding host, the number of matching files found, and their indexed result set.

After joining the Gnutella network (by connecting to nodes found in databases such as gnutellahosts.com), a node sends out a *Ping* message to any node it is connected to. The nodes send back a *Pong* message identifying themselves, and also propagate the *Ping* message to their neighbors.

In order to locate a file in unstructured systems such as gnutella, nondeterministic searches are the only option since the nodes have no way of guessing where (at which nodes) the files may lie.

The original Gnutella architecture uses a flooding (or broadcast) mechanism to distribute *Ping* and *Query* messages: each Gnutella node forwards the received messages to all of its neighbors. The response messages received are routed back along the opposite path through which the original request arrived. To limit the spread of messages through the network, each message header contains a time-to-live (TTL) field. At each hop, the value of this field is decremented, and when it reaches zero, the message is dropped.

The above mechanism is implemented by assigning each message a unique identifier and equipping each host with a dynamic routing table of message identifiers and node addresses. Since the response messages contain the same ID as the original messages, the host checks its routing table to determine along which link the response message should be forwarded. In order to avoid loops, the nodes use the unique message identifiers to detect and drop duplicate messages, to improve efficiency, and preserve network bandwidth [Jovanovic 2000].

Once a node receives a QueryHit message, indicating that the target file has been identified at a certain node, it initiates a download by establishing a direct connection between the two nodes. Figure 3 illustrates an example of the Gnutella search mechanism.

Scalability issues in the original purely decentralized systems arose from the fact that the use of the TTL effectively segmented the network into "sub-networks", imposing on each user a virtual "horizon" beyond which their messages could not reach [Jovanovic 2000]. Removing the TTL limit, on the other hand, would result in the network being swamped with messages.

Significant research has been carried out to address the above issues, and various solutions have been proposed. These will be discussed in more detail in Section 3.3.4.

3.3.3. Partially Centralized. Partially centralized systems are similar to purely decentralized, but they use the concept of *supernodes*: nodes that are dynamically assigned the task of servicing a small subpart of the peer network by indexing and caching files contained therein.

Peers are automatically elected to become supernodes if they have sufficient bandwidth and processing power (although a configuration parameter may allow users to disable this feature) [FastTrack 2003].

Supernodes index the files shared by peers connected to them, and proxy search requests on behalf of these peers. All queries are therefore initially directed to supernodes.

Two major advantages of partially centralized systems are that:

—Discovery time is reduced in comparison with purely decentralized systems, while there still is no unique point of failure. If one or more supernodes go down, the nodes connected to them can open new connections with other supernodes, and the network will continue to operate.

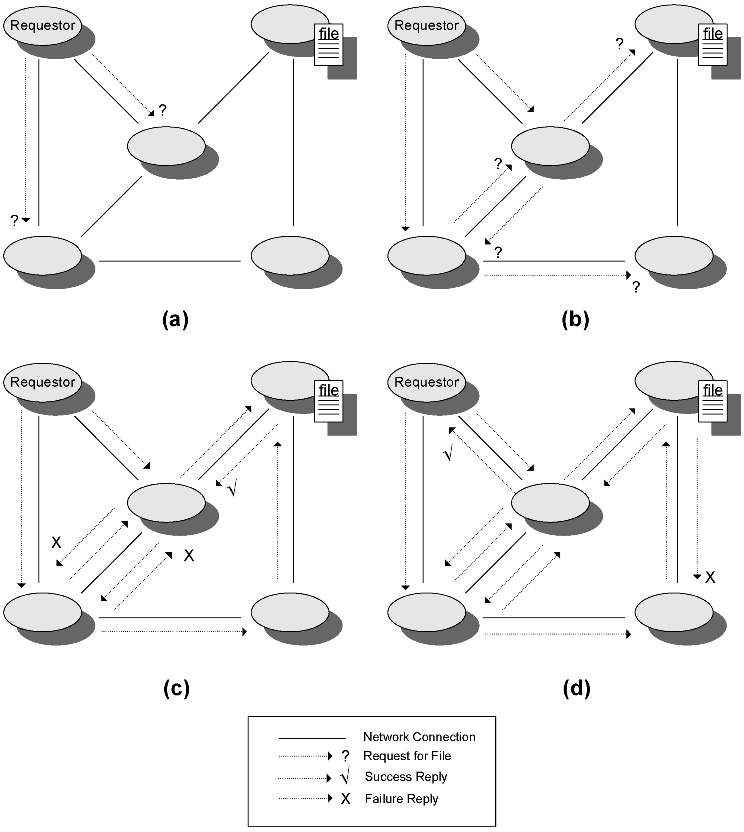

Fig. 3. An example of the Gnutella search mechanism. Solid lines between the nodes represent connections of the Gnutella network. The search originates at the "requestor" node, for a file maintained by another node. Request messages are dispatched to all neighboring nodes, and propagated from node-to-node as shown in the four consecutive steps (a) to (d). When a node receives the same request message (based on its unique identifier) multiple times, it replies with a failure message to avoid loops and minimize traffic. When the file is identified, a success message is returned.

—The inherent heterogeneity of peer-to-peer networks is taken advantage of, and exploited. In a purely decentralized network, all of the nodes will be equally (and usually heavily) loaded, regardless of their CPU power, bandwidth, or storage capabilities. In partially centralized systems, however, the supernodes will undertake a large portion of the entire network load, while most of the other (so called "normal") nodes will be very lightly loaded, in comparison (see also [Lv et al. 2002; Zhichen et al. 2002]).

Kazaa is a typical, widely used instance of a partially centralized system implementation (as it is a proprietary system, there is no detailed documentation on its structure and operation). Edutella [Nejdl et al. 2003] is another partially centralized architecture.

Yang and Garcia-Molina [2002a, 2002b] present research addressing the design of, and searching techniques for, partially centralized peer-to-peer networks.

The concept of supernodes has also been proposed in a more recent version of the Gnutella protocol. A mechanism for dynamically selecting supernodes organizes the Gnutella network into an interconnection of *superpeers* (as they are referred to) and client nodes.

When a node with enough CPU power joins the network, it immediately becomes a *superpeer* and establishes connections with other superpeers, forming a flat unstructured network of superpeers. If it establishes a minimum required number of connections to client nodes within a specified time, it remains a *superpeer*. Otherwise, it turns into a regular client node.

3.3.4. Shortcomings and Evolution of Unstructured Architectures. A number of methods have been proposed to overcome the original unscalability of unstructured peer-to-peer systems. These have been shown to drastically improve their performance.

Lv et al. [2002] proposed to replace the original flooding approach by multiple parallel random walks, where each node chooses a neighbor at random, and propagates the request only to it. The use of random walks, combined with proactive object replication (discussed in Section 4), was found to significantly improve the performance of the system as measured by query resolution time (in terms of numbers of hops), per-node query load, and message traffic generated. Proactive data replication schemes are further examined in Cohen and Shenker [2001].

Yang and Garcia-Molina [2002b] suggested the use of more sophisticated broadcast policies, selecting which neighbors to forward search queries to based on their past history, as well as the use of *local indices*: data structures where each node maintains an index of the data stored at nodes located within a radius from itself.

A similar solution to the information retrieval problem is proposed by Kalogeraki et al. [2002] in the form of an *Intelligent Search Mechanism* built on top of a *Modified Random BFS Search Mechanism*. Each peer forwards queries to a subset of its neighbors, selecting them based on a profile mechanism that maintains information about their performance in recent queries. Neighbours are ranked according to their profiles and queries are forwarded selectively to the most appropriate ones only.

Chawathe et al. [2003] presented the Gia System, which addresses the above issues by means of:

—A dynamic topology adaptation protocol that ensures that most nodes will be at a short distance from high capacity nodes. This protocol, coupled with replicating pointers to the content of neighbors, ensures that high capacity nodes will be able to provide answers to a very large number of queries.

—An active flow control scheme that exploits network heterogeneity to avoid hotspots, and

—A search protocol that is based on random walks directed towards high capacity nodes.

Simulations of Gia were found to increase overall system capacity by three to five orders of magnitude. Similar approaches, based on taking advantage of the underlying network heterogeneity, are described in Lv et al. [2002] and Zhichen et al. [2002].

In another approach, Crespo and Garcia-Molina [2002] use *routing indices* to address the searching and scalability issues. Routing indices are tables of information about other nodes, stored within each node. They provide a list of neighbors that are most likely to be "in the direction" of the content corresponding to the query. These tables contain information about the total number of files maintained by nearby nodes, as well as the number of files corresponding to various topics. Three different variations of the approach are presented, and simulations are shown to improve performance by 1-2 orders of magnitude with respect to flooding techniques.

Finally, the connectivity properties and reliability of unstructured peer-to-peer networks such as Gnutella have been studied in Ripeanu and Foster [2002]. In particular, emphasis was placed on the fact that peer-to-peer networks exhibit the properties of power-law networks, in which the number of nodes with L links is proportional to L^{-k}, where k is a network dependent constant. In other words, most nodes have few links (thus, a large fraction of them can be taken away without seriously damaging the network connectivity), while there are a few highly-connected nodes which, if taken away, are likely to cause the whole network to be broken down in disconnected pieces. One implication of this is that such networks are robust when facing random node attacks, yet vulnerable to well-planned attacks. The topology mismatch between the Gnutella network and the underlying physical network infrastructure was also documented in Ripeanu and Foster [2002]. Tsoumakos and Roussopoulos [2003] present a more comprehensive analysis and comparison of the above methods.

Overall, unstructured peer-to-peer content distribution systems might be the preferred choice for applications where the following assumptions hold:

—keyword searching is the common operation,

—most content is typically replicated at a fair fraction of participating sites,

—the node population is highly transient,

—users will accept a best-effort content retrieval approach, and

—the network size is not so large as to incur scalability problems [Lv et al. 2002] (The last issue is alleviated through the various methods described in this Section).

3.4. Structured Architectures

The various structured content distribution systems and infrastructures employ different mechanisms for routing messages and locating data. Four of the most interesting and representative mechanisms and their corresponding systems are examined in the following sections.

—Freenet is a *loosely structured* system that uses file and node identifier similarity to produce an estimate of where a file may be located, and a *chain mode propagation* approach to forward queries from node-to-node.

—Chord is a system whose nodes maintain a distributed routing table in the form of an identifier circle on which all nodes are mapped, and an associated *finger table*.

—CAN is a system using an n-dimensional Cartesian coordinate space to implement the distributed location and routing table, whereby each node is responsible for a *zone* in the coordinate space.

—Tapestry (and the similar Pastry and Kademlia) are based on the *plaxton mesh* data structure, which maintains pointers to nodes in the network whose IDs match the elements of a tree-like structure of ID prefixes up to a digit position.

3.4.1. Freenet—A Loosely Structured System. The defining characteristic of loosely structured systems is that the nodes of the peer-to-peer network can produce an estimate (not with certainty) of which node is most likely to store certain content. This affords them the possibility of avoiding blindly broadcasting request messages to all (or a random subset) of their neighbors. Instead, they use a *chain mode propagation* approach, where each node makes a local decision about which node to send the request message to next.

Freenet [Clarke et al. 2000] is a typical, purely decentralized loosely-structured content distribution system. It operates as a self-organizing peer-to-peer network, pooling unused disk space in peer computers to create a collaborative virtual file system. Important features of Freenet include its focus on security, publisher anonymity, deniability, and data replication for availability and performance.

Files in Freenet are identified by unique binary keys. Three types of keys are supported, the simplest is based on applying a hash function on a short descriptive text string that accompanies each file as it is stored in the network by its original owner.

Each Freenet node maintains its own local data store, that it makes available to the network for reading and writing, as well as a dynamic routing table containing the addresses of other nodes and the files they are thought to hold. To search for a file, the user sends a request message specifying the *key* and a timeout (*hops-to-live*) value.

Freenet uses the following types of messages, which all include the node identifier (for loop detection), a hops-to-live value (similar to the Gnutella TTL, see Section 3.3.2), and the source and destination node identifiers:

Data insert. A node inserts new data in the network. A key and the actual data (file) are included.

Data request. A request for a certain file. The key of the file requested is also included.

Data reply. A reply initiated when the requested file is located. The actual file is also included in the reply message.

Data failed. A failure to locate a file. The location (node) of the failure and the reason are also included.

New nodes join the Freenet network by first discovering the address of one or more existing nodes, and then starting to send *Data Insert* messages. To insert a new file in the network, the node first calculates a binary key for the file, and then sends a data insert message to itself. Any node that receives the insert message, first checks to see if the key is already taken. If the key is not found, the node looks up the closest key (in terms of lexicographic distance) in its routing table, and forwards the insert message to the corresponding node. By this mechanism, newly inserted files are placed at nodes possessing files with similar keys.

This continues as long as the hops-to-live limit is not reached. In this way, more than one node will store the new file. At the same time, all the participating nodes will update their routing tables with the new information (this is the mechanism through which the new nodes announce their presence to the rest of the network). If the hops-to-live limit is reached without any key collision, an "all clear" result will be propagated back to the original inserter, informing that the insert was successful.

If the key is found to be taken, the node returns the preexisting file as if a request were made for it. In this way, malicious attempts to supplant existing files by inserting junk will result in the existing files being spread further.

If a node receives a request for a locally-stored file, the search stops and the data is forwarded back to the requestor.

If the node does not store the file that the requestor is looking for, it forwards the request to its neighbor that is most likely to have the file, by searching for the file key in its local routing table that is closest to the one requested. The messages, therefore, form a chain, as they propagate from node-to-node. To avoid huge chains, messages are deleted after passing through a certain number of nodes, based on the hops-to-live value they carry. Nodes also store the ID and other information of the requests they have seen, in order to handle "data reply" and "data failed" messages.

If a node receives a backtracking "data failed" message from a downstream node, it selects the next best node from its routing stack and forwards the request to it. If all nodes in the routing table have been explored in this way and failed, it sends back a "data failed" message to the node from which it originally received the data request message.

If the requested file is eventually found at a certain node, a reply is passed back through each node that forwarded the request to the original node that started the chain. This data reply message will include the actual data, which is cached in all intermediate nodes for future requests. A subsequent request with the same key will be served immediately with the cached data. A request for a similar key will be forwarded to the node that previously provided the data.

To address the problem of obtaining the key that corresponds to a specific file, Freenet recommends the use of a special class of lightweight files called "indirect files". When a real file is inserted, the author also inserts a number of indirect files that are named according to search keywords and contain pointers to the real file. These indirect files differ from normal files in that multiple files with the same key (i.e. search keyword) are permitted to exist, and requests for such keys keep going until a specified number of results is accumulated, instead of stopping at the first file found. The problem of managing the

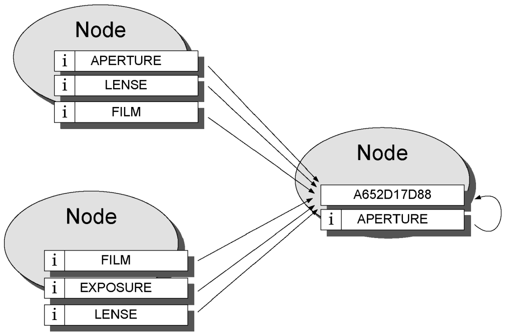

Fig. 4. The use of indirect files in Freenet. The diagram illustrates a regular file with key "A652D17D88", which is supposed to contain a tutorial about photography. The author chose to insert the file itself, and then a set of *indirect* files (marked with an **i**), named according to search keywords she considered relevant. The indirect files are distributed among the nodes of the network; they do not contain the actual document, simply a "pointer" to the location of the regular file containing the document.

large volume of such indirect files remains open. Figure 4 illustrates the use of indirect files.

The following properties of Freenet are a result of its routing and location algorithms:

—Nodes tend to specialize in searching for similar keys over time, as they get queries from other nodes for similar keys.

—Nodes store similar keys over time, due to the caching of files as a result of successful queries.

—Similarity of keys does not reflect similarity of files.

—Routing does not reflect the underlying network topology.

3.4.2. Chord. Chord [Stoica et al. 2001] is a peer-to-peer routing and location infrastructure that performs a mapping of file identifiers onto node identifiers. Data location can be implemented on top of Chord by identifying data items (files) with keys and storing the *(key, data item)* pairs at the node that the keys map to.

In Chord, nodes are also identified by keys. The keys are assigned both to files and nodes by means of a deterministic function, a variant of consistent hashing

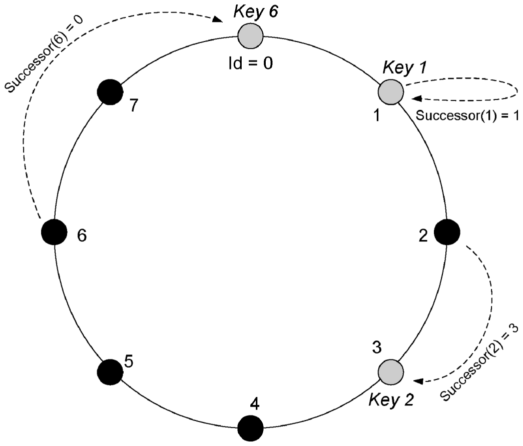

Fig. 5. A Chord identifier circle consisting of the three nodes 0,1 and 3. In this example, key 1 is located at node 1, key 2 at node 3, and key 6 at node 0.

[Karger et al. 1997]. All node identifiers are ordered in an "identifier circle" modulo 2^m (Figure 5 shows an identifier circle with $m = 3$). Key k is assigned to the first node whose identifier is equal to, or follows k, in the identifier space. This node is called the successor node of key k. The use of consistent hashing tends to balance load, as each node receives roughly the same number of keys.

The only routing information required is for each node to be aware of its successor node on the circle. Queries for a given key are passed around the circle via these successor pointers until a node that contains the key is encountered. This is the node the query maps to.

When a new node n joins the network, certain keys previously assigned to n's successor will become assigned to n. When node n leaves the network, all keys assigned to it will be reassigned to its successor. These are the only changes in key assignments that need to take place in order to maintain load balance.

As discussed, only one data element per node needs to be correct for Chord to guarantee correct (though slow) routing of queries. Performance degrades gracefully when routing information becomes out of date due to nodes joining and leaving the system, and availability remains high only as long as nodes fail independently. Since

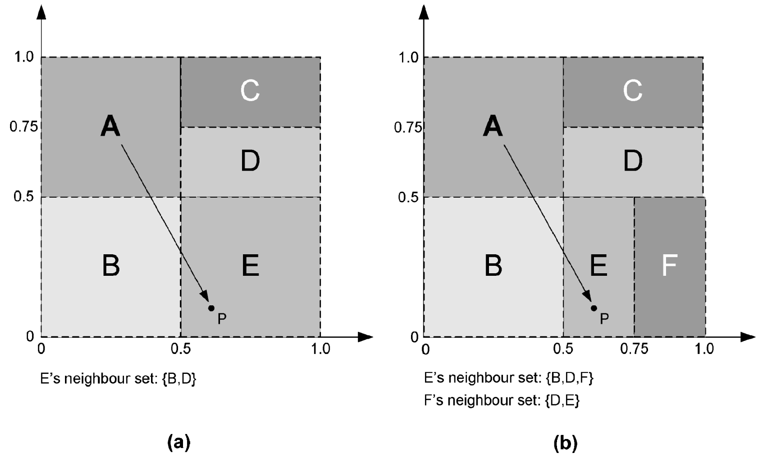

Fig. 6. CAN: (a) Example 2-d $[0, 1] \times [0, 1]$ coordinate space partitioned between 5 CAN nodes; (b) Example 2-d space after node F joins.

the overlay topology is not based on the underlying physical IP network topology, a single failure in the IP network may manifest itself as multiple, scattered link failures in the overlay [Saroiu et al. 2002].

To increase the efficiency of the location mechanism described previously, that may, in the worst case, require traversing all N nodes to find a certain key, Chord maintains additional routing information, in the form of a "finger table". In this table, each entry i points to the successor of node $n + 2i$. For a node n to perform a lookup for key k, the finger table is consulted to identify the highest node n' whose ID is between n and k. If such a node exists, the lookup is repeated starting from n'. Otherwise, the successor of n is returned. Using the finger table, both the amount of routing information maintained by each node and the time required for resolving lookups are $O(logN)$ for an N-node system in the steady state.

Achord [Hazel and Wiley 2002] is proposed as a variant of Chord that provides censorship resistance by limiting each node's knowledge of the network in ways similar to Freenet (see Section 3.4.1).

3.4.3. CAN. The CAN ("Content Addressable Network") [Ratnasamy et al. 2001] is essentially a distributed, Internet-scale hash table that maps file names to their location in the network, by supporting the insertion, lookup, and deletion of *(key,value)* pairs in the table.

Each individual node of the CAN network stores a part (referred to as a "zone") of the hash table, as well as information about a small number of adjacent zones in the table. Requests to insert, lookup, or delete a particular key are routed via intermediate zones to the node that maintains the zone containing the key.

CAN uses a virtual d-dimensional Cartesian coordinate space (see Figure 6) to store *(key K,value V)* pairs. The *zone* of the hash table that a node is responsible for corresponds to a segment of this coordinate space. Any key K is, therefore, deterministically mapped onto a point P in the coordinate space. The (K, V) pair is then stored at the node that is responsible for the zone within which point P lies. For example, in the case of Figure 6(a), a key that maps to coordinate *(0.1,0.2)* would be stored at the node responsible for zone B.

To retrieve the entry corresponding to K, any node can apply the same deterministic function to map K to P, and then retrieve the corresponding value V from the node covering P. Unless P happens to lie in the requesting node's zone, the request must be routed from node-to-node until it reaches the node covering P.

CAN nodes maintain a routing table containing the IP addresses of nodes that hold zones adjoining their own, to enable routing between arbitrary points in space. Intuitively, routing in CAN works by following the straight line path through the Cartesian space from source to destination coordinates. For example, in Figure 6(a), a request from node *A* for a key mapping to point *p* would be routed though nodes *A, B, E*, along the straight line represented by the arrow.

A new node that joins the CAN system is allocated its own portion of the coordinate space by splitting the allocated zone of an existing node in half, as follows:

(1) The new node identifies a node already in CAN network, using a bootstrap mechanism as described in Francis [2000].

(2) Using the CAN routing mechanism, the node randomly chooses a point *P* in the coordinate space and sends a JOIN request to the node covering *P*. The zone is then split, and half of it is assigned to the new node.

(3) The new node builds its routing table with the IP addresses of its new neighbors, and the neighbors of the split zone are also notified to update their routing tables to include the new node.

When nodes gracefully leave CAN, the zones they occupy and the associated hash table entries are explicitly handed over to one of their neighbors. Under normal conditions, a node sends periodic update messages to each of its neighbors reporting its zone coordinates, its list of neighbors, and their zone coordinates. If there is prolonged absence of such an update message, the neighbor nodes realize there has been a failure, and initiate a controlled takeover mechanism. If many of the neighbors of a failed node also fail, an *expanding ring* search mechanism is initiated by one of the neighboring nodes, to identify any functioning nodes outside the failure region.

A list of design improvements is proposed over the basic CAN design described including the use of multi-dimensional coordinate space, or multiple coordinate spaces, for improving network latency and fault tolerance; the employment of more advanced routing metrics, such as connection latency and underlying IP topology, alongside the Cartesian distance between source and destination; allowing multiple nodes to share the same zone, mapping the same key onto different points; and the application of caching and replication techniques [Ratnasamy et al. 2001].

3.4.4. Tapestry. Tapestry [Zhao et al. 2001] supports the location of objects and the routing of messages to objects (or the closest copy of them, if more than one copy exist in the network) in a distributed, self-administering, and fault-tolerant manner, offering system-wide stability by bypassing failed routes and nodes, and rapidly adapting communication topologies to circumstances.

The topology of the network is self-organizing as nodes come and go, and network latencies vary. The routing and location information is distributed among the network nodes; the topology's consistency is checked on-the-fly, and if it is lost due to failures or destroyed, it is easily rebuilt or refreshed.

Tapestry is based on the location and routing mechanisms introduced by Plaxton, Rajamaran and Richa [1997], in which they present the *Plaxton mesh*, a distributed data structure that allows nodes to locate objects and route messages to them across an arbitrarily-sized overlay network, while using routing maps of small and constant size. In the original Plaxton mesh, the nodes can take on the role of servers (where objects are stored), routers (which forward messages), and clients (origins of requests).

Each node maintains a *neighbor map*, as shown in the example in Table IV. The neighbor map has multiple levels, each level l containing pointers to nodes whose ID must be matched with l digits (the x's represent wildcards). Each entry in the neighbor map corresponds to a pointer to the closest node in the network whose ID matches the number in the neighbor map, up to a digit position.

For example, the 5th entry for the 3rd level for node 67493 points to the node

Table IV. The Neighbor Map Held by Tapestry Node With ID 67493

	Level 5	Level 4	Level 3	Level 2	Level 1
Entry 0	07493	x0493	xx093	xxx03	xxxx0
Entry 1	17493	x1493	xx193	xxx13	xxxx1
Entry 2	27493	x2493	xx293	xxx23	xxxx2
Entry 3	37493	x3493	xx393	xxx33	xxxx3
Entry 4	47493	x4493	xx493	xxx43	xxxx4
Entry 5	57493	x5493	xx593	xxx53	xxxx5
Entry 6	**67493**	x6493	xx693	xxx63	xxxx6
Entry 7	77493	x7493	xx793	xxx73	xxxx7
Entry 8	87493	x8493	xx893	xxx83	xxxx8
Entry 9	97493	x9493	xx993	xxx93	xxxx9

Each entry in this table corresponds to a pointer to another node.

closest to 67493 in network distance whose ID ends in ..593. Table IV shows an example neighbor map maintained by a node with ID 67493.

Messages are, therefore, incrementally routed to the destination node digit-by-digit, from the right to the left. Figure 7 shows an example path taken by a message from node with $ID = 67493$, to node $ID = 34567$. The digits are resolved right to left as follows:

$$xxxx7 \rightarrow xxx67 \rightarrow xx567 \rightarrow x4567 \rightarrow 34567$$

The Plaxton mesh uses a *root node* for each object, that serves to provide a guaranteed node from which the object can be located. When an object o is inserted in the network and stored at node n_s, a root node n_r is assigned to it by using a globally consistent deterministic algorithm. A message is then routed from n_s to n_r, storing data in the form of a mapping *(object id o, storer id n_s)* at all nodes along the way. During a location query, messages destined for o are initially routed towards n_r, until a node is encountered containing the *(o, n_s)* location mapping.

The Plaxton mesh offers: (1) simple fault-handling by its potential to route around a single link or node by choosing a node with a similar suffix, and (2) scalability (with the only bottleneck existing at the root nodes). Its limitations include: (1) the need for global knowledge required for assigning and identifying root nodes, and (2) the vulnerability of the root nodes.

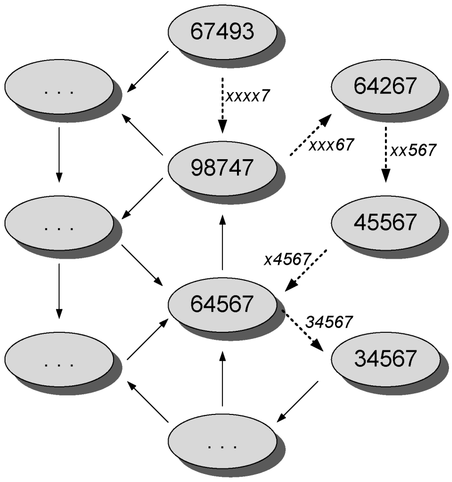

Fig. 7. Tapestry: Plaxton mesh routing example, showing the path taken by a message originating from node 67493, and destined for node 34567 in a Plaxton mesh, using decimal digits of length 5.

The Plaxton mesh assumes a static node population. Tapestry extends its design to adapt it to the transient populations of peer-to-peer networks and provide adaptability, fault tolerance, as well as various optimizations described in Zhao et al. [2001] and outlined below:

—Each node additionally maintains a list of back-pointers, which point to nodes where it is referred to as a neighbor. These are used in dynamic node insertion algorithms to generate the appropriate neighbor maps for new nodes. Dynamic algorithms are employed for node insertion, populating neighbor maps, and notifying neighbors of new node insertions.

—The concept of distance between nodes becomes semantically more flexible, and locations of more than one replica of an object are stored, allowing the application architecture to define how the "closest" node will be interpreted. For example, in the Oceanstore architecture [Kubiatowicz et al. 2000], a "freshness" metric is incorporated in the concept of distance, that is taken into account when finding the closest replica of a document.

—The use of soft-state to maintain cached content, based on the announce/listen approach [Deering 1998], is adopted by Tapestry to detect, circumvent, and recover from failures in routing or object location. Caches are periodically updated by refreshment messages, or purged if no such messages are received. Additionally, the neighbor map is extended to maintain two backup neighbors, in addition to the closest (primary) neighbor. Furthermore, to avoid costly reinsertions of nodes after failures, when a node realizes that a neighbor is unreachable, instead of removing its pointer, it temporarily marks it as invalid in the hope that the failure will be repaired, and, in the meantime, routes messages through alternative paths.

—To avoid the single point of failure that root nodes constitute, Tapestry assigns multiple roots to each object. This enables a trade-off between reliability and redundancy. A distributed algorithm called *surrogate routing* is employed to compute a unique root node for an object in a globally consistent fashion, given the nonstatic set of nodes in the network

—A set of optimizations improve performance by adapting to environment changes. Tapestry nodes tune their neighbor pointers by running refresher threads that update network latency values. Algorithms are implemented to detect query hotspots and offer suggestions as to where additional copies of objects can be placed to significantly improve query response time. A "hotspot cache" is also maintained at each node.

Tapestry is used by several systems such as Oceanstore [Kubiatowicz et al. 2000; Rhea et al. 2001], Mnemosyne [Hand and Roscoe 2002], and Scan [Chen et al. 2000].

Oceanstore further enhances the performance and fault tolerance of Tapestry by applying an additional "introspection layer", where nodes monitor usage patterns, network activity, and resource availability to adapt to regional outages and denial of service attacks.

Pastry [Rowstron and Druschel 2001] is a scheme very similar to Tapestry, differing mainly in its approach to achieving network locality and object replication. It is employed by the PAST [Druschel and Rowstron 2001] large-scale persistent peer-to-peer storage utility.

Finally Kademlia [Mayamounkov and Mazieres 2002] is proposed as an improved XOR-topology-based routing algorithm similar to Tapestry and Pastry, focusing on consistency and performance. It introduces the use of a *concurrence parameter*, that lets users trade bandwidth for better latency and fault recovery.

3.4.5. Comparison, Shortcomings and Evolution of Structured Architectures.

In the previous sections, we have presented characteristic examples of structured peer-to-peer object location and routing systems, and their main characteristics. A comparison of these (and similar) systems can be based on the size of the routing information maintained by each node (the routing table), their search and retrieval performance (measured in number of hops), and the flexibility with which they can adapt to changing network topologies.

It turns out that in their basic design, most of these systems are equivalent in terms of routing table space cost, which is $O(\log N)$, where N is the size of network. Similarly, their performance is again mostly $O(\log N)$, with the exception of CAN, where the performance is given by $O(\frac{d}{4}N^{\frac{1}{d}})$, d being the number of employed dimensions.

Maintaining the routing table in the face of transient node populations is relatively costly for all systems, perhaps more for Chord, as nodes joining or leaving induce changes to all other nodes, and less so in Kademlia which maintains a more flexible routing table. Liben-Nowell et al. [2002a] introduce the notion of the "half-life" of a system as an approach to this issue.

Overall, these systems present relatively equivalent solutions to routing and location in distributed peer-to-peer environments, focusing, however, on different aspects, and prioritizing differently the various design issues (see also Balakrishnan et al. [2003]) that have been adopted by various systems.

One of the major limitations of structured peer-to-peer systems is that they are designed to support exact match lookups. One needs to know the exact identifier (key) of a data item in order to be able to locate the node that stores it. To address this issue, keyword query techniques can be designed on top of exact-match queries, although it is arguable whether such techniques can be efficiently implemented [Lv et al. 2002]. In Garces-Erice et al. [2004] an approach is proposed for augmenting peer-to-peer systems based on distributed hash tables with a mechanism for locating data using incomplete information. The mechanism relies on distributing hierarchically-organized indexes that maintain information about the data stored in nodes. A system to support complex queries over structured peer-to-peer systems is proposed in Harren et al. [2002]. This approach relies on the underlying peer-to-peer system for both indexing and routing, and implements a parallel query processing layer on top of it.

Heleher et al. [2002], furthermore, argue that by creating keys for accessing data items (i.e. by virtualizing the names of the data items) two main problems arise:

—*Locality is Destroyed*. Files originating from a single site are not usually colocated, meaning that opportunities for enhanced browsing, prefetching, and efficient searching are lost. They propose an approach allowing systems to exploit object locality.

—*Useful Application Level Information is Lost*. Data used by applications is often naturally organized in a hierarchical-fashion. The virtualization of the file namespace that takes place as files are represented by keys discards this information.

The ability of structured peer-to-peer systems to maintain the structure required for routing in order to function efficiently when nodes are joining and leaving at a high rate is an open research issue. The resilience of structured peer-to-peer systems in the face of a very transient user population is considered in Lv et al. [2002] and Liben-Nowell et al. [2002b].

Finally, Saroiu et al. [2002] argue that, due to the deterministic nature of the topology of structured systems, it is possible for an attacker to construct a set of node identifiers that, if inserted in the network, could intercept all lookup requests originating from a particular node. There are mechanisms for preventing a peer from deliberately selecting arbitrary identifiers, however, they can probabilistically be circumvented, if enough addresses are selected.

4. CONTENT CACHING, REPLICATION AND MIGRATION

Peer-to-peer content distribution systems rely on the replication of content on more than one node for improving the availability of content, enhancing performance, and resisting censorship attempts. Content replication can be categorized as follows:

Passive Replication. Content replication occurs naturally in peer-to-peer systems as nodes request and copy content from one another. This is referred to as passive replication.

Cache-Based Replication. A form of replication employed in several systems, cache-based replication (e.g. OceanStore, Mojonation, Freenet) is the result of caching copies of content as it passes through nodes in the network. In Freenet [Clarke et al. 2000] for instance, when a search request succeeds in locating a file, the file is transferred through the network node-by-node back to the originator of the request (see also Section 3.4.1). In the process, copies of the file are cached on all intermediated nodes, increasing its availability and location characteristics in future requests.

Active Replication. Active (or *proactive*) content replication and *migration* methods are often employed to improve locality of data, as well as availability and performance.

Instrospective replica management techniques are employed by OceanStore, on top of extensive caching, whereby traffic is observed, and replicas of content objects are created to accommodate demand [Rhea et al. 2001]. OceanStore thus adapts to regional outages and denial of service attacks by migrating data to areas of use, and maintaining sufficiently high levels of data redundancy.

A similar approach is seen in MojoNation, where additional copies of popular data are made by a central services broker that monitors traffic and requests, and are distributed to peer nodes for storage [MojoNation 2003].

Dynamic replica management algorithms are used in Scan to dynamically place a minimum number of replicas, while meeting quality of service and server capacity constraints [Chen et al. 2000]. Such algorithms are designed to satisfy both client latency and server loads by first searching for existing replicas that meet the client latency constraint without being overloaded, and, if unsuccessful, proceeding to place new replicas. Different versions of these algorithms differ in the degree to which replicas are enforced to lie close to the client requesting them.

In proactive replication studies for the Gnutella network [Cohen and Shenker 2001], it was found that the optimal replication scheme for nondeterministic search is to replicate objects such that

$$p \propto \sqrt{q_r},$$

where p is the number of object replicas, and q_r is the object query rate.

By replicating content, data consistency and synchronization issues come up that need to be addressed, especially in systems that allow the deletion and update of content (see also Section 9). Some applications effectively decide to weaken their consistency restrictions in favor of more extensive data replication and higher availability.

Replication is of increased significance to systems employing steganographic file storage techniques for encryption, such as Mnemosyne [Hand and Roscoe 2002], as these systems must ensure that enough copies of the files are available to avoid collisions. Such encryption techniques are further discussed in Section 5.

Replication is more of a challenge for structured systems such as Chord, where the location of objects is directly bound to the object identifiers. In some cases, the use of aliases is employed to allow for replication [Stoica et al. 2001; Saroiu et al. 2002]. In Tapestry, a replication function produces a set of random keys, yielding a set of replica roots at random points in the ID space; a replica function in Chord maps an object's key to the node IDs of the neighbor set of the key's root; in CAN a replica function produces random keys for storing replicas at diverse locations [Wallach 2002]

Finally mechanisms for trading content, discussed in Section 8, essentially result in an additional form of data replication. In FreeHaven, for example, peers frequently trade and exchange parts of documents (shares) with each other, with the sole aim of increasing availability and censorship resistance [Dingledine et al. 2001a].

5. SECURITY

Peer-to-peer content distribution architectures present a particular challenge for providing the levels of availability, privacy, confidentiality, integrity, and authenticity often required, due to their open and autonomous nature. The network nodes must be considered untrusted parties, and no assumptions can be made regarding their behavior.

This Section discusses various approaches that address a number of security issues characteristic of peer-to-peer content distribution systems. We particularly focus on secure storage, secure routing, access control, authentication, and identity management.

5.1. Secure Storage

Various *cryptographic algorithms and protocols* are employed to provide security for content published and stored in peer-to-peer networks.

Self-Certifying Data. Self-certifying data is data whose integrity can be verified by the node retrieving it [Castro et al. 2002]. A node inserting a file in the network calculates a cryptographic hash of the file's content, based on a known hashing function, to produce the file key. When a node retrieves the file using its key, it calculates the same hash function to verify the integrity of the data. Note the requirement for common knowledge of the hashing function by all nodes in the network. The above approach is adopted by the CFS system [Dabek et al. 2001], while PAST uses a similar approach, based on signed files [Druschel and Rowstron 2001]. Note that the method depends on using the hash-derived signature as a file's unique access key.

Information Dispersal. The information dispersal algorithm by Rabin [1989] is widely used. Publius [Waldman et al. 2000], Mnemosyne [Hand and Roscoe 2002], and FreeHaven [Dingledine et al. 2000] employ the algorithm for encoding information to be published in the network. Files are encoded into m blocks, so that any n is sufficient to reassemble the original data ($m < n$). This gives resilience "proportional" to a redundancy factor equal to $\frac{m}{n}$.

Shamir's Secret Sharing Scheme. Shamir's [1979] secret sharing scheme (SHA) is also used in several systems (Publius [Waldman et al. 2000], MojoNation [MojoNation 2003], and PAST [Druschel and Rowstron 2001]). The publisher of the content encrypts a file with a key K, then splits K into l shares, so that any k of them can reproduce K, but $k - 1$ give no hints about K. Each server then encrypts one of the key shares, along with the file block. In order for the file to become inaccessible, at least $(l - k - 1)$ servers containing the key must be shut down.

Anonymous Cryptographic Relays. A mechanism based on *publisher, forwarder, storer*, and *client* peers communicating through anonymous connection layers is employed by PAST [Serjantov 2002]. A *publisher* selects several *forwarders* and sends to them, via an anonymous connection, encrypted shares of a file. The *forwarders*, in turn, select other nodes to act as *storers* for the shares, and forward the shares to them (again via an anonymous connection). Once all shares are stored, the *publisher* destroys them, and announces the file name, together with the list of *forwarders* that were used.

In order to retrieve the content, a *client* will contact the *forwarders*; the *forwarders* will, in turn, contact random servers to act as decryptors for the addresses of the *storers* holding the shares. The *forwarders* will then contact the *storers* requesting the shares. The *storers* will decrypt the shares and send them back to the *client*. The process repeats until enough shares are collected to reconstruct the document.

Forwarders are visible to a potential attacker, but they are less likely to be attacked since they don't store the content, nor the addresses of the nodes storing the content.

Smartcards. The use of *smartcards* for tracking each node's use of remote resources and issuing *digitally signed tickets* was also suggested in the PAST system [Druschel and Rowstron 2001]. This would allow nodes to prove to other nodes that they are operating within their quota. It is argued, however, that it may be impractical to issue smartcards to a very large number of nodes, and that the danger of the smartcard keys being compromised by nodes of the network is considerable [Wallach 2002].

Distributed Steganographic File Systems. A *distributed steganographic file system*, based on Anderson et al. [1998] is implemented by Mnemosyne [Hand and Roscoe 2002]. The main property of a steganographic file system is that encrypted blocks are indistinguishable from a random substrate, so that their presence cannot even be detected. The system is prepared by first writing random data to all blocks, and then files are stored by encrypting their blocks and placing them at pseudo-randomly chosen locations (e.g. by hashing the block number with a randomly chosen key). To avoid collisions, a considerable amount of replication is required.

Erasure Coding. A mechanism based on *erasure coding* for data durability and a Byzantine agreement protocol [Lamport et al. 1982] for consistency and update serialization is implemented by OceanStore [Rhea et al. 2001].

With erasure coding, the data is broken in to blocks and spread over many servers. Only a fraction is needed to reconstruct the original block (similar to the information dispersal algorithm). The objects and their associated fragments are then named using a secure hash over the object contents, giving them globally unique identifiers. This provides data integrity, by ensuring that a recovered file has not been corrupted, since a corrupted file would produce a different identifier. Blocks are dispersed with care to avoid possible correlated failures, picking nodes in different geographic locations or administrative domains, or based on models of historical measurements.

An "inner ring" of servers is set up for each object to provide versioning capabilities. The role of the ring is to maintain a mapping from the original identifier of the object, to the identifier of the most recent version of the object. The ring is kept consistent through a Byzantine agreement protocol that lets $3f + 1$ servers reach an agreement when no more than f are faulty. So the mapping from an active (original) identifier, to the most recent identifier, is fault tolerant.

The inner ring is also responsible for verifying the object's legitimate writers and maintaining a history of the object's updates, thus providing a universal *undo* mechanism, as well as true referential integrity by storing previous versions of objects.

5.2. Secure Routing

The goal of secure routing is to resolve the problem of malicious nodes attempting to corrupt, delete, deny access to, or supply stale copies of object replicas that are transferred between nodes.

The secure routing primitives proposed in Castro et al. [2002] cope with the problems of:

(1) secure assignment of IDs to nodes.

(2) secure maintenance of routing tables.

(3) secure forwarding of messages.

These primitives can be combined with other existing security techniques to yield more robust applications.

5.3. Access Control, Authentication and Identity Management

The issues of access control, authentication, and identity management are often ignored in peer-to-peer content distribution systems. When an access control mechanism is used, it follows the discretionary access control paradigm [Anderson 2001], clearly an insecure approach in the face of untrusted clients.

Within a distributed environment, it is possible for the same physical entity to appear under different identities, particularly in systems with highly transient populations of nodes. This poses a security threat, especially in peer-to-peer systems that employ content replication, or fragmentation schemes over many peers for security and availability (see also Section 5.1), and, therefore, rely on the existence of independent peers with different identities.

This problem is labeled a "Sybil Attack" and analyzed in Douceur [2002], concluding that unless a central certification or identification authority is employed, peer-to-peer systems will be susceptible to this kind of attack. The only proposed alternative is the use of (typically impractical) resource-demanding identification challenges, that assume that the resources of the attacker are limited.

In several systems, the lack of authentication is overcome by the distribution of the keys necessary for accessing content to a subset of privileged users.

Access control lists can also be assigned to objects by their original authors through the use of signed certificates (e.g. in Oceanstore [Kubiatowicz et al. 2000]). All content modifications relating to an object are then verified against the access control list that is assigned to it, and unauthorized updates are ignored.

The issue of peers that often appear with different identities in order to escape the consequences of past actions, and, in particular, the effects of this on the performance of reputation or incentive mechanisms is discussed in Lai et al. [2003].

Finally, it is argued that access control is closely related to intellectual property management and digital rights management issues. Whether—or which—peer-to-peer systems should enforce access control and to what extent, and whether this enforcement should take place at the endpoints of the network, remains an open issue [Daswani et al. 2003].

6. PROVISIONS FOR ANONYMITY

Anonymity is the main focus of several peer-to-peer based infrastructures and content distribution systems targeting privacy, confidentiality, and censorship resistance. In content distribution systems anonymity can refer to [Dingledine et al. 2001a]:

—the author (or publisher) of the content,
—the identity of a node storing the content,
—the identity and details of the content itself, and
—the details of a query for retrieval of the content.

This section describes the main approaches currently adopted for providing anonymity.

Disassociation of Content Source and Requestor. Freenet [Clarke et al. 2000] is a peer-to-peer content distribution system specifically targeted towards providing anonymity to its users by making it infeasible to discover the true origin or destination of a file passing through its network, and difficult for a node operator to determine or be held responsible for the actual physical content of their own node. It employs a mechanism whereby, when a search for a file succeeds, the file is passed from the node that holds it to the originator of the request through every node that forwarded the search request. In order to achieve anonymity, any node along this path can unilaterally decide to claim itself or another arbitrarily chosen node as the data source. In this way, there is no connection between the requestor of the file and the actual source node.

As a further measure, the initial value of the hops-to-live counter associated with each search message is randomly chosen in order to obscure the distance of the message from the originator node. More details about the location and routing mechanism employed by Freenet are discussed in Section 3.4.1.

Anonymous Connection Layers. Content distribution systems that seek to provide anonymity often employ infrastructures for providing anonymous connection layers, such as *onion routing* [Goldschlag et al. 1999], *mix networks* [Chaum 1981; Berthold et al. 1998] or the Freedom anonymity system [Freedom 2003]. For instance, the anonymizing, censorship-resistant system proposed in Serjantov [2002] splits documents to be published into encrypted shares, and uses an anonymizing layer of nodes (which it refers to as "forwarders") to pick nodes on which to store the shares, construct "onions" around them (as in onion routing), and anonymously forward them, including their anonymous return addresses, while the original documents are destroyed. FreeHaven [Dingledine et al. 2000] is a similar system built on top of anonymous remailers for providing pseudonyms and communication channels.

The Tarzan system [Freedman et al. 2002] is a completely decentralized anonymizing network layer infrastructure that builds anonymous IP tunnels between an open-ended set of peers. By using the Tarzan infrastructure, a client can communicate with a server without anybody being able to determine their identity.

The Tarzan anonymizing layer is completely decentralized and transparent both to clients and servers, and it involves sequences of mix relays chosen from a large pool of volunteer participant nodes. Packets are routed through a tunnel of randomly chosen Tarzan peers using mix-style layered encryption, similar to onion routing. The two ends of the tunnel are a Tarzan node, running a client application, and a Tarzan node, running a network address translator. The latter forwards the traffic to the ultimate destination, an ordinary Internet server. A suggested policy to reduce the risk of attack to the system is for tunnels to contain peers from different jurisdictions or organizations, even if performance is sacrificed. Crowds [Reiter and Rubin 1999] is a system similar to Tarzan, its main difference being that, in Tarzan, the data is encrypted in the tunnels, while in Crowds it is not.

Censorship Resistant Lookup. A censorship resistant design based on the Chord lookup service is proposed in the Achord system [Hazel and Wiley 2002]. Achord provides censorship resistance by focusing on publisher, storer, and retriever anonymity, and by making it difficult for a node to voluntarily assume responsibility for a certain document. The design of Chord is carefully varied so that these anonymity and censorship resistance properties are achieved without altering its main operation. In particular, the Chord algorithm is modified so that the node identification information is suppressed as the successor nodes are located. More details about the Chord Distributed Object Location and Routing system can be found in Section 3.4.2.

7. PROVISIONS FOR DENIABILITY

Deniability in a peer-to-peer content distribution system refers to each user's ability to deny knowledge of content stored in their node. As a consequence, users cannot be held responsible for the content stored in their node, or for actions that have been carried out by their node as part of it's operation in the peer-to-peer network.

A deniable peer-to-peer system can be used without fear of censorship or repercussions for dealing with particular content, increasing the degree of freedom it provides to its users. On the other hand, it makes the exchange of illegal content less risky and, therefore easier, as there is less control over the operation of content stored at each node of the network.

Deniability can apply both to content being stored and content being transferred.

Deniability of Stored Content. This feature is offered by systems that store encrypted shares of files and no keys for them, and, therefore, cannot know the content of the files whose shares they are storing (examples are Publius [Waldman et al. 2000], Oceanstore [Kubiatowicz et al. 2000], and PAST [Druschel and Rowstron 2001; Serjantov 2002]). Similarly, systems that implement distributed steganographic storage systems (such as Mnemosyne [Hand and Roscoe 2002]) offer deniability through the fact that blocks of files that are written on a peer node's file system are undetectable (cannot be directly searched).

Deniability of Content in Transit. This feature can be offered through the use of anonymous connection layers incorporated in systems such as PAST (see also Section 6). It makes it possible to deny that a request from a client node n_c had anything to do with a share arriving at n_c some time later.

In a different approach with the same goal, Freenet [Clarke et al. 2000] offers deniability by making it infeasible to discover the true origin of a file passing through the network (see Section 3.4.1 for a description of the Freenet location algorithm). Furthermore, as files passing through the network are transparently cached at all nodes encountered, it is difficult for a node operator to determine or be held responsible for the actual physical content of their own node.

Additional measures for deniability can be incorporated by inhibiting the use of traffic analysis for concluding where a file was read from. For example, in Mnemosyne [Hand and Roscoe 2002], during the retrieval of a file, more nodes than those needed will be contacted and more files will be retrieved so that the actual file targeted will be disguised.

It should be noted that structured systems are apparently nondeniable, as the identifiers of the files stored at the nodes are bound to the node addresses; if a file is known to exist in the network, its location, and, therefore, the identity of the node that stores it, is also known. On the other hand, the owner of the node has not necessarily requested the file, and, in any event, has no control over whether the file will be stored in their node, and, in this sense, cannot be held responsible for it.

8. INCENTIVE MECHANISMS AND ACCOUNTABILITY

The operation, performance and availability of an uncontrolled decentralized peer-to-peer system relies to a large extent on the voluntary participation of its users. It is, therefore, necessary to employ mechanisms that provide incentives and stimulate cooperative behavior between the users, as well as some notion of accountability for actions performed.

In the absence of such provisions, the results can range from significant degradation of performance to variable and unpredictable availability of resources, or even to complete collapse.

An example of uncooperative behavior is the so-called "free-rider" effect, where users only consume resources without contributing any. This can be interpreted as a manifestation of the "Tragedy of the Commons" [Harding 1968], which argues that people tend to abuse shared resources that they do not have to pay for in some way.

Providing incentives and accountability in peer-to-peer networks with transient populations of users, where it is hard to identify peers and obtain information about their past behavior in order to predict their future performance, can be a particularly challenging task, especially due to the absence of a ubiquitous, effective, robust, and secure system for making and accepting anonymous micropayments.

Two general categories of solutions are proposed:

—Trust-based Incentive Mechanisms. Trust is a straightforward incentive for cooperation, in which one engages in a transaction based on whether he/she trusts the other party. Reputation mechanisms belong in this category.

—Trade-based Incentive Mechanisms. In trade-based incentive mechanisms, one party offering some service to another is explicitly remunerated, either directly or indirectly. This category is mainly represented by various micropayment mechanisms and resource trading schemes.

8.1. Reputation Mechanisms

Online reputation management systems can be described as large-scale "online word-of-mouth communities" in which individuals share opinions about other individuals.

Centralized reputation systems (such as the one found in eBay) are successful to a large extent because people trust the reputation information presented by them [Dellarocas 2001]. In a peer-to-peer network, however, there is no single, recognizable organization or entity to maintain and distribute reputation information. As a result, reputation information must be distributed throughout the network, and hosted on many different nodes.

The main goal of a peer-to-peer reputation mechanism is to take the reputation information that is locally generated as a result of an interaction between peers, and spread it throughout the network to produce a global reputation rating for the network nodes. In the process, such reputation information must be kept secure and available. Various complex reputation management mechanisms have been developed to address these challenging tasks.

Kamvar et al. [2003] propose the *EigenTrust* algorithm, which produces global reputation ratings for users based on their history of uploads. These ratings can then be used by other peers to decide where to download files from. The global reputation values are computed from the local reputation values assigned to a peer by other peers, weighted by the global reputation of the assigning peers. This approach was found to reduce the number of rogue files on the network.

Gupta et al. [2003] present a partially centralized mechanism using *reputation computation agents* and data encryption, where the reputation values are calculated, encrypted, and stored locally using a reputation computation agent. They propose two different schemes for calculating reputation values, a credit/debit scheme, and a credit only scheme.

Xiong and Liu [2002] present a feedback-based reputation mechanism where a scalar trust value for a peer is computed based on a figure of the satisfaction received by other peers, the total number of interactions, and a balancing factor to counteract reports by malicious peers. The reputation information is distributed in the network so that each peer maintains a portion of the total trust data.

Various simple (and to some extent naive) approaches calculate a user participation level [Kazaa 2003] and reward peers with higher participation by giving higher priority to their requests, or by reducing their waiting times in file transfer queues. In a more abstract approach [Ramaswamy and Liu 2003], utility functions, based on the amount and popularity of content stored by the peer, are used to estimate a figure for their usefulness. Such mechanisms are generally easy to subvert by malicious users.

Finally, distributed auditing mechanisms have also been devised to apply peer-pressure to misbehaving nodes [Wallach 2002]. Nodes can consume resources to compare the logs of another node, chosen at random, with the logs of every node with which it is sharing information. The system appears to provide strong disincentives to cheaters at a reasonable cost.

At the development level, peer-to-peer frameworks such as Sun's JXTA [Jxta 2003] include provisions for incorporating peer-monitoring hooks in a distributed content distribution environment, that could be used for collecting transaction information and building centralized or distributed reputation databases.

Some potential problems with distributed reputation management need to be addressed. For example, an entity acquiring reputation in one context where it has functioned well, may spend it in another context for malicious purposes. An immediate extension of the above scenario is the possibility of using reputation as currency, exchanging it for content or services.

A discussion of accountability and reputation can also be found in Dingledine et al. [2001b].

8.2. Micropayments Mechanisms

A variety of both centralized [MojoNation 2003; Ioannidis et al. 2002] and decentralized [Vishnimurthy et al. 2003; Yu and Singh 2003] currency-based micropayment and credit systems have been deployed in peer-to-peer networks.

A typical example is the MojoNation system, where a currency (the "Mojo") is gained by offering disk space, bandwidth, or CPU cycles, and used to obtain access to distributed storage space. This system employs a scheme based on trusted third parties to ensure honest transactions between peers.

More complex systems offer more advanced capabilities to users, for example, allowing them to negotiate prices and perform auctions for services rendered (e.g. Vishnimurthy et al. [2003])

It is, however, claimed in Buragohain et al. [2003] that monetary schemes, although providing clear economic models, may be impractical for most applications.

8.3. Resource Trading Schemes

Various decentralized resource trading and exchange schemes have been proposed [Anagnostakis and Greenwald

2004; Cohen 2003; Cooper and Garcia-Molina 2002; Dingledine et al. 2000].

Anagnostakis and Greenwald [2004] proposed an extension to accommodate transitive chains of peer transactions, instead of simple one-to-one exchanges.

A system in which each node publishes and digitally signs logs containing lists of files it stores on behalf of remote nodes and lists of files that other nodes are storing on its behalf is proposed in Wallach [2002]. Through the use of these lists it can be ensured that no node is using more resources than it is providing, and a balanced system is achieved.

A flexible algorithm for trading content is proposed by Cooper and Garcia-Molina [2002]. Peers make local decisions about how much data to trade, what resources to contribute and how many trades to make. This algorithm ensures fairness while trying to maximize content preservation by choosing the most appropriate policies for deed trading, advertising, remote site selection, and bidding.

Another example of a purely decentralized reputation system based on resource trading exist seen in FreeHaven [Dingledine et al. 2000]. The FreeHaven nodes form contracts to store each other's material for a certain period of time. By successfully fulfilling a contract, a node increases its reputation. Contracts are formed whenever new data is inserted in the network, or swapped between peers. A trading handshake protocol is employed. It includes "receipts" for resources traded, and the use of "buddy" peer nodes that record the transactions carried out by other nodes. Each node, therefore, maintains a database containing the perceived reputation of others. Nodes periodically check whether other nodes have dropped data earlier than they were supposed to, and decrease the level of trust of these peers. In order for nodes to leave the network without decreasing their reputation value, they must first swap their documents for shorter-lived ones, and wait until they expire.

An additional benefit of trading content is that it offers a way of actively replicating it, and hence, making it more available and less likely to be lost due to failures. Digital libraries may collaborate with one another to provide content preservation by storing each other's material. Systems such as OceanStore [Kubiatowicz et al. 2000] and Intermemory [Chen et al. 1999] employ this idea.

Finally, it is argued in Chun et al. [2003] that such resource trading models are particularly suitable for bootstrapping peer-to-peer economies in the first period of their operation.

9. RESOURCE MANAGEMENT CAPABILITIES

The resources that peer-to-peer content distribution systems typically deal with are content (files), storage (disk space), and transmission capacity (bandwidth). Obviously inserting, locating (searching), and retrieving content are the minimum operations that any system is required to perform. Additional resource management facilities may include removing or updating content, maintaining previous versions of updated content, managing storage space, and setting bandwidth limits. Different peer-to-peer content distribution system allow different levels of resource management, as now described.

Content Deletion and Update. Deleting and updating content are not straightforward operations in a peer-to-peer environment if correct synchronization is to be maintained. Systems such as MojoNation [MojoNation 2003] use immutable files, which cannot be updated. The only way to update the content is to post a new version of the document under a different name. PAST [Druschel and Rowstron 2001] is similar in this respect. It does offer a delete functionality for reclaiming the disc space occupied by a file, but it does not guarantee that the file will no longer be available anywhere in the network. FreeHaven [Dingledine et al. 2000] also forbids unpublishing, or revocation of documents, but mainly for reasons of security and robustness to attacks.

Both deletion and update of content are offered by Publius [Waldman et al. 2000], which is, however, based on a static set of servers that store the content. Files are published along with a password file that ensures that only the publisher will be able to delete or modify the content.

Content Expiration. Document expiration dates are effectively introduced in FreeHaven [Dingledine et al. 2000] through the use of contracts with different durations, as described in Section 8.

Content Versioning. A more sophisticated approach is employed by OceanStore [Rhea et al. 2001], which offers a version-based archival storage system. Versions of documents are stored in permanent read-only form, encoded with erasure codes, and spread over hundreds of servers. OceanStore maintains active maps of document IDs, which redirect to the most recent version of a document, providing both an update history and an undo mechanism.

Directory Structure. An entire distributed directory structure system including directories and Unix-like inodes [Bach 1986] is built on top of files by Mnemosyne [Hand and Roscoe 2002]. Directories are optionally used to aggregate files sharing a common key, serving as a mnemonic service for a set of file names.

Content Searching. Searching facilities may also vary in their degree of functionality and performance. Unstructured systems such as Gnutella, Kazaa, and FreeHaven offer keyword-search mechanisms that are convenient, but do not scale well (see also our discussion in 3.3.4). In structured systems, searches are very efficient, but can only be based on file identifiers. The problem of providing keyword search facilities on top of exact-match queries is open. One solution proposed in Freenet is the use of indirect files, published along with regular files and pointing to them, which are named according to relevant search keywords. The problem of managing the large volume of such files, however, also remains unresolved [Clarke et al. 2000].

Storage and Bandwidth Management. Managing the storage space available to the peers also varies between systems. Usually the amount of disk space contributed can be specified by each peer independently. In systems such as MojoNation users contribute storage in exchange for economic or other compensation. PAST [Druschel and Rowstron 2001] uses a secure quota system, where users are either assigned a fixed quota of storage they can use, or they can use as much as they contribute on their node. This system can be optionally extended to incorporate the use of smartcards.

In order to protect from denial of service attacks, apart from enforcing a published size quota to users, the idea of "hash cash" is also employed (e.g. in Publius [Waldman et al. 2000]). A publisher is requested to carry out computational work to solve a mathematical problem before being allowed to publish a certain amount of content. Alternatively, a per-node rate limiter can be used, as is the case for example in Mnemosyne [Hand and Roscoe 2002]. Bandwidth management is also available in systems such as Kademlia [Mayamounkov and Mazieres 2002], which allows the users to trade bandwidth for latency and fault recovery.

10. SEMANTIC GROUPING OF INFORMATION

The topic of semantic grouping and organization of content and information within peer-to-peer networks has attracted considerable research attention lately [Zhou et al. 2003; Ayyasamy et al. 2003; Castano et al. 2003; Broekstra et al. 2003], driven to a large extent by the recent advances in the field of knowledge management.

Khambatti et al. [2003] introduce the notion of interest-based "peer communities", similar to real human communities, in which membership depends on the relationship between peers that share common interests. The authors argue—and show through their experiments—that such communities can be used to make searching more efficient and produce better quality of results. The grouping is implemented through sets of *attributes* that the peers choose to claim, and communities are formed between peers that share similar attributes. As an extension of this notion, a peer-to-peer community-based trust model is introduced in Khambatti et al. [2004], where links between peers that trust each other are used to form dynamic coalitions between different communities.

The *semantic overlay clustering* approach, based on partially-centralized (super-peer) networks [Loeser et al. 2003] aims at creating logical layers above the physical network topology, by matching semantic information provided by peers to clusters of nodes based on super-peers. Clusters of nodes with similar semantic profiles are, therefore, formed based on some matching engine and clustering policy. Each cluster contains a set of peers, together with their corresponding super-peer.

Bonifacio et al. [2002] propose a distributed knowledge management architecture consisting of *knowledge nodes* (peers that own local knowledge), that cooperate through a set of services. Each node can act as a *seeker* (allowing the user to search for information and forwarding queries), as a *provider* (accepting and resolving queries and returning results), or both. Searching can be either semantic (based on a context matching algorithm), or textual (based on keywords). Groups of nodes have the ability to form *federations*, similar to social aggregations, by agreeing to be considered as a sole entity by other peers when they request certain semantic search services.

A peer-to-peer architecture based on semantic grouping, context, and peer profiles applied in the area of eLearning is proposed in Hummel et al. [2003].

From a different perspective, a number of different mechanisms are proposed for constructing efficient overlay network topologies by taking into account the locality and network proximity of peer computers to decrease the communication cost [Zhang et al. 2004; Zhao et al. 2002; Castro et al. 2002].

11. CONCLUSIONS

Peer-to-peer systems are distributed systems consisting of interconnected nodes, able to self-organize into network topologies with the purpose of sharing resources such as content, CPU cycles, storage, and bandwidth. Content distribution, a prominent application area of peer-to-peer systems, is based on systems and infrastructures designed for sharing digital media and other data between users.

Peer-to-peer content distribution systems range from relatively simple direct file sharing applications, to more sophisticated systems that create a distributed storage infrastructure for securely and efficiently publishing, organizing, indexing, searching, updating, and retrieving data.

We performed this study of peer-to-peer content distribution systems and infrastructures by identifying the feature space of their nonfunction properties, and determining the way in which these nonfunctional properties depend on, and are affected by various design features. The main nonfunctional characteristics we identified include provisions for security, anonymity, fairness, increased scalability, and performance, as well as resource management, and organization capabilities.

One of the central characteristics of peer-to-peer systems—which reflects upon issues of performance, availability, and scalability—is their ability to function, scale, and self-organize in the presence of a highly transient population of nodes and network and computer failures, without the need for a central server administration. This characteristic is mainly attributed to the network organization and location and routing algorithms. Two general categories of systems can be identified in this respect: the *unstructured systems* and the *structured* (or DHT-based) systems. The rationale behind structured systems often was to conquer the severe scalability and performance problems of unstructured ones. However, recent research developments in both structured and unstructured systems, renders them both as complementary and acceptable solutions.

Research on other aspects and properties of peer-to-peer content distribution systems has taken advantage of knowledge and solutions from numerous scientific fields. Characteristic examples include research on the application of reputation systems in peer-to-peer environments, the semantic organization of information, as well as the use of cryptographic techniques for the protection of data, both stored and in transit through the network.

Finally, as peer-to-peer technologies are still evolving, there are a multitude of open research problems, directions, and opportunities, including, but not limited to:

—The design of new distributed object location, routing and distributed hash table data structures and algorithms for maximizing performance, security and scalability, both in structured and unstructured network architectures.

—The study of more efficient security, anonymity, and censorship resistance schemes. These features will be critical to the future of peer-to-peer systems and their adoption for increasingly more sensitive applications.

—The semantic grouping of information in peer-to-peer networks. This direction that has a lot in common with efforts in the semantic Web domain.

—The design of incentive mechanisms and reputation systems that will stimulate the cooperative behavior between the users, and make the overall operation of peer-to-peer networks more fair.

—The convergence of Grid and Peer-to-Peer systems. Research in this direction aims to combine the benefits of the established field of distributed computing (including interoperability standards) with the merits of new peer-to-peer architectures.

ACKNOWLEDGMENTS

The authors would like to thank the anonymous reviewers for their insightful comments and suggestions.

REFERENCES

AGRE, P. 2003. P2p and the promise of internet equality. *Comm. ACM 46*, 2 (Feb.), 39–42.

ANAGNOSTAKIS, K. AND GREENWALD, M. 2004. Exchange-based incentive mechanisms for peer-to-peer file sharing. To Appear *in the Proceedings of the 24th International Conference on Distributed Computing (ICDCS04).*

ANDERSON, R. 2001. *Security Engineering: A Guide to Building Dependable Distributed Systems*. John Wiley & Sons, New York.

ANDERSON, R., NEEDHAM, R., AND SHAMIR, A. 1998. The steganographic file system. In *Proceedings of International Workshop on Information Hiding* (IWIH) .

AYYASAMY, S., PATEL, C., AND LEE, Y. 2003. Semantic web services and dht-based peer-to-peer networks: A new symbiotic relationship. In *Proceedings of the 1st Workshop on Semantics in Peer-to-Peer and Grid Computing at the 12th International World Wide Web Conference*. Budapest, Hungary.

BACH, M. 1986. *The Design of the UNIX Operating System*. Prentice-Hall.

BALAKRISHNAN, H., KAASHOEK, M., KARGER, D., R, M., AND STOICA, I. 2003. Looking up data in p2p systems. *Comm. ACM 46*, 2 (Feb.), 43–48.

BERNERS-LEE, T., HENDLER, J., AND LASSILA, O. 2001. The semantic web. *Scientific American*.

BERNSTEIN, P., GIUNCHIGLIA, F., KEMENTSIETSIDIS, A., MYLOPOULOS, J., SERAFINI, L., AND ZAIHRAYEU, I. 2002. Data management for peer-to-peer computing: A vision. In *Proceedings of the Workshop on the Web and Databases (WebDB'02).*

BERTHOLD, O., FEDERRATH, H., AND KOPSELL, S. 1998. Web mixes: A system for anonymous and unobservable internet access. In *Proceedings of the Workshop on Design Issues in Anonymity and Unobservability*. Berkeley, CA.

BONIFACIO, M., CUEL, R., MAMELI, G., AND NORI, M. 2002. A peer-to-peer architecture for distributed knowledge management. In *Proceedings of the 3rd International Symposium on Multi-Agent Systems, Large Complex Systems, and E-Businesses (MALCEB'02).*

BROEKSTRA, J., EHRIG, M., HAASE, P., VAN HARMELEN, F., KAMPMAN, A., SABOU, M., SIEBES, R., STAAB, S., STUCKENSCHMIDT, H., AND TEMPICH, C. 2003. A metadata model for semantics-based peer-to-peer systems. In *Proceedings of the 1st Workshop on Semantics in Peer-to-Peer and Grid Computing at the 12th International World Wide Web Conference*. Budapest, Hungary.

BURAGOHAIN, C., AGRAWAL, D., AND SURI, S. 2003. A game theoretic framework for incentives in p2p systems. In *Proceedings of the 3rd International Conference on Peer-to-Peer Computing*.

CASTANO, S., FERRARA, A., MONTANELLI, S., PAGANI, E., AND ROSSI, G. 2003. Ontology-addressable contents in p2p networks. In *Proceedings of the 1st Workshop on Semantics in Peer-to-Peer and Grid Computing at the 12th International World Wide Web Conference*. Budapest, Hungary.

CASTRO, M., DRUSCHEL, P., GANESH, A., A, R., AND WALLACH, D. 2002. Secure routing for structured peer-to-peer overlay networks. In *Proceedings of the 5th Usenix Symposium on Operating Systems*. Boston, MA.

CASTRO, M., DRUSCHEL, P., KERMARREE, A.-M., AND ROWSTRON, A. 2002. Scribe: A large-scale and decentralized application-level multicast infrastructure. *IEEE J. Select. Areas Comm. 20*, 8 (Oct.).

CASTRO, M., DRUSCHEL, P., YC, H., AND ROWSTRON, A. 2002. Exploiting network proximity in peer-topeer overlay networks. In *Proceedings of the International Workshop on Future Directions in Distributed Computing (FuDiCo'02)*.

CHAUM, D. 1981. Untraceable electronic mail, return addresses and digital pseudonyms. *Comm. ACM 24*, 2, 84–88.

CHAWATHE, Y., RATNASAMY, S., BRESLAU, L., LANHAM, N., AND SHENKER, S. 2003. Making Gnutella-like p2p systems scalable. In *Proceedings of the 2003 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*. Karlsruhe, Germany, 407–418.

CHEN, Y., EDLER, J., GOLDBERG, A., GOTTLIEB, A., SOBTI, S., AND YANILOS, P. 1999. A prototype implementation of archival intermemory. In *Proceedings of the 4th ACM Conference on Digital Libraries*. Berkeley, CA.

CHEN, Y., KATZ, R., AND KUBIATOWICZ, J. 2000. Scan: A dynamic, scalable and efficient content distribution network. In *Proceedings of International Conference on Pervasive Computing*.

CHUN, B., FU, Y., AND VAHDAT, A. 2003. Bootstrapping a distributed computational economy with peer-to-peer bartering. In *Proceedings of the 1st Workshop on Economics of Peer-to-Peer Systems*.

CLAKE, I., HONG, T., SANBERG, O., AND WILEY, B. 2002. Protecting free expression online with Freenet. *IEEE Internet Comput. 6*, 1 (Jan.-Feb.), 40–49.

CLARKE, I., SANDBERG, O., AND WILEY, B. 2000. Freenet: A distributed anonymous information storage and /etrieval system. In *Proceedings of the Workshop on Design Issues in Anonymity and Unobservability*. Berkeley, CA.

COHEN, B. 2003. Incentives build robustness in bitorrent. In *Proceedings of the 1st Workshop on Economics of Peer-to-Peer Systems*.

COHEN, E. AND SHENKER, S. 2001. Optimal replication in random search networks. Preprint.

COOPER, B. AND GARCIA-MOLINA, H. 2002. Peer-to-peer resource trading in a reliable distributed system. In *Proceedings of the 1st International Workshop on Peer-to-Peer Systems (IPTPS '02)*. MIT Faculty Club, Cambridge, MA.

CRESPO, A. AND GARCIA-MOLINA, H. 2002. Routing indices for peer-to-peer systems. In *Proceedings of the 22nd International Conference on Distributed Computing Systems (ICDCS'02)*. Vienna, Autria.

DABEK, F., KAASHOEK, M., KARGER, D., MORRIS, R., AND STOICA, I. 2001. Wide-area cooperative storage with CFS. In *Proceedings of the ACM SOSP'01 Conference*. Banff, Canada.

DASWANI, N., GARCIA-MOLINA, H., AND YANG, B. 2003. Open problems in data-sharing peer-to-peer systems. In *Proceedings of the 9th International Conference on Database Theory*. Siena, Italy.

DEERING, S. 1998. Host extensions for IP multicasting. Tech. Rep. RFC-1112, IETF, (Aug.). SRI International, Menlo Park, CA.

DELLAROCAS, C. 2001. Analyzing the economic efficiency of ebay-like online reputation mechanisms. In *Proceedings of the 3rd ACM Conference on Electronic Commerce*. Tampa, FL.

DINGLEDINE, R., FREEDMAN, M., AND MOLNAR, D. 2000. The FreeHaven project: Distributed anonymous storage service. In *Workshop on Design Issues in Anonymity and Unobservability*. 67–95.

DINGLEDINE, R., FREEDMAN, M., AND MOLNAR, D. 2001a. *Peer-to-peer: Harnessing the Power of Disruptive Technology*, 1st Ed. O'Reilly (Chapter 1. A network of peers: Peer-to-peer models through the history of the Internet, 3–20)

DINGLEDINE, R., FREEDMAN, M., AND MOLNAR, D. 2001b. *Peer-to-peer: Harnessing the power of disruptive technology*, 1st Ed. O'Reilly (Chapter 16. Accountability, 271–340)

DOUCEUR, J. 2002. The Sybill attack. In *Proceedings of the 1st International Workshop on Peer-to-Peer Systems (IPTPS'02)*. MIT Faculty Club, Cambridge, MA.

DRUSCHEL, P. AND ROWSTRON, A. 2001. Past: A large-scale, persistent peer-to-peer storage utility. In *Proceedings of the Eighth Workshop on Hot Topics in Operating Systems*.

FastTrack Accessed on-line 2003. The FastTrack web site. http://www.fasttrack.nu.

FOSTER, I. 2000. Internet computing and the emerging grid. *Nature Web Matters*.

FOSTER, I. AND IAMNITCHI, A. 2003. On death, taxes, and the convergence of peer-to-peer and grid computing. In *Proceedings of the 2nd International Workshop on Peer-to-Peer Systems (IPTPS'03)*. Berkley, CA.

FOSTER, I., KESSELMAN, C., AND TUECKE, S. 2001. The anatomy of the grid. *Intl. J. Supercomput. Appl.*

FRANCIS, P. 2000. Yoid: Extending the internet multicast architecture. Unpublished Paper, available on-line at http://www.aciri.org/yoid/docs/index.html.

FREEDMAN, M., SIT, E., CATES, J., AND MORRIS, R. 2002. Introducing tarzan, a peer-to-peer anonymizing network layer. In *Proceedings of the 1st International Workshop on Peer-to-Peer Systems (IPTPS'02)*. MIT Faculty Club, Cambridge, MA.

Freedom 2003. The Freedom anonymity system web site. http://www.freedom.net.

GARCES-ERICE, L., FELBER, P., BIERSACK, E., URVOY-KELLER, G., AND ROSS, K. 2004. Data indexing in peer-to-peer dht networks. In *Proceedings of the 24th IEEE International Conference on Distributed Computing Systems (ICDCS)*. Tokyo, Japan, 200–208.

GenomeAtHome 2003. The genome@home project web site. http://genomeathome.stanford.edu/.

Gnutella 2003. The Gnutella web site: http://gnutella.wego.com.

GOLDSCHLAG, D., REED, M., AND SYVERSON, P. 1999. Onion routing for anonymous and private internet connections. *Comm. ACM 42*, 2, 39–41.

Groove 2003. The Groove web site. http://www.groove.net.

GUPTA, M., JUDGE, P., AND AMMAR, M. 2003. A reputation system for peer-to-peer networks. In *Proceedings of the NOSSDAV'03 Conference*. Monterey, CA.

HALEVY, A., IVES, Z., MORK, P., AND TATARINOV, I. 2003. Piazza: Data management infrastructure for semantic web applications. In *Proceedings of the 12th International Conference on World Wide Web*. Budapest, Hungary, 556–567.

HAND, S. AND ROSCOE, T. 2002. Mnemosyne: Peer-to-peer steganographic storage. In *Proceedings of the 1st International Workshop on Peer-to-Peer Systems (IPTPS'02)*. MIT Faculty Club, Cambridge, MA.

HARDING, G. 1968. The tragedy of the commons. *Science 162*, 1243–1248.

HARREN, M., HELLERSTEIN, J., HUEBSCH, R., LOO, B., SHENKER, S., AND STOICA, I. 2002. Complex queries in dht-based peer-to-peer networks. In *Proceedings of the 1st International Workshop on Peer-to-Peer Systems (IPTPS '02)*. MIT Faculty Club, Cambridge, MA.

HAZEL, S. AND WILEY, B. 2002. Achord: A variant of the Chord lookup service for use in censorship resistant peer-to-peerpublishing systems. In *Proceedings of the 1st International Workshop on Peer-to-Peer Systems (IPTPS '02)*. MIT Faculty Club, Cambridge, MA.

HELEHER, P., BHATTACHARJEE, B., AND SILAGHI, B. 2002. Are vitrualized overlay networks too much of a good thing? In *Proceedings of the 1st International Workshop on Peer-to-Peer Systems (IPTPS '02)*. MIT Faculty Club, Cambridge, MA.

HUEBSCH, R., HELLERSTEIN, J., LANHAM, N., AND THAU LOO, B. 2003. Querying the internet with pier. In *Proceedings of the 29th VLDB Conference*. Berlin, Germany.

HUMMEL, K., KOTSIS, G., AND KOPECNY, R. 2003. Peer profile driven group support for mobile learning teams. In *Proceedings of the CATE/IASTED Conference*. Rhodes, Greece.

IOANNIDIS, J., IOANNIDIS, S., KEROMYTIS, A., AND PREVELAKIS, V. 2002. Fileteller: Paying and getting paid for file storage. In *Proceedings of the Sixth International Conference on Financial Cryptography*.

Jabber 2003. The Jabber web site. http://www.jabber.org/.

JANAKIRAMAN, R., WALDVOGEL, M., AND ZHANG, Q. 2003. Indra: A peer-to-peer approach to network intrusion detection and prevention. In *Proceedgings of 2003 IEEE WET ICE Workshop on Enterprize Security*. Linz, Austria.

JOVANOVIC, M. 2000. Modelling large-scale peer-to-peer networks and a case study of Gnutella. M.S. thesis, Department of Electrical and Computer Engineering and Computer Science, University of Cincinnati, Cincinnati, OH 45221.

JOVANOVICH, M., ANNEXSTEIN, F., AND BERMAN, K. 2001. Scalability issues in large peer-to-peer networks—a case study of Gnutella. Tech. rep., ECECS Department, University of Cincinnati, Cincinnati, OH 45221.

Jxta 2003. The project JXTA web site. http://www.jxta.org.

KALOGERAKI, V., GUNOPOULOS, D., AND ZEINALIPOUR-YAZTI, D. 2002. A local search mechanism for peer-to-peer networks. In *Proceedings of the 11th International Conference on Information and Knowledge Management (CIKM'02)*. McLean, VA.

KAMVAR, S. D., SCHLOSSER, M. T., AND GARCIA-MOLINA, H. 2003. The Eigentrust algorithm for reputation management in p2p networks. In *Proceedings of the 12th International Conference on World Wide Web*. ACM Press, 640–651.

KARGER, D., LEHMAN, E., LEIGHTON, F., LEVINE, M., LEWIN, D., AND PANIGRAHY, R. 1997. Consistent hashing and random trees: Distributed caching protocols for relieving hot spots on the world wide web. In *Proceedings of the 29th Annual ACM Symposium on Theory of Computing*. El Paso, TX, 654–663.

Kazaa 2003. The Kazaa web site. http://www.kazaa.com.

KEROMYTIS, A., V, M., AND RUBENSTEIN, D. 2002. SOS: Secure overlay services. In *Proceedings of the ACM SIGCOMM'02 Conference*. Pittsburgh, PA.

KHAMBATTI, M., DASGUPTA, P., AND RYU, K. 2004. A role-based trust model for peer-to-peer communities and dynamic coalitions. In *Proceedings of the Second IEEE International Information Assurance Workshop*. Charlotte, NC.

KHAMBATTI, M., RYU, K., AND DASGUPTA, P. 2003. Structuring peer-to-peer networks using interest-based communities. In *Proceedings of the International Workshop On Databases, Information Systems and Peer-to-Peer Computing (P2PDBIS)*. Berlin, Germany.

KIM, H. 2001. P2p overview. Tech. rep., Korea Advanced Institute of Technology. (Aug.)

KUBIATOWICS, J. 2003. Extracting guarantees from chaos. *Comm. ACM 46*, 2 (Feb.), 33–38.

KUBIATOWICZ, J., BINDEL, D., CHEN, Y., EATON, P., GEELS, D., GUMMADI, S., WEATHERSPOON, H., WEIMER, W., WELLS, C., AND ZHAO, B. 2000. Oceanstore: An architecture for global-scale persistent storage. In *Proceedings of ACM ASPLOS*.

LAI, K., FELDMAN, M., STOICA, I., AND CHUANG, J. 2003. Incentives for cooperation in peer-to-peer networks. In *Proceedings of the Workshop on Economics of Peer-to-Peer Systems*. Berkeley, CA.

LAMPORT, L., SHOSTAK, R., AND PEASE, M. 1982. The Byzantine generals problem. *ACM Trans. Program. Lang. Syst. 4*, 3 (July), 382–401.

LARSON, S., SNOW, C., AND PANDE, V. 2003. *Modern Methods in Computational Biology,*. (Chapter Folding@Home and Genome@Home: Using distributed computing to tackle previously intractable problems in computational biology) Horizon Press.

LEE, J. 2003. An end-user perspective on file-sharing systems. *Comm. ACM 46*, 2 (Feb.), 49–53.

LIBEN-NOWELL, D., BALAKRISHNAN, H., AND KARGER, D. 2002a. Analysis of the evolution of peer-to-peer systems. In *Proceedings of the ACM Symposium on the Principles of Distributed Computing (PODC)*. Monterey, CA.

LIBEN-NOWELL, D., BALAKRISHNAN, H., AND KARGER, D. 2002b. Observations on the dynamic evolution of peer-to-peer networks. In *Proceedings of the 1st International Workshop on Peer-to-Peer Systems (IPTPS'02)*. MIT Faculty Club, Cambridge, MA.

LOESER, A., WOLPERS, M., SIBERSKI, W., AND NEJDL, W. 2003. Semantic overlay clusters within super-peer networks. In *Proceedings of the International Workshop On Databases, Information Systems and Peer-to-Peer Computing (P2PDBIS)*. Berlin, Germany.

LV, Q., CAO, P., COHEN, E., LI, K., AND SHENKER, S. 2002. Search and replication in unstructured peer-to-peer networks. In *Proceedings of the 16th ACM International Conference on Supercomputing (ICS'02)*. New York, NY.

LV, Q., RATNASAMY, S., AND SHENKER, S. 2002. Can heterogeneity make Gnutella scalable? In *Proceedings of the 1st International Workshop on Peer-to-Peer Systems (IPTPS'02)*. MIT Faculty Club, Cambridge, MA.

MAYAMOUNKOV, P. AND MAZIERES, D. 2002. Kademlia: A peer-to-peer information system based on the xor metric. In *Proceedings of the 1st International Workshop on Peer-to-Peer Systems (IPTPS'02)*. MIT Faculty Club, Cambridge, MA.

MojoNation 2003. The MojoNation web site. http://www.mojonation.net.

NEJDL, W., WOLF, B., QU, C., DECKER, S., SINTEK, M., NAEVE, A., NILSSON, M., PALMER, M., AND RISCH, T. 2003. Edutella: A p2p networking infrastructure based on rdf. In *Proceedings of the 12th International Conference on World Wide Web*. Budapest, Hungary.

PLAXTON, C., RAJARAMAN, R., AND RICHA, A. 1997. Accessing nearby copies of replicated objects in a distributed environment. In *Proceedings of ACM SPAA*. ACM.

RABIN, M. 1989. Efficient dispersal of information for security, load balancing and fault tolerance. *J. ACM 36*, 2 (April), 335–348.

RAMASWAMY, L. AND LIU, L. 2003. Free riding: A new challenge for peer-to-peer file sharing systems. In *Proceedings of the Hawaii International Conference on Systems Science*.

RATNASAMY, S., FRANCIS, P., HANDLEY, M., AND KARP, R. 2001. A scalable content-addressable network. In *Proceedings of SIGCOMM 2001*.

REITER, M. AND RUBIN, A. 1999. Anonymous web transactions with Crowds. *Comm. ACM 42*, 2, 32–38.

RHEA, S., WELLS, C., ET AL. 2001. Maintenance-free global storage. *IEEE Internet Compu.*, 40–49.

RIPEANU, M. AND FOSTER, I. 2002. Mapping the Gnutella network: Macroscopic properties of large-scale peer-to-peer systems. In *Proceedings of the 1st International Workshop on Peer-to-Peer Systems (IPTPS'02)*. MIT Faculty Club, Cambridge, MA.

ROWSTRON, A. AND DRUSCHEL, P. 2001. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. In *Proceedings of IFIP/ACM Middleware*. Heidelberg, Germany.

SAROIU, S., GUMMADI, P., AND GRIBBLE, S. 2002. Exploring the design space of distributed peer-to-peer systems: Comparing the web, TRIAD and Chord/CFS. In *Proceedings of the 1st International Workshop on Peer-to-Peer Systems (IPTPS '02)*. MIT Faculty Club, Cambridge, MA.

SCHODER, D. AND FISCHBACH, K. 2003. Peer-to-peer prospects. *Comm. ACM 46*, 2 (Feb.), 27–29.

SERJANTOV, A. 2002. Anonymizing censorship resistant systems. In *Proceedings of the 1st International Workshop on Peer-to-Peer Systems (IPTPS'02)*. MIT Faculty Club, Cambridge, MA.

SetiAtHome 2003. The seti@home project web site. http://setiathome.ssl.berkeley.edu.

SHAMIR, A. 1979. How to share a secret. *Comm. ACM 22*, 612–613.

SHAW, M. AND GARLAN, D. 1995. Formulations and formalisms in software architecture. In *Computer Science Today: Recent Trends and Developments*, Lecture Notes in Computer Science, 1000. J. van Leeuwen, Ed. Springer Verlag, 307–323.

SHIRKY, C. 2000. What is p2p... and what isn't. *Network, available online at http://www.oreillynet.com/pub/a/p2p/2000/11/24/shirky1-whatisp2p.html*. O'Reilly

STOICA, I., ADKINS, D., ZHUANG, S., SHENKER, S., AND SURANA, S. 2002. Internet indirection infrastructure. In *Proceedings of the ACM SIGCOMM'02 Conference*. Pittsburgh, PA.

STOICA, I., MORRIS, R., KARGER, D., KAASHOEK, M., AND BALAKRISHNAN, H. 2001. Chord: A scalable peer-to-peer lookup service for internet applications. In *Proceedings of SIGCOMM 2001*.

STUBBLEFIELD, A. AND WALLACH, D. 2001. Dagster:censorship-resistant publishing without replication. Tech. Rep. Technical Report TR01-380, Rice University, Dept. of Computer Science. (July).

SULLIVAN III, W., WERTHIMER, D., BOWYER, S., COBB, J., GEDYE, D., AND ANDERSON, D. 1997. A new major seti project based on project serendip data and 100,000 personal computers. In *Proceedings of the 5th International Conference on Bioastronomy*.

TSOUMAKOS, D. AND ROUSSOPOULOS, N. 2003. A comparison of peer-to-peer search methods. In *Proceedings of the Sixth International Workshop on the Web and Databases*. San Diego, CA.

VANRENESSE, R., BIRMAN, K., BOZDOG, A., DIMITRIU, D., SINGH, M., AND VOGELS, W. 2003. Heterogeneity-aware peer-to-peer multicast. In *Proceedings of the 17th International Symposium on Distributed Computing (DISC2003)*. Sorrento, Italy.

VISHNIMURTHY, V., CHANDRAKUMAR, S., AND GUN SIRER, E. 2003. Karma: A secure economic framework for p2p resource sharing. In *Proceedings of the 1st Workshop on Economics of Peer-to-Peer Systems*.

VLACHOS, V., ANDROUTSELLIS-THEOTOKIS, S., AND SPINELLIS, D. 2004. Security applications of peer-topeer networks. *Comput. Netw. J. 45*, 2, 195–205.

WALDMAN, M., AD, R., AND LF, C. 2000. Publius: A robust, tamper-evident, censorship-resistant web publishing system. In *Proceedings of the 9th USENIX Security Symposium*.

WALDMAN, M. AND MAZI, D. 2001. Tangler: a censorship-resistant publishing system based on document entanglements. In *Proceedings of the ACM Conference on Computer and Communications Security*. 126–131.

WALLACH, D. 2002. A survey of peer-to-peer security issues. In *International Symposium on Software Security*. Tokyo, Japan.

WITTEN, I., MOFFAT, A., AND BELL, T. 1999. *Managing Gigabytes: Compressing and Indexing Documents and Images*, 2nd ed. Morgan Kauffman.

XIONG, L. AND LIU, L. 2002. Building trust in decentralized peer-to-peer communities. In *Proceedings of the International Conference on Electronic Commerce Research*.

YANG, B. AND GARCIA-MOLINA, H. 2001. Comparing hybrid peer-to-peer systems. In *Proceedings of the 27th VLDB Conference*. Rome, Italy, 561–570.

YANG, B. AND GARCIA-MOLINA, H. 2002a. Designing a super-peer network. Tech. rep., Stanford University. (Feb.). Available online: http://dbpubs.stanford.edu/pub/2002-13.

YANG, B. AND GARCIA-MOLINA, H. 2002b. Improving search in peer-to-peer networks. In *Proceedings of the 22nd International Conference on Distributed Computing Systems (ICDCS'02)*. Vienna, Autria.

YU, B. AND SINGH, M. 2003. Incentive mechanisms for peer-to-peer systems. In *Proceedings of the 2nd International Workshop on Agents and Peer-to-Peer Computing*.

ZHANG, X., ZHANG, Q., ZHANG, Z., SONG, G., AND ZHU, W. 2004. A construction of locality-aware overlay network: moverlay and its performance. *IEEE JSAC Special Issue on Recent Advances on Service Overlay Networks*.

ZHAO, B., JOSEPH, A., AND KUBIATOWICZ, J. 2002. Locality aware mechanisms for large-scale networks. In *Proceedings of the International Workshop on Future Directions in Distributed Computing (FuDiCo2002)*.

ZHAO, B., KUBIATOWICZ, J., AND JOSEPH, A. 2001. Tapestry: An infrastructure for fault-tolerant wide-area location and routing. Tech. Rep. UCB/CSD-01-1141, Computer Science Division, University of California, Berkeley, 94720. (April)

ZHICHEN, X., MAHALINGAM, M., AND KARLSSON, M. 2002. Turning heterogeneity to an advantage in overlay routing. Tech. Rep. HPL-2002-126, HP Labs.

ZHOU, J., DIALANI, V., DE ROURE, D., AND HALL, W. 2003. A semantic search algorithm for peer-to-peer open hypermedia systems. In *Proceedings of the 1st Workshop on Semantics in Peer-to-Peer and Grid Computing at the 12th International World Wide Web Conference*. Budapest, Hungary.

Received May 2003; revised May 2004; accepted September 2004