Peer-to-Peer Computing

Dejan S. Milojicic, Vana Kalogeraki, Rajan Lukose,
Kiran Nagaraja[1], Jim Pruyne, Bruno Richard,
Sami Rollins [2] , Zhichen Xu
HP Laboratories Palo Alto
HPL-2002-57 (R.1)
July 3rd , 2003*

E-mail: [dejan, vana, lukose, pruyne, zhichen] @ exch.hpl.hp.com, bruno_richard @ hp.com,
knagaraj @ cs.rutgers.edu, srollins @ cs.ucsb.edu

peer-to-peer, decentralization, self-organization, anonymity, cost of ownership

The term "peer-to-peer" (P2P) refers to a class of systems and applications that employ distributed resources to perform a function in a decentralized manner. With the pervasive deployment of computers, P2P is increasingly receiving attention in research, product development, and investment circles. Some of the benefits of a P2P approach include: improving scalability by avoiding dependency on centralized points; eliminating the need for costly infrastructure by enabling direct communication among clients; and enabling resource aggregation.

This survey reviews the field of P2P systems and applications by summarizing the key concepts and giving an overview of the most important systems. Design and implementation issues of P2P systems are analyzed in general, and then revisited for eight case studies. This survey will help people in the research community and industry understand the potential benefits of P2P. For people unfamiliar with the field it provides a general overview, as well as detailed case studies. Comparison of P2P solutions with alternative architectures is intended for users, developers, and system administrators (IT).

* Internal Accession Date Only
Approved for External Publication?
[1] Rutgers University, NJ, 08901
[2] University of California at Santa Barbara, CA, 93106?

Peer-to-Peer Computing

DEJAN S. MILOJIČIĆ[1], VANA KALOGERAKI[1], RAJAN LUKOSE[1], KIRAN NAGARAJA[2], JIM PRUYNE[1], BRUNO RICHARD[1], SAMI ROLLINS[3], and ZHICHEN XU[1]

[dejan, vana, lukose, pruyne, zhichen]@exch.hpl.hp.com, bruno_richard@hp.com, knagaraj@cs.rutgers.edu, srollins@cs.ucsb.edu

[1]HP Labs, [2]Rutgers University, [3]University of California at Santa Barbara

Abstract

The term "peer-to-peer" (P2P) refers to a class of systems and applications that employ distributed resources to perform a function in a decentralized manner. With the pervasive deployment of computers, P2P is increasingly receiving attention in research, product development, and investment circles. Some of the benefits of a P2P approach include: improving scalability by avoiding dependency on centralized points; eliminating the need for costly infrastructure by enabling direct communication among clients; and enabling resource aggregation.

This survey reviews the field of P2P systems and applications by summarizing the key concepts and giving an overview of the most important systems. Design and implementation issues of P2P systems are analyzed in general, and then revisited for eight case studies. This survey will help people in the research community and industry understand the potential benefits of P2P. For people unfamiliar with the field it provides a general overview, as well as detailed case studies. Comparison of P2P solutions with alternative architectures is intended for users, developers, and system administrators (IT).

Categories and Subject Descriptors: C.2.4 [**Computer-Communication Networks**]: Distributed Systems - network operating systems; D.1.3 [**Programming Techniques**]: Concurrent Programming - distributed programming; D.4.7 [**Operating Systems**]: Organization and Design - distributed systems; E.1 [**Data**]: Data Structures - distributed data structures; F.1.2 [**Theory of Computation**]: Modes of Computation - parallelism and concurrency; H.3.4 [**Information Systems**]: Systems and Software - Distributed systems.

General Terms: design, experimentation

Additional Key Words and Phrases: peer-to-peer, decentralization, self-organization, anonymity, cost of ownership.

1 INTRODUCTION

Peer-to-Peer (P2P) computing is a very controversial topic. Many experts believe that there is not much new in P2P. There is a lot of confusion: what really constitutes P2P? For example, is distributed computing really P2P or not? We believe that P2P does warrant a thorough analysis. The goals of the paper are threefold: 1) to understand what P2P is and it is not, as well as what is new, 2) to offer a thorough analysis of and examples of P2P computing, and 3) to analyze the potential of P2P computing.

The term "peer-to-peer" refers to a class of systems and applications that employ distributed resources to perform a function in a decentralized manner. The resources encompass computing power, data (storage and content), network bandwidth, and presence (computers, human, and other resources). The critical function can be distributed computing, data/content sharing, communication and collaboration, or platform services. Decentralization may apply to algorithms, data, and meta-data, or to all of them. This does not preclude retaining centralization in some parts of the systems and applications. Typical P2P systems reside on the edge of the Internet or in ad-hoc networks. P2P enables:

- **valuable externalities**, by aggregating resources through low-cost interoperability, the whole is made greater than the sum of its parts
- **lower cost of ownership and cost sharing**, by using existing infrastructure and by eliminating or distributing the maintenance costs
- **anonymity/privacy**, by incorporating these requirements in the design and algorithms of P2P systems and applications, and by allowing peers a greater degree of autonomous control over their data and resources

However, P2P also raises some security concerns for users and accountability concerns for IT. In general it is still a technology in development where it is hard to distinguish useful from hype and new from old. In the rest of the paper we evaluate these observations in general as well as for specific P2P systems and applications.

P2P gained visibility with Napster's support for music sharing on the Web [Napster 2001] and its lawsuit with the music companies. However, it is increasingly becoming an important technique in various areas, such as distributed and collaborative computing both on the Web and in ad-hoc networks. P2P has received the attention of both industry and academia. Some big industrial efforts include the P2P Working Group, led by many industrial partners such as Intel, HP, Sony, and a number of startup companies; and JXTA, an open-source effort led by Sun. There are already a number of books published [Oram 2000, Barkai 2001, Miller 2001, Moore and Hebeler 2001, Fattah and Fattah 2002], and a number of theses and projects in progress at universities, such as Chord [Stoica et al 2001], OceanStore [Kubiatowicz et al. 2000], PAST [Druschel and Rowstron 2001], CAN [Ratnasamy 2001], and FreeNet [Clark 1999].

Here are several of the definitions of P2P that are being used by the P2P community. The Intel P2P working group defines P2P as "the sharing of computer resources and services by direct exchange between systems" [p2pwg 2001]. David Anderson calls SETI@home and similar P2P projects that do not involve communication as "inverted client-server", emphasizing that the computers at the edge provide power and those in the middle of the network are there only to coordinate them [Anderson 2002]. Alex Weytsel of Aberdeen defines P2P as "the use of devices on the internet periphery in a non-client capacity" [Veytsel 2001]. Clay Shirky of O'Reilly and Associate uses the following definition: "P2P is a class of applications that takes advantage of resources – storage, cycles, content, human presence – available at the edges of the Internet. Because accessing these decentralized resources means operating in an environment of unstable connectivity and unpredictable IP addresses, P2P nodes must operate outside the DNS system and have significant or total autonomy from central servers" [Shirky 2001]. Finally, Kindberg defines P2P systems as those with independent lifetimes [Kindberg 2002].

In our view, P2P is about sharing: giving to and obtaining from a peer community. A peer gives some resources and obtains other resources in return. In the case of Napster, it was about offering music to the rest of the community and getting other music in return. It could be donating resources for a good cause, such as searching for extraterrestrial life or combating cancer, where the benefit is obtaining the satisfaction of helping others. P2P is also a way of implementing systems based on the notion of increasing the decentralization of systems, applications, or simply algorithms. It is based on the principles that the world will be connected and widely distributed and that it will not be possible or desirable to leverage everything off of centralized, administratively managed infrastructures. P2P is a way to leverage vast amounts of computing power, storage, and connectivity from personal computers distributed around the world.

Assuming that "peer" is defined as "like each other," a P2P system then is one in which autonomous peers depend on other autonomous peers. Peers are autonomous

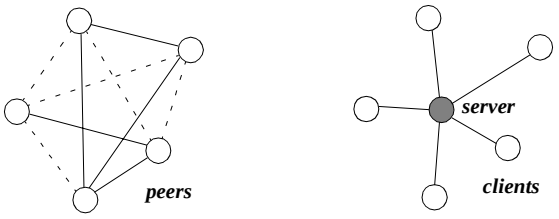

Figure 1: Simplified, High-Level View of Peer-to-Peer versus Centralized (Client-Server) Approach.

when they are not wholly controlled by each other or by the same authority, e.g., the same user. Peers depend on each other for getting information, computing resources, forwarding requests, etc. which are essential for the functioning of the system as a whole and for the benefit of all peers. As a result of the autonomy of peers, they cannot necessarily trust each other and rely completely on the behavior of other peers, so issues of scale and redundancy become much more important than in traditional centralized or distributed systems.

Conceptually, P2P computing is an alternative to the centralized and client-server models of computing, where there is typically a single or small cluster of servers and many clients (see Figure 1). In its purest form, the P2P model has no concept of server; rather all participants are peers. This concept is not necessarily new. Many earlier distributed systems followed a similar model, such as UUCP [Nowitz 1978] and switched networks [Tanenbaum 1981]. The term P2P is also not new. In one of its simplest forms, it refers to the communication among the peers. For example, in telephony users talk to each other directly once the connection is established, in a computer networks the computers communicate P2P, and in games, such as Doom, players also interact directly. However, a comparison between client-server and P2P computing is significantly more complex and intertwined along many dimensions. Figure 2 is an attempt to compare some aspects of these two models.

The P2P model is quite broad and it could be evaluated from different perspectives. Figure 3 categorizes the scope of P2P development and deployment. In terms of development, platforms provide an infrastructure to support P2P applications. Additionally, developers are beginning to explore the benefit of implementing various horizontal technologies such as distributed computing, collaborative, and content sharing software using the P2P model rather than more traditional models such as client-server. Applications such as file sharing and messaging software are being deployed in a number of different vertical markets. Section 2.3 provides a more thorough evaluation of P2P markets and Section 5 describes the horizontal technologies in more detail.

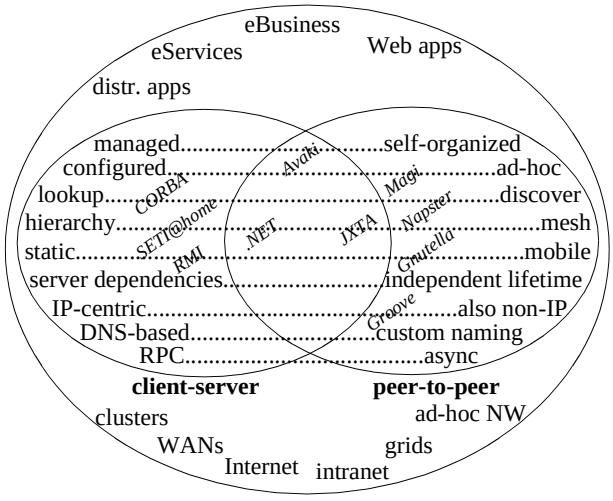

Figure 2: Peer-to-Peer versus Client-Server. *There is no clear border between a client-server and a P2P model. Both models can be built on a spectrum of levels of characteristics (e.g., manageability, configurability), functionality (e.g., look-up versus discovery), organizations (e.g., hierarchy versus mesh), components (e.g., DNS), and protocols (e.g., IP), etc. Furthermore, one model can be built on top of the other or parts of the components can be realized in one or the other model. Finally, both models can execute on different types of platforms (Internet, intranet, etc.) and both can serve as an underlying base for traditional and new applications. Therefore, it should not be a surprise that there is so much confusion about what P2P is and what it is not. It is extremely intertwined with existing technologies [Morgan 2002].*

The following three lists, which are summarized in Table 1, are an attempt to define the nature of P2P, what is and is not new in P2P.

P2P is concerned with:

- The historical evolution of computing in general and the Internet in particular; computing at the edge of the Internet. (e.g., SETI@home and other distributed computing systems)

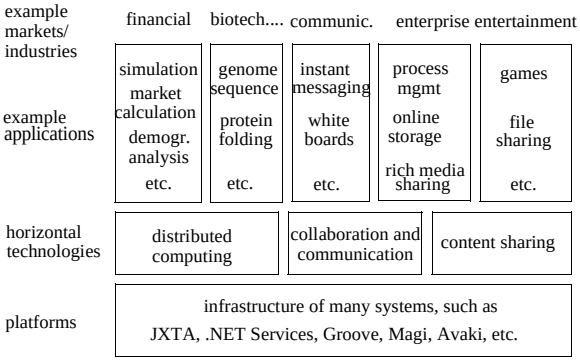

Figure 3: P2P Solutions. *P2P can be classified into interoperable P2P platforms, applications of the P2P technology, and vertical P2P applications.*

Perspective	Comparison								
	What is New		What is Not New		Well-Known Examples	Enabler	Enabling	Alternatives	Paper Roadmap
Historical/Evolutionary (Computing)	computing on the edge of the Internet	scalability, availability, security, connectivity	distributed scheduling	concept, applications, distribution	SETI@home	ubiquitous computing/communication	use vast available computer power at home and office	classic network clusters, Grids	Sections 5.2, 6.1, 6.2
Cultural/Sociological (Content sharing/Services)	direct sharing (privacy, anonymity)		decentralization		Napster	broad Internet connectivity	user-to-user exchange, minimal broker engagement	client-server, B2B Web Services	Sections 5.3, 6.5, 6.6
Communication/ Collaboration	apps & systems for ad-hoc & disconnected		ad-hoc NW, disconnected operation		AOL Chat, Groove parasitic NW	new NWs: wireless, broadband	improved communication and collaboration	Lotus Notes, NetMeeting	Sections 5.4, 6.3, 6.4
Architectural	cost of ownership		P2P concept and applications		JXTA, .NET	increased component decentralization	larger scale, better accessibility	CORBA, RMI, other middleware	Sections 5.5, 6.7, 6.8
Algorithms/ Programming Model	particular algorithms		distributed state algor. in general		Gnutella	decentralized state	improved scalability, availability, and anonymity	client-server	Section 5.3

Table 1. Various P2P Perspectives Illustrating What is and What is Not New in P2P.

- Sociological aspects of sharing of content (e.g., Napster and other file/content sharing systems)
- Technological improvements to networks and communication advances (e.g., wireless networks, handheld devices enabling better collaboration and communication)
- Software architectures (e.g., JXTA or .NET)
- Deployed algorithms (e.g., Gnutella, FreeNet).

New aspects of P2P include:

- Technology requirements
  - scale of deployed computers
  - ad-hoc connectivity
  - security (e.g., crossing a firewall)
- Architectural requirements
  - availability of computers
  - scalability of future systems (world-wide and embedded)
  - privacy and anonymity (beyond the Web)
- Economy requirements
  - cost of ownership
  - pervasive use (home versus business – expectations about quality and guarantees versus best effort)

Aspects of P2P that are not new include:

- Concept and applications (e.g., telephony, networks, servers)
- Decentralization for scalability and availability (e.g., distributed systems in general, replication)
- Distributed state management (e.g., load information management in load balancing systems)
- Disconnected operations in general (e.g., work in mobile computing systems)
- Distributed scheduling algorithms (e.g., on clusters and grids)
- Scalability (WWW and Web services in particular)
- Ad-hoc networks
- eBusiness
- Algorithms (many P2P and distributed algorithms already exist)

Paper Organization and Intended Audience

The paper is organized as follows. Section 2 provides background on P2P. Section 3 presents P2P components and algorithms. Section 4 describes characteristics of P2P solutions. Section 5 classifies P2P systems into five categories and describes a few representatives for each category. In Section 6, we present more detailed case studies. Section 7 presents lessons learned from analyzing the P2P area. Finally, in Section 8, we summarize the paper and describe opportunities for future research.

This paper is intended for people new to the field of P2P as well as for experts. It is intended for users, developers, and IT personnel. This first section provided a brief overview of the field. Readers interested in learning about the field in general should read Sections 2 through 5. Experts can benefit from the detailed case studies in Section 6. Perspectives of users, developers, and IT personnel are addressed in Section 7.3. We assume that the readers have a general knowledge of computing systems.

2 OVERVIEW

P2P is frequently confused with other terms, such as traditional distributed computing [Coulouris et al. 2001], grid computing [Foster and Kesselman 1999], and ad-hoc networking [Perkins 2001]. To better define P2P, this section introduces P2P goals, terminology, and taxonomies.

2.1 Goals

As with any computing system, the goal of P2P systems is to support applications that satisfy the needs of users.

Selecting a P2P approach is often driven by one or more of the following goals.

- **Cost sharing/reduction**. Centralized systems that serve many clients typically bear the majority of the cost of the system. When that main cost becomes too large, a P2P architecture can help spread the cost over all the peers. For example, in the file-sharing space, the Napster system enabled the cost sharing of file storage, and was able to maintain the index required for sharing. Much of the cost sharing is realized by the utilization and aggregation of otherwise unused resources (e.g. SETI@home), which results both in net marginal cost reductions and a lower cost for the *most* costly system component. Because peers tend to be autonomous, it is important for costs to be shared reasonably equitably.
- **Resource aggregation (improved performance) and interoperability**. A decentralized approach lends itself naturally to aggregation of resources. Each node in the P2P system brings with it certain resources such as compute power or storage space. Applications that benefit from huge amounts of these resources, such as compute-intensive simulations or distributed file systems, naturally lean toward a P2P structure to aggregate these resources to solve the larger problem. Distributed computing systems, such as SETI@Home, distributed.net, and Endeavours are obvious examples of this approach. By aggregating compute resources at thousands of nodes, they are able to perform computationally intensive functions. File sharing systems, such as Napster, Gnutella, and so forth, also aggregate resources. In these cases, it is both disk space to store the community's collection of data and bandwidth to move the data that is aggregated. Interoperability is also an important requirement for the aggregation of diverse resources.
- **Improved scalability/reliability**. With the lack of strong central authority for autonomous peers, improving system scalability and reliability is an important goal. As a result, algorithmic innovation in the area of resource discovery and search has been a clear area of research, resulting in new algorithms for existing systems, and the development of new P2P platforms (such as CAN [Ratnasamy 2001], Chord [Stoica et al. 2001], and PAST [Rowstron and Druschel 2001]). These developments will be discussed in more detail in Section 3. Scalability and reliability are defined in traditional distributed system terms, such as the bandwidth usage — how many systems can be reached from one node, how many systems can be supported, how many users can be supported, and how much storage can be used. Reliability is related to systems and network failure, disconnection, availability of resources, etc.
- **Increased autonomy**. In many cases, users of a distributed system are unwilling to rely on any centralized service provider. Instead, they prefer that all data and work on their behalf be performed locally. P2P systems support this level of autonomy simply because they require that the local node do work on behalf of its user. The principle example of this is the various file sharing systems such as Napster, Gnutella, and FreeNet. In each case, users are able to get files that would not be available at any central server because of licensing restrictions. However, individuals autonomously running their own servers have been able to share the files because they are more difficult to find than a server operator would be.
- **Anonymity/privacy**. Related to autonomy is the notion of anonymity and privacy. A user may not want anyone or any service provider to know about his or her involvement in the system. With a central server, it is difficult to ensure anonymity because the server will typically be able to identify the client, at least by Internet address. By employing a P2P structure in which activities are performed locally, users can avoid having to provide any information about themselves to anyone else. FreeNet is a prime example of how anonymity can be built into a P2P application. It uses a forwarding scheme for messages to ensure that the original requestor of a service cannot be tracked. It increases anonymity by using probabilistic algorithms so that origins cannot be easily tracked by analyzing network traffic.
- **Dynamism**. P2P systems assume that the computing environment is highly dynamic. That is, resources, such as compute nodes, will be entering and leaving the system continuously. When an application is intended to support a highly dynamic environment, the P2P approach is a natural fit. In communication applications, such as Instant Messaging, so-called "buddy lists" are used to inform users when persons with whom they wish to communicate become available. Without this support, users would be required to "poll" for chat partners by sending periodic messages to them. Likewise, distributed computing applications such as distributed.net and SETI@home must adapt to changing participants. They therefore must re-issue computation jobs to other participants to ensure that work is not lost if earlier participants drop out of the network while they were performing a computation step.
- **Enabling ad-hoc communication and collaboration**. Related to dynamism is the notion of supporting ad-hoc environments. By ad hoc, we mean environments where members come and go based perhaps on their current physical location or their current inter-

ests. Again, P2P fits these applications because it naturally takes into account changes in the group of participants. P2P systems typically do not rely on established infrastructure — they build their own, e.g., logical overlay in CAN and PAST.

2.2 Terminology

The following terminology is used in the taxonomies in Section 2.3 and in the comparisons between P2P and its alternatives in Section 7.1.

- *Centralized systems* represent single-unit solutions, including single- and multi-processor machines, as well as high-end machines, such as supercomputers and mainframes.
- *Distributed systems* are those in which components located at networked computers communicate and coordinate their actions only by passing messages [Couloris, et al. 2001].
- *Client* is informally defined as an entity (node, program, module, etc.) that initiates requests but is not able to serve requests. If the client also serves the request, then it plays the role of a server.
- *Server* is informally defined as an entity that serves requests from other entities, but does not initiate requests. If the server does initiate requests, then it plays the role of a client. Typically, there are one or a few servers versus many clients.
- *Client-Server model* represents the execution of entities with the roles of clients and servers. Any entity in a system can play both roles but for a different purpose, i.e. server and client functionality residing on separate nodes. Similarly an entity can be a server for one kind of request and client for others.
- *Peer* is informally defined as an entity with capabilities similar to other entities in the system.
- *P2P model* enables peers to share their resources (information, processing, presence, etc.) with at most a limited interaction with a centralized server. The peers may have to handle a limited connectivity (wireless, unreliable modem links, etc.), support possibly independent naming, and be able to share the role of the server [Oram 2000]. It is equivalent to having all entities being client and servers for the same purpose.

Other terms frequently associated with P2P include:

- *Distributed computing, which* is defined as "a computer system in which several interconnected computers share the computing tasks assigned to the system" [IEEE 1990]. Such systems include *computing clusters*, *Grids* (see below), and *global computing systems* gathering computing resources from individual PCs over the Internet.

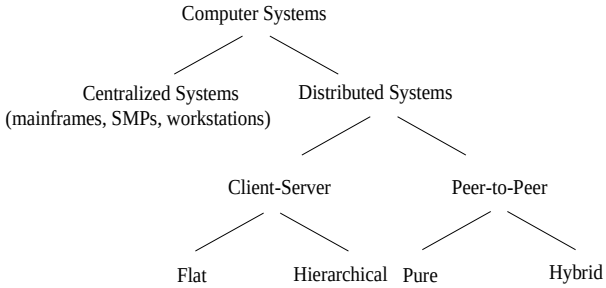

Figure 4: A Taxonomy of Computer Systems Architectures.

- *Grid computing*, which is defined as "coordinated resource sharing and problem solving in large, multi-institutional virtual organization." [Foster and Kesselman 1999]. More specifically, a Grid is an infrastructure for globally sharing compute-intensive resources such as supercomputers or computational clusters.
- *Ad-hoc communication, which is defined as a system that* enables communication to take place without any preexisting infrastructure in place, except for the communicating computers. These computers form an ad-hoc network. This network and associated computers take care of communication, naming, and security. P2P systems can be used on top of an ad-hoc communication infrastructure.

There are many examples of distributed systems, at various scales, such as the Internet, wide-area networks, intranets, local-area networks, etc. Distributed system components can be organized in a P2P model or in a client-server model. (We believe that other models, such as three-tier and publish-subscribe, can be mapped onto client-server). Typical client examples include Web browsers (e.g., Netscape Communicator or Internet Explorer), file system clients, DNS clients, CORBA clients, etc. Server examples include name servers (DNS [Albitz and Liu 1995, Mockapetris 1989], LDAP [Howes and Smith 2001], etc.), distributed file servers (NFS [Sandberg et al. 1985], AFS [Howard et al. 1988], CORBA Object Request Brokers [OMG 1996], HTTP Server, authentication server, etc. Client-server model examples include CORBA, RMI [Wollrath et al 1996], and other middleware [Bernstein 1996; Britton 2000]). Peer examples include computers in a network that serve a similar role. There are numerous examples of the P2P model throughout the paper.

2.3 P2P Taxonomies

A taxonomy of computer systems from the P2P perspective is presented in Figure 4. All computer systems can be classified into centralized and distributed. Distributed systems can be further classified into the client-server

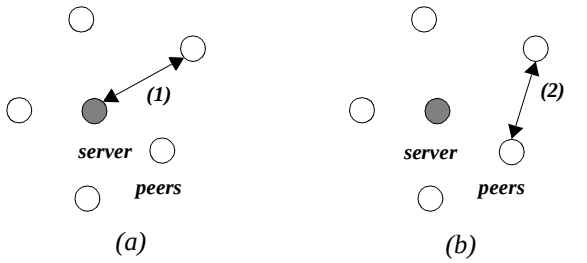

Figure 5: Hybrid Peer-to-Peer Model. *(1) Initial communication with a server, e.g., to obtain the location/identity of a peer, followed by (2) direct communication with a peer.*

model and the P2P model. The client-server model can be flat where all clients only communicate with a single server (possibly replicated for improved reliability), or it can be hierarchical for improved scalability. In a hierarchal model, the servers of one level are acting as clients to higher level servers. Examples of a flat client-server model include traditional middleware solutions, such as object request brokers and distributed objects. Examples of a hierarchical client-server model include DNS server and mounted file systems.

The P2P model can either be *pure* or it can be *hybrid*. In a pure model, there does not exist a centralized server. Examples of a pure P2P model include Gnutella and Freenet. In a hybrid model, a server is approached first to obtain meta-information, such as the identity of the peer on which some information is stored, or to verify security credentials (see Figure 5a). From then on, the P2P communication is performed (see Figure 5b). Examples of a hybrid model include Napster, Groove, Aimster, Magi, Softwax, and iMesh. There are also intermediate solutions with SuperPeers, such as KaZaa. SuperPeers contain some of the information that others may not have [KaZaa 2001]. Other peers typically lookup information at SuperPeers if they cannot find it otherwise.

Figure 6 presents a coarse taxonomy of P2P systems. We classify P2P systems into *distributed computing* (e.g., SETI@home, Avaki, Entropia), *file sharing* (e.g., Napster, Gnutella, Freenet, Publius, Free Haven), *collaboration* (e.g., Magi, Groove, Jabber), and *platforms* (e.g., JXTA and .NET My Services). In this paper, under the term platform we assume an infrastructure that enables other applications and/or other systems to run in a P2P

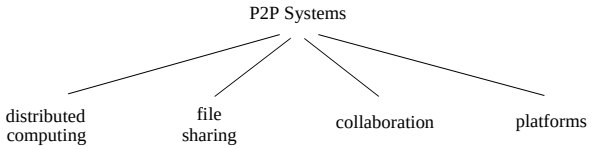

Figure 6: A Taxonomy of P2P System Categories (see also Section 5).

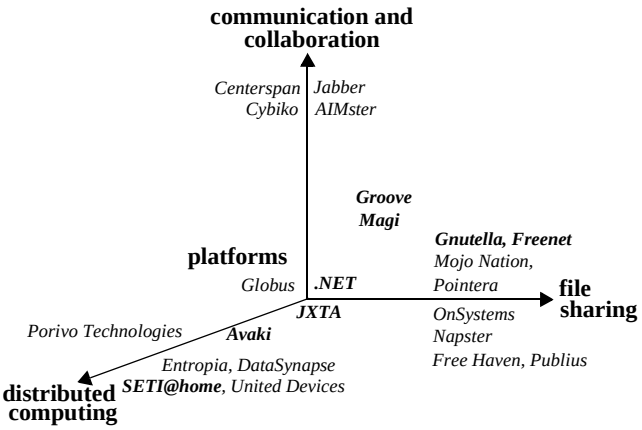

Figure 7: A Classification of P2P Systems Based on the Taxonomy in Figure 6. *Systems in bold-italic are case studies described in detail in Section 6.*

fashion. It is important to point out that many of the systems described in this paper are to various degrees platforms in their own right. For example, we could have also classified Groove and Magi as platforms. Section 6 contains a description of eight case studies of P2P systems according to the taxonomy presented in Figure 6.

In Figure 7, we present a classification of various P2P systems according to the taxonomy presented in Figure 6. This figure demonstrates that certain P2P systems emphasize different aspects along the taxonomy dimensions (computing, storage, communication), whereas the platforms support all of these dimensions. Some of the systems belong to multiple categories. For example, Groove and Magi can also be considered platforms and the also support file sharing.

Application Taxonomy. Three main classes of P2P applications have emerged: parallelizable, content and file management, and collaborative (see Figure 8).

- **Parallel computing.** Parallelizable P2P applications split a large compute-intensive task into smaller subpieces that can execute in parallel over a number of independent peer nodes. Most often, the same task is performed on each peer using different sets of parameters. Examples of implementations include searching for extraterrestrial life [Anderson et al., 2002], code breaking, portfolio pricing, risk hedge calculation,

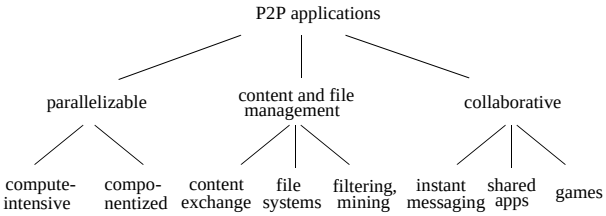

Figure 8: A Taxonomy of P2P Applications.

market and credit evaluation, and demographic analysis. Section 5.2 presents systems supporting this class of application. Componentized applications, such as Workflow, JavaBeans, or Web Services in general, have not been widely recognized as P2P. However, we envision applications that can be built out of finer-grain components that execute over many nodes in parallel. In contrast to compute-intensive applications that run the same task on many peers, componentized applications run different components on each peer.

- **Content and file management.** Content and file management P2P applications focus on storing information on and retrieving information from various peers in the network. Applications, such as Napster [2001] and Gnutella [2001] allow peers to search for and download files, primarily music files, that other peers have made available. For the most part, current implementations have not focused much on providing reliability and rely on the user to make intelligent choices about the location from which to fetch files and to retry when downloads fail. They focus on using otherwise unused storage space as a server for users. These applications must ensure reliability by using more traditional database techniques such as replication. A number of research projects have explored the foundations of P2P file systems [Ratnasamy et al 2001, Bolosky et al 2000, Kubiatowicz et al 2000, Rowstron and Druschel 2001, Gribble et al 2001, Stoica et al 2001]. Finally, filtering and mining applications such as OpenCOLA [2001] and JXTA Search [Waterhouse et al. 2002] are beginning to emerge. Instead of focusing on sharing information, these applications focus on collaborative filtering techniques that build searchable indices over a peer network. A technology such as JXTA Search can be used in conjunction with an application like Gnutella to allow more up-to-date searches over a large, distributed body of information.

- **Collaborative.** Collaborative P2P applications allow users to collaborate in real time, without relying on a central server to collect and relay information. Email and news are classic examples. Instant messaging, such as Yahoo!, AIM, and Jabber, have become popular among a wide variety of users [Strom 2001]. Shared applications that allow people (e.g., business colleagues) to interact while viewing and editing the same information simultaneously, yet possibly thousands of miles apart, are also emerging. Examples include Buzzpad [www.buzzpad.com] and distributed PowerPoint [Rice and Mahon 2000]. Games are another example of a collaborative P2P application. P2P games are hosted on all peer computers and updates are distributed to all peers without requiring a central server. Example games include NetZ by Quazal, Scour Exchange by CenterSpan, Descent, and Cybiko.

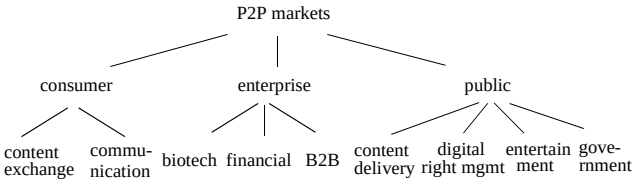

Figure 9: A Taxonomy of P2P Markets. *There are also other markets that can map to some of the presented ones, such as military, education, scientific, etc.*

P2P Target Environments. The target environments for P2P consist of the *Internet, intranets,* and *ad-hoc networks*. P2P systems connected to Internet support connections in the spectrum from dialup lines to broadband (DSL). The underlying architecture can rely on *personal home computers, corporate desktops,* or *personal mobile computers* (laptops and handhelds).

The most frequent environment is personal home computers connected to the Internet. The early P2P systems in this environment were primarily used for content sharing. Examples include Napster, Gnutella, and Aimster. Distributed computing in a P2P fashion also started on desktop computers on the Internet such as SETI@home, but it also gained acceptance in Intranets, such as in DataSynapse [Intel 2001, DataSynapse 2001].

Ad-hoc networks of handhelds are only recently becoming available (e.g., Endeavors Technologies Magi) dedicated for collaborative computing, but similar platforms are expected to follow with a wider deployment of handheld computers and wireless networks.

Future environments may include variations of existing deployment scenarios, such as using corporate desktops for content sharing, e.g., for distributed Internet data centers, or handheld computers for resource aggregation (content sharing or distributed computing). Furthermore, Internet2 (http://www.internet2.edu/) may offer more enabling support for P2P systems and applications.

P2P Markets. There are three main markets for P2P: *consumer*, *enterprise*, and *public*. The *consumer space* encompasses the use of P2P for personal use, such as music and content sharing, instant messaging, email, and games. Application examples include music sharing and communication. Examples of companies in this space include Napster and Gnutella. *Enterprise space* P2P applications include biotech, financial, traditional IT solutions, and B2B. Example companies include Data Synapse, Information Architects, and WorldStreet. Finally, the *public* class of applications includes the following types of applications: information sharing, digital

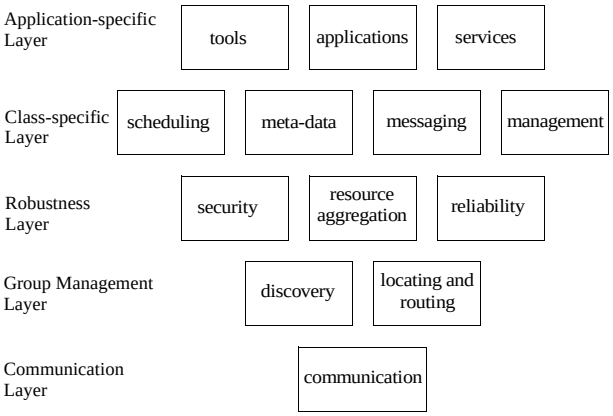

Figure 10: An Informal P2P System Architecture. *The order of components does not strictly follow layering.*

rights management, and entertainment. Centerspan, AIM, and Scour deliver music and video on broadband using P2P technology. Various government agencies can be classified into public sector market, Groove has been delivering solutions to this market.

According to Phillips, markets can be classified based on user activities into those performed @work, @play, and @rest [Nethisinghe 200]. We extended this perspective to map user activities to markets: consumer, enterprise, and public (see Table 2). The insight is that P2P permeates a number of user activities and markets.

3 COMPONENTS AND ALGORITHMS

This section introduces components of P2P systems and algorithms used in P2P.

3.1 Infrastructure Components

Figure 10 illustrates an abstract P2P architecture. In this section, we discuss the functions of each component and look at some of the tradeoffs involved in making component implementation decisions.

Communication. The P2P model covers a wide spectrum of communication paradigms. At one end of the spectrum are desktop machines mostly connected via stable, high-speed links over the Internet [Saroiu et al. 2002]. At the other end of the spectrum, are small wireless devices such as PDAs or even sensor-based devices that are connected in an ad-hoc manner via a wireless medium. The fundamental challenge of communication in a P2P community is overcoming the problems associated with the dynamic nature of peers. Either intentionally (e.g., because a user turns off her computer) or unintentionally (e.g., due to a, possibly dial-up, network link failing) peer groups frequently change. Maintaining application-level connectivity in such an environment is one of the biggest challenges facing P2P developers.

type of activity	**scope**		
	consumer	**enterprise**	**public**
@work	collaboration, communication	distributed computing, storage, communication, collaboration	communication, digital rights mgmt
@play	games	HR-sponsored events	digital media digital experience
@rest	music sharing	content consumption	instant messaging

Table 2. P2P Markets versus P2P Applications.

Group Management. Peer group management includes discovery of other peers in the community and location and routing between those peers. Discovery of peers can be highly centralized such as in Napster [2001], highly distributed such as in Gnutella [2001], or somewhere in between. Existing Gnutella clients rely on central host caches to a large extent. A number of factors influence the design of discovery algorithms. For example, mobile, wireless devices can discover other peers based upon their range of communication [Roman et al. 2001]. Protocols built for desktop machines often use other approaches such as centralized directories.

Location and routing algorithms generally try to optimize the path of a message traveling from one peer to another. While deployed systems such as Napster and Gnutella try to optimize factors such as underlying network latency, the research in this space takes a more systematic approach as discussed in Section 3.2.

Robustness. There are three main components that are essential to maintaining robust P2P systems: security, resource aggregation, and reliability. Security is one of the biggest challenges for P2P infrastructures. A benefit of P2P is that it allows nodes to function as both a client and a server. However, transforming a standard client device into a server poses risks to the system. Only trusted or authenticated sources should have access to information and services provided by a given node. For a more detailed discussion of P2P security concerns see Section 4.8.

The P2P model provides the basis for interacting peers to aggregate resources available on their systems. Classifying the architecture of a P2P resource aggregation component is difficult because there are a wide variety of resources that may be aggregated across peers. On the one hand, resources include files or other content residing on a computer. A wide variety of file sharing systems have addressed the problem of aggregating this type of resource. On the other hand, resources can be defined in

terms of the resources available on a given peer device such as CPU processing power, bandwidth, energy, and disk space.

Reliability in P2P systems is a hard problem. The inherently distributed nature of peer networks makes it difficult to guarantee reliable behavior. The most common solution to reliability across P2P systems is to take advantage of redundancy. For example, in case of compute-intensive applications upon a detection of a failure the task can be restarted on other available machines. Alternatively, the same task can be initially assigned to multiple peers. In file sharing applications, data can be replicated across many peers. Finally, in messaging applications, lost messages can be resent or can be sent along multiple paths simultaneously.

Class-Specific. While the components discussed so far are applicable to any P2P architecture, application-specific components abstract functionality from each class of P2P application. Scheduling applies to parallelizable or compute-intensive applications. Compute-intensive tasks are broken into pieces that must be scheduled across the peer community. Metadata describes the content stored across nodes of the peer community and may be consulted to determine the location of desired information. Metadata applies to content and file management applications. Messaging applies to collaborative applications. Messages sent between peers enable communication. Management supports managing the underlying P2P infrastructure.

Application-Specific. Tools, applications, and services implement application-specific functionality, which correspond to certain P2P applications running on top of the underlying P2P infrastructure. It corresponds to specific cases of distributed scheduling (e.g.scientific, financial, biotechnology, etc.), content and file sharing (e.g., music MP3 file exchange), or specific applications running on top of collaborative and communication systems, such as calendaring, notes, messaging, and chatting.

3.2 Algorithms

This section overviews three common P2P algorithms and then compares their implementations in a few P2P systems.

Centralized directory model. This model was made popular by Napster. The peers of the community connect to a central directory where they publish information about the content they offer for sharing (see Figure 11). Upon request from a peer, the central index will match the request with the best peer in its directory that matches the request. The best peer could be the one that is cheapest, fastest, or the most available, depending on the user needs. Then a file exchange will occur directly between the two peers. This model requires some managed infrastructure (the directory server), which hosts information about all participants in the community. This can cause the model to show some scalability limits, because it requires bigger servers when the number of requests increase, and larger storage when the number of users increase. However, Napster's experience showed that – except for legal issues – the model was very strong and efficient.

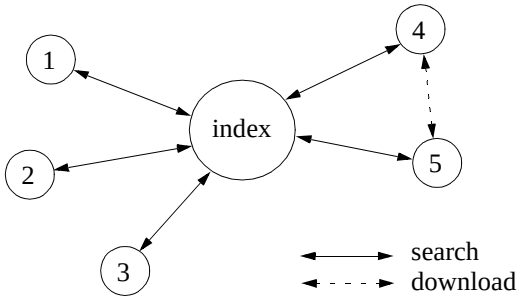

Figure 11: Central Index Algorithm.

Flooded requests model. The flooding model is different from the central index one. This is a pure P2P model in which no advertisement of shared resources occurs. Instead, each request from a peer is *flooded* (broadcast) to directly connected peers, which themselves flood their peers etc., until the request is answered or a maximum number of flooding steps (typically 5 to 9) occur (see Figure 12). This model is typically used by Gnutella and requires a lot of network bandwidth. As a result, the scalability properties of the model are often confused with its reachability properties. If the goal is to reach all the peers in the network, then this model does not prove to be very scalable, but it is efficient in limited communities such as a company network. By manipulating the number of connections of each node and appropriately setting the TTL parameter of the request messages, the flooded requests model can scale to a reachable horizon of hundreds of

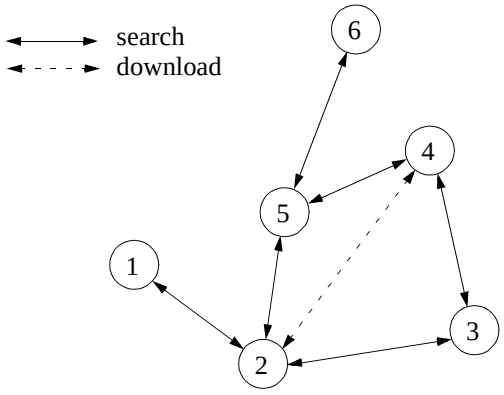

Figure 12: Flooded Requests Algorithm.

P2P System	Algorithm Comparison Criteria					
	Model	**Parameters**	**Hops to locate data**	**Routing state**	**Peer joins and leaves**	**Reliability**
Napster	centralized metadata index location inquiry from central server download directly from peers	none	constant	constant	constant	central server returns multiple download locations, client can retry
Gnutella	Broadcast request to as many peers as possible, download directly	none	no guarantee	constant (approx 3-7)	constant	receive multiple replies from peers with available data, requester can retry
Chord	uni-dimensional, circular ID space	N - number of peers in network	$\log N$	$\log N$	$(\log N)^2$	replicate data on multiple consecutive peers, app retries on failure
CAN	multidimensional ID space	N - number of peers in network d - number of dimensions	$d \cdot N^{1/d}$	$2 \cdot d$	$2 \cdot d$	multiple peers responsible for each data item, app retries on failure
Tapestry	Plaxton-style global mesh	N – number of peers in network b – base of the chosen identifier	$\log_b N$	$\log_b N$	$\log N$	replicate data across multiple peers, keep track of multiple paths to each peer
Pastry				$b \cdot \log_b N + b$	$\log N$	

Table 3. Comparison of Different P2P Location Algorithms.

thousands of nodes. Furthermore, some companies have been developing "super-peer" client software, that concentrates lots of the requests. This leads to much lower network bandwidth requirement, at the expense of high CPU consumption. Caching of recent search requests is also used to improve scalability.

Document routing model. The document routing model, used by FreeNet, is the most recent approach. Each peer from the network is assigned a random ID and each peer also knows a given number of peers (see Figure 13). When a document is *published* (shared) on such a system, an ID is assigned to the document based on a hash of the document's contents and its name. Each peer will then *route* the document towards the peer with the ID that is most similar to the document ID. This process is repeated until the nearest peer ID is the current peer's ID. Each routing operation also ensures that a local copy of the document is kept. When a peer requests the document from the P2P system, the request will go to the peer with the ID most similar to the document ID. This process is repeated until a copy of the document is found. Then the document is transferred back to the request originator, while each peer participating the routing will keep a local copy.

Although the document routing model is very efficient for large, global communities, it has the problem that the document IDs must be known before posting a request for a given document. Hence it is more difficult to implement a search than in the flooded requests model. Also, network partitioning can lead to an *islanding* problem, where the community splits into independent sub-communities, which don't have links to each other.

Four main algorithms have implemented the document routing model: Chord, CAN, Tapestry, and Pastry. The goals of each algorithm are similar: to reduce the number of P2P hops to locate a document of interest and to reduce the amount of routing state that must be kept at each peer. Each of the four algorithms either guarantee logarithmic bounds with respect to the size of the peer community, or argue that logarithmic bounds can be achieved with high probability. The differences in each approach are minimal, however each is more suitable for slightly different environments. In Chord, each peer keeps track of $\log N$ other peers (where N is the total number of peers in the community). The highly optimized version of the algorithm only needs to notify $\log N$ other peers about the join/leave events. In CAN, each peer keeps track of only a small number of other peers (possibly less than $\log N$). Only this set of peers is affected during insertion and deletion, making CAN more suitable for dynamic communities. However, the tradeoff in this case lies in the fact that the smaller the routing table of a CAN peer, the longer the length of searches. Tapestry and Pastry are very similar.

Comparison of algorithms. The Chord algorithm models the identifier space as a uni-dimensional, circular identifier space. Peers are assigned IDs based on a hash on the IP address of the peer. When a peer joins the network, it contacts a gateway peer and routes toward its successor. The routing table at each peer n contains entries for $\log N$ other peers where the *i-th* peer succeeds n

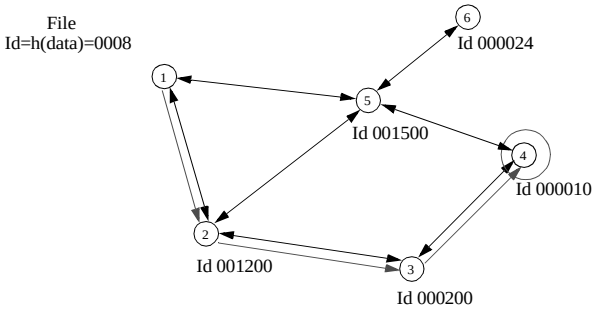

Figure 13: Document Routing Algorithm.

by at least 2^{i-1}. To route to another peer, the routing table at each hop is consulted and the message is forwarded toward the desired peer. When the successor of the new peer is found, the new peer takes responsibility for the set of documents that have identifiers less than or equal to its identifier and establishes its routing table. It then updates the routing state of all other peers in the network that are affected by the insertion. To increase the robustness of the algorithm, each document can be stored at some number of successive peers. Therefore, if a single peer fails, the network can be repaired and the document can be found at another peer.

CAN models the identifier space as multidimensional. Each peer keeps track of its neighbors in each dimension. When a new peer joins the network, it randomly chooses a point in the identifier space and contacts the peer currently responsible for that point. The contacted peer splits the entire space for which it is responsible into two pieces and transfers responsibility of half to the new peer. The new peer also contacts all of the neighbors to update their routing entries. To increase the robustness of this algorithm, the entire identifier space can be replicated to create two or more “realities”. In each reality, each peer is responsible for a different set of information. Therefore, if a document cannot be found in one reality, a peer can use the routing information for a second reality to find the desired information.

Tapestry and Pastry are very similar and are based on the idea of a Plaxton mesh. Identifiers are assigned based on a hash on the IP address of each peer. When a peer joins the network, it contacts a gateway peer and routes toward the peer in the network with the ID that most closely matches its own ID. Routing state for the new peer is built by copying the routing state of the peers along the path toward the new peer's location. For a given peer n, its routing table will contain *i* levels where the *i-th* level contains references to *b* nodes (where *b* is the base of the identifier) that have identifiers that match *n* in the last *i* positions. Routing is based on a longest suffix protocol that selects the next hop to be the peer that has a suffix that matches the desired location in the greatest number of positions. Robustness in this protocol relies on the fact that at each hop, multiple nodes, and hence multiple paths, may be traversed.

4 CHARACTERISTICS

This section addresses issues in P2P technology, which have a major impact on the effectiveness and deployment of P2P systems and applications. We also compare them in the summary, Section 4.12.

4.1 Decentralization

P2P models question the wisdom of storing and processing data only on centralized servers and accessing the content via request-response protocols. In traditional client-server models, the information is concentrated in centrally located servers and distributed through networks to client computers that act primarily as user interface devices. Such centralized systems are ideal for some applications and tasks. For example, access rights and security are more easily managed in centralized systems. However, the topology of the centralized systems may yield inefficiencies, bottlenecks, and wasted resources. Furthermore, although hardware performance and cost have improved, centralized repositories are expensive to set up and hard to maintain. They require human intelligence to build, and to keep the information they contain relevant and current.

One of the more powerful ideas of decentralization is the emphasis on the users' ownership and control of data and resources. In a fully decentralized system, every peer is an equal participant. This makes the implementation of the P2P models difficult in practice because there is no centralized server with a global view of all the peers in the network or the files they provide. This is the reason why many P2P file systems are built as hybrid approaches as in the case of Napster, where there is a centralized directory of the files but the nodes download files directly from their peers.

In fully decentralized file systems, such as Freenet and Gnutella, just finding the network becomes difficult. In Gnutella, for example, new nodes must know the address of another Gnutella node or use a host list with known IP addresses of other peers. The node joins the network of peers by establishing a connection with at least one peer currently in the network. Then, it can begin discovering other peers and cache their IP addresses locally.

The first big distributed computing project was the Great Internet Mersenne Prime Search (GIMPS, www.mersenne.org). GIMPS initially worked by email communication because email has built-in decentralization in it. When a server distributes work to computers that are unavailable, email servers will queue up email and deliver when the recipient is up. The next big distributed computing project was called distributed.net; it did encryption breaking and they initially used only a central server, which caused availability problems and did not work for several weeks in a row. They came up with a two-level proxy system, where they distribute a version of the server that can act as a proxy. In their case, task distribution was simple, because the description of work

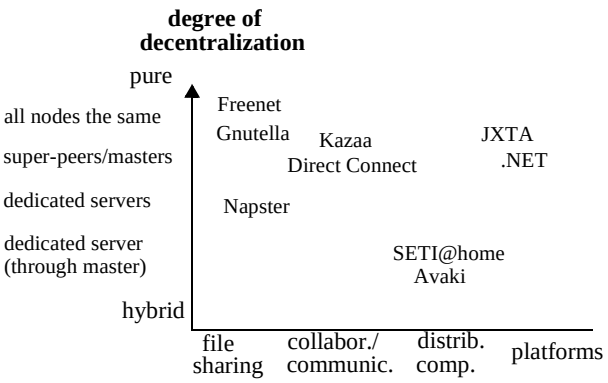

Figure 14: Examples of Levels of Decentralization in Various P2P Systems.

is small: two integers describing the range of keys to be searched.

One way to categorize the autonomy of a P2P system is through the "pure P2P" versus "hybrid P2P" distinction. A more precise decomposition may be as presented in Figure 14. This categorization has a direct effect on the self-organization and scalability of a system, as the purest systems are loosely coupled to any infrastructure.

4.2 Scalability

An immediate benefit of decentralization is improved scalability. Scalability is limited by factors such as the amount of centralized operations (e.g, synchronization and coordination) that needs to be performed, the amount of state that needs to be maintained, the inherent parallelism an application exhibits, and the programming model that is used to represent the computation.

Napster attacked the scalability problem by having the peers directly download music files from the peers that possess the requested document. As a result, Napster was able to scale up to over 6 million users at the peak of its service. In contrast, SETI@home [2001] focuses on a task that is embarrassingly parallel. It harnesses the computer power that is available over the Internet to analyze data collected from its telescopes with the goal of searching for extraterrestrial life forms. SETI@home has close to 3.5 million users so far. Systems like Avaki address scalability by providing a distributed object model.

Scalability also depends on the ratio of communication to computation between the nodes in a P2P system. Applications, such as code breaking or prime number search have the ratio close to zero, making it extremely scalable. For example, send two integers and then do the computation for two hours [Anderson 2002]. In other cases, such as SETI@home, the scalability bottleneck exists in transferring data, for example, 1MB results in 10 hours of computing, requiring bandwidth of 1MB times the number of machines that participate. Yet other applications, such as computer graphics for rendering movies, can use hundreds of thousands of computers to create a movie within hours instead of within a year. However, this application requires a lot of data movement, making it much harder to scale.

Early P2P systems such Gnutella [2001] and Freenet [Clark 2001] are ad-hoc in nature. A peer has to "blindly" send its requests to many other peers, causing the rest of the peers to search for the requested document. This can cause the time to retrieve a document to be unbounded. In addition, searching may fail even when an object exists, making the behavior of the system nondeterministic.

Recent P2P systems, represented by CAN, Chord, Oceanstore, and PAST, dictate a consistent mapping between an object key and hosting node. Therefore, an object can always be retrieved as long as the hosting nodes can be reached. Nodes in these systems compose an overlay network. Each node only maintains information about a small number of other nodes in the system. This limits the amount of state that needs to be maintained, and hence increases scalability. The logical topology of the overlay provides some guarantees on the lookup cost. Oceanstore was designed to scale to billions of users, millions of servers and over 10^{14} files [Kubiatowicz et al. 2000].

4.3 Anonymity

One goal of P2P is to allow people to use systems without concern for legal or other ramifications. Another goal is to guarantee that censorship of digital content is not possible. The authors of the Free Haven [Dingledine et al. 2001] have identified the following aspects of anonymity:

- Author: A document's author or creator cannot be identified
- Publisher: The person who published the document to the system cannot be identified
- Reader: People who read or otherwise consume data cannot be identified
- Server: Servers containing a document cannot be identified based on the document
- Document: Servers do not know what documents they are storing
- Query: A server cannot tell what document it is using to respond to a user's query

Regardless of the above entity, there are three different kinds of anonymity between each communicating pair:

sender anonymity, which hides the sender's identity; *receiver anonymity*, which hides a receiver's identity; and *mutual anonymity*, which hides the identities of the sender and receiver are hidden from each other and other peers [Pfitzmann and Waidner 1987].

Besides the kinds of anonymity, it is also very important to understand the degree of anonymity a certain technique can achieve. Reiter and Rubin [1998] presented a spectrum of anonymity degrees that cover *absolute privacy*, *beyond suspicion*, *probable innocence*, and *provably exposed*. For example, beyond suspicion means that even though an attacker can see evidence of a sent message, the sender appears no more likely to be the originator of that message than any other potential sender in the system.

There are six popular techniques, each suitable for enforcing different kinds of anonymity and with different kinds of constraints. We summarize them below.

Multicasting. Multicasting (or broadcasting) can be used to enforce receiver anonymity [Pfitzmann and Waidner 1987]. A multicast group is formed for parties who wish to keep anonymous. An entity that is interested in obtaining a document subscribes to the multicast group. The party that possesses the document sends the document to the group. The identity of the requestor is effectively hidden from both the sender and other members of the group, and the requestor's anonymity is *beyond suspicion*. This technique can take advantage of the underlying network that supports multicast (e.g., Ethernet or token ring).

Spoofing the sender's address. For connectionless protocols such as UDP, the anonymity of the sender of a message can be enforced by spoofing the sender's IP address. This however, requires changing the protocol. In addition, this is not always feasible, because most ISPs now filer packets originating from invalid IP addresses.

Identity Spoofing: Besides changing the originator's address, anonymity can also be ensured by changing the identity of a communicating party. For example, in Freenet [Clark 2001], a peer passing a file to a requestor, either out of its own cache or from an upstream peer, can claim to be the owner of the content. The responder is *possibly innocent* from an attackers's point view, because there is a nontrivial probability that the real responder is someone else.

Covert paths. Instead of communicating directly, two parties communicate through some middle nodes. Most existing techniques ensure only sender anonymity. A party that wishes to hide its identity prepares a covert path with the other party as the end of the path. Examples include Mix [Chaum 1981], Onion [Syverson 1997], Anonymizing Proxy [Gabber 1997], Crowds [Reiter and Rubin 1998] and Herdes [Shields and Levine 2000]. The covert paths can use store/forward or persistent connections. By varying the length of the covert paths and changing the selected paths with different frequency, different degrees of anonymity can be achieved.

Project	Types and Techniques of Anonymity			
	Publisher	**Reader**	**Server**	**Document**
Gnutella	multicasting, covert paths	not applicable	not applicable	not applicable
Freenet	covert path, identity spoofing	covert paths	non-voluntary placement	encryption
APFS	covert paths	covert paths	not applicable	not applicable
Free Haven	covert paths (remailer)	covert paths	broadcast	encryption/split files into shares
Publius	covert paths (remailer)	not applicable	non-voluntary placement	encryption/ split key
PAST	not applicable	not applicable	non-voluntary placement	encryption

Table 4. Types of Anonymity and Techniques to Enforce Them.

Intractable aliases. LPWA [Gabber 1999] is a proxy server that generates consistent untraceable aliases for clients from the servers. The client can open an account and be recognized upon returning to the opened account, while hiding the true identity of the client from the server. Techniques of this kind ensure sender anonymity and rely on a trusted proxy server. The degree of anonymity that can be achieved falls in between *absolute privacy* and *beyond suspicion*.

Non-voluntary placement. An interesting new approach is anonymity via non-voluntary placement of a document on a hosting node. Here, a publisher forces a document onto a hosting node using, for example, consistent hashing. Because the placement is non-voluntary, the host cannot be held accountable for owning the document.

We now summarize the forms of anonymity some of the popular P2P systems support and the techniques they employ (see Table 4).

Gnutella [2001] and Freenet [Clark 2001] provide a certain degree of anonymity in the way peers request/send a document. In Gnutella, a request is broadcast and rebroadcast until it reaches a peer with the content. In Freenet, a request is sent and forwarded to a peer that is most likely to have the content. The reply is sent back along the same path.

APFS [Scarlata et al. 2001] addresses the mutual anonymity problem assuming that trusted centralized support does not exist. Peers may inform a (untrusted) coordinator about their willingness to be index servers. Both the initiator and the responder need to prepare their own covert paths.

Free Haven [Dingledine 2001] and Publius [Waldman et al. 2000] are designed to defend against censorship. They strengthen the document's anonymity by further splitting the files as they are stored at each server. In this way, no single server even contains all of the data needed to perform an attack on the encrypted file contents. The anonymity among a pair of communicating parties (publisher/server, reader/server) is enforced via *covert paths*. Both Free Haven and Publius build the covert paths using an anonymous re-mailer system. Publius could be extended to support reader anonymity by using its re-mailer for publishing, but it does not currently do so.

PAST [Rowstron and Druschel 2001], CAN [Ratnasamy 2001] and Chord [Stoica et al 2001] represent a new class of P2P system that provides a reliable infrastructure. One common property among these systems is that object placement can be entirely non-voluntary. As a result, when an object is placed on a node, that node cannot be held accountable for owning that object. The embedded routing mechanisms in these systems can also easily be adapted to covert path for mutual anonymity.

4.4 Self-Organization

In cybernetics, self-organization is defined as "a process where the organization (constraint, redundancy) of a system spontaneously increases, i.e., without this increase being controlled by the environment or an encompassing or otherwise external system" [Heylighen 1997].

In P2P systems, self-organization is needed because of scalability, fault resilience, intermittent connection of resources, and the cost of ownership. P2P systems can scale unpredictably in terms of the number of systems, number of users, and the load. It is very hard to predict any one of them, requiring frequent re-configuration of centralized system. The significant level of scale results in an increased probability of failures, which requires self-maintenance and self-repair of the systems. Similar reasoning applies to intermittent disconnection; it is hard for any predefined configuration to remain intact over a long period of time. Adaptation is required to handle the changes caused by peers connecting and disconnecting from the P2P systems. Finally, because it would be costly to have dedicated equipment and/or people for managing such a fluctuating environment, the management is distributed among the peers.

There are a number of academic systems and products that address self-organization. In OceanStore, self-organization is applied to location and routing infrastructure [Kubiatowicz et al 2000, Rhea et al. 2001, Zhao et al. 2001]. Because of intermittent peer availability, as well as variances in network latency and bandwidth, the infrastructure is continuously adapting its routing and location support.

In Pastry, self-organization is handled through protocols for node arrivals and departures based on a fault-tolerant overlay network [Druschel and Rowstron 2001, Rowstron and Druschel 2001]. Client requests are guaranteed to be routed in less than $\lceil \log_{16} N \rceil$ steps on average. Also, file replicas are distributed and storage is randomized for load balancing.

The FastTrack product attributes quicker search and download to self-organizing distributed networks. In these networks, more powerful computers automatically become SuperNodes and act as search hubs. Any client can become a SuperNode if it meets processing and networking criteria (bandwidth and latency). The distributed networks replace any centralized service [FastTrack 2001].

SearchLing uses self-organization to adapt its network according to the type of search, resulting in reduced network traffic and less unreachable information [SearchLing 2000].

4.5 Cost of Ownership

One of the premises of P2P computing is shared ownership. Shared ownership reduces the cost of owning the systems and the content, and the cost of maintaining them. This is applicable to all classes of P2P systems. It is probably most obvious in distributed computing. For example, SETI@home is faster than the fastest supercomputer in the world, yet at only a fraction of its cost – 1% [Anderson 2002].

The whole concept of Napster music sharing was based on each member contributing to the pool of music files. Similar assumptions for peers are used in other file systems, such as OceanStore.

In P2P collaboration and communication systems, and in platforms, elimination of centralized computers for storing information also provides reduced ownership and maintenance costs. A similar approach is taken in wireless communication in the United States. A so-called "Parasitic grid" wireless movement, enables sharing of

the existing home-installed 802.11b bandwidth among the users [Schwartz 2001]. These networks compete with the companies installing wireless infrastructure at the fraction of the cost.

4.6 Ad-Hoc Connectivity

The ad-hoc nature of connectivity has a strong effect on all classes of P2P systems. In distributed computing, the parallelized applications cannot be executed on all systems all of the time; some of the systems will be available all of the time, some will be available part of the time, and some will be not be available at all. P2P systems and applications in distributed computing need to be aware of this ad-hoc nature and be able to handle systems joining and withdrawing from the pool of available P2P systems. While in traditional distributed systems this was an exceptional event, in P2P systems it is considered usual.

In content sharing P2P systems and applications, users expect to be able to access content intermittently, subject to the connectivity of the content providers. In systems with higher guarantees, such as service-level agreements, the ad-hoc nature is reduced by redundant service providers, but the parts of the providers may still be unavailable.

In collaborative P2P systems and applications, the ad-hoc nature of connectivity is even more evident. Collaborative users are increasingly expected to use mobile devices, making them more connected to Internet and available for collaboration. To handle this situation, collaborative systems support transparent delay of communication to disconnected systems. This can be accomplished by having proxies delegated on networks to receive messages, or by having other sorts of relays on the sending system or somewhere in the network that will temporarily hold communication for an unavailable system.

Furthermore, not everything will be connected to the Internet. Even under these circumstance, ad-hoc groups of people should be able to form ad-hoc networks in order to collaborate. The supporting ad-hoc networking infrastructures, such as 802.11b, Bluetooth, and infrared, have only a limited radius of accessibility. Therefore, both P2P systems and applications need to be designed to tolerate sudden disconnection and ad-hoc additions to groups of peers.

4.7 Performance

Performance is a significant concern in P2P systems. P2P systems aim to improve performance by aggregating distributed storage capacity (e.g., Napster, Gnutella) and computing cycles (e.g., SETI@home) of devices spread across a network. Because of the decentralized nature of these models, performance is influenced by three types of resources: processing, storage, and networking. In particular, networking delays can be significant in wide-area networks. Bandwidth is a major factor when a large number of messages are propagated in the network and large amounts of files are being transferred among many peers. This limits the scalability of the system. Performance in this context does not put emphasis in the millisecond level, but rather tries to answer questions of how long it takes to retrieve a file or how much bandwidth a query will consume.

In centrally coordinated systems (e.g., Napster, Seti@home) coordination between peers is controlled and mediated by a central server, although the peers also may later contact each other directly. This makes these systems vulnerable to the problems facing centralized servers. To overcome the limitations of a centralized coordinator, different hybrid P2P architectures [Yang and Garcia-Molina 2001] have been proposed to distribute the functionality of the coordinator in multiple indexing servers that cooperate with each other to satisfy user requests. DNS is another example of a hierarchical P2P system that improves performance by defining a tree of coordinators, with each coordinator responsible for a peer group. Communication between peers in different groups is achieved through a higher level coordinator.

In decentralized coordinated systems such as Gnutella and Freenet, there is no central coordinator; communication is handled individually by each peer. Typically, they use message forwarding mechanisms that search for information and data. The problem with such systems is that they end up sending a large number of messages over many hops from one peer to another. Each hop contributes to an increase in the bandwidth on the communication links and to the time required to get results for the queries. The bandwidth for a search query is proportional to the number of messages sent, which in turn is proportional to the number of peers that must process the request before finding the data.

There are three key approaches to optimize performance: replication, caching, and intelligent routing.

Replication. Replication puts copies of objects/files closer to the requesting peers, thus minimizing the connection distance between the peers requesting and providing the objects. Changes to data objects have to be propagated to all the object replicas. Oceanstore uses an update propagation scheme based on conflict resolution that supports a wide range of consistency semantics. The geographic distribution of the peers helps to reduce congestion on both the peers and the network. In combina-

tion with intelligent routing, replication helps to minimize the distance delay by sending requests to closely located peers. Replication also helps to cope with the disappearance of peers. Because peers tend to be user machines rather than dedicated servers, there is no guarantee that the peers won't be disconnected from the network arbitrary.

Caching. Caching reduces the path length required to fetch a file/object and therefore the number of messages exchanged between the peers. Reducing such transmissions is important because the communication latency between the peers is a serious performance bottleneck facing P2P systems. In Freenet for example, when a file is found and propagated to the requesting node, the file is cached locally in all the nodes in the return path. More efficient caching strategies can be used to cache large amounts of data infrequently. The goal of caching is to minimize peer access latencies, to maximize query throughput and to balance the workload in the system. The object replicas can be used for load balancing and latency reduction.

Intelligent routing and network organization. To fully realize the potential of P2P networks, it is important to understand and explore the social interactions between the peers. The most pioneering work in studying the social connections among people is the “small-world phenomenon” initiated by Milgram [1967]. The goal of his experiment was to find short chains of acquaintances linking pairs of people in the United States who did not know one another. Using booklets of postcards he discovered that Americans in the 1960s were, on average, about six acquaintances away from each other. Adamic et al. have explored the power-law distribution of the P2P networks, and have introduced local search strategies that use high-degree nodes and have costs that scale sub-linearly with the size of the network [Adamic et al. 2001]. Ramanathan et al. [2001] determine “good” peers based on interests, and dynamically manipulate the connections between peers to guarantee that peers with a high degree of similar interests are connected closely. Establishing a good set of peers reduces the number of messages broadcast in the network and the number of peers that process a request before a result is found. A number of academic systems, such as Oceanstore and Pastry, improve performance by proactively moving the data in the network (see also Section 2.6). The advantage of these approaches is that peers decide whom to contact and when to add/drop a connection based on local information only.

4.8 Security

P2P systems share most of their security needs with common distributed systems: trust chains between peers and shared objects, session key exchange schemes, encryption, digital digests, and signatures. Extensive research has been done in these areas, and we will not discuss it further in the present document. New security requirements appeared with P2P systems.

- **Multi-key encryption.** File sharing systems, such as Publius [Waldman et al. 2001], intend to protect a shared object, as well as the anonymity of its author, publishing peer and hosting peer. The security scheme chosen by Publius developers is based on a (Public key, Multiple private keys) asymmetric encryption mechanism derived from R. Shamir’s “shared secrets” encryption method [Shamir 1979]. Byzantine attacks by malicious authenticated users have typically been an issue for such schemes. Recent improvements (see [Castro and Liskov 2001] for an example) have reduced the costs inherent to Byzantine agreements and opened the way to solid systems used by large numbers of users.
- **Sandboxing.** Some distributed computing P2P systems require downloading code on peer machines. It is crucial to protect the peer machines from potentially malicious code and protect the code from a malicious peer machine. Protecting a peer machine typically involves enforcing (1) safety properties such that the external code will not crash the host box, or will only access the host data in a type-safe way, and (2) enforcing security properties to prevent sensitive data from being leaked to malicious parties. Techniques to enforce the first include sandboxing (i.e., techniques that isolate external code to its own protection domains), safe languages that prevent type-unsafe code being written (e.g., java), virtual machines (e.g., Internet C++ POSIX virtual machine, real mode Linux derivatives, which run a virtual machine on top of the actual OS, VMware), proof-carrying code and certifying compilers [Necula 1997, 1998] and program verification techniques applied to verifying the safety properties of machine code [Xu et al. 2000, 2001]. Techniques to check the latter include information flow theory [Denning 1976], and model checking [Ball 2001].
- **Digital Rights Management.** P2P file sharing makes file copying easy. It is necessary to be able to protect the authors from having their intellectual property stolen. One way to handle this problem is to add a signature *in* the file that makes it recognizable (the signature remains attached to the file contents) although the file contents do not seem affected. This technique, refer-

enced as *watermarking* or *steganography* [Katzenbeisser 1999], has been experimented with by RIAA to protect audio files such as MP3s, hiding the Copyright information in the file in inaudible ways.

- **Reputation and Accountability.** In P2P systems, it is often important to keep tack the reputation of peers to prevent ill-behaved peers from harming the whole system. Reputation requires ways to measure how "good" or "useful" a peer is. For instance, if a given user shares lots of interesting files, its reputation should be high. *Freeloader* is a common term for a user who downloads files from P2P systems without offering files to others. A freeloader usually has a low reputation. To prevent this kind of non-cooperative behavior, some accountability mechanisms need to be devised. Current systems typically rely on cross-rating among a community of *authenticated* users. However, authentication does not provide any guarantee of a peer's behavior; it is therefore difficult to produce a solid system.
- **Firewalls.** P2P applications inherently require direct connections between peers. However, in corporate environments internal networks get isolated from the external network (the Internet), leaving reduced access rights to applications. For instance, most firewalls block inbound TCP connections [p2pwg 2001]. This means that a machine within a Firewall will not be accessible from a machine external to the network. Worse, home users frequently use IP Masquerading or Network Address Translation (NAT) technology to share an internet connection between several machines, which leads to the same inaccessibility problem. However, as outbound access through port 80 (HTTP) is often allowed by firewalls, some mechanisms have been devised that enable connections between hidden (machines behind a firewall or NAT, inaccessible from the Internet) and Internet machines. This is quite limiting however, as it requires connection to be initiated from the hidden machine. When both peers who want to communicate reside behind different firewalls, the problem becomes harder. It requires a central reflector (or relay) server on the Internet, which provides a connection between the hidden peers.

4.9 Transparency and Usability

In distributed systems, transparency was traditionally associated with the ability to transparently connect distributed systems into a seamlessly local system. The primary form of transparency was *location transparency*, but other forms included transparency of *access*, *concurrency*, *replication*, *failure*, *mobility*, *scaling*, etc. [Coulouris et al. 1990]. Over time, some of the transparencies were further qualified, such as transparency for failure, by requiring distributed applications to be aware of failures [Waldo et al 1997], and addressing transparency on the Internet and Web (see next paragraph).

From its beginning, the Internet paid particular attention to transparency at the protocol level (TCP/IP), so called *end-to-end address transparency* [Carpenter 2000]. The end-to-end argument [Saltzer et al. 1984] claims that certain functions in a communication between two entities can be performed only with the knowledge and state maintained at these entities at the application level (hence, end-to-end: all the way from application through the communication stack, up to the other application). This implies that application communication state is not maintained in the network, but rather at the communication end points.

This also implies that any point in the network knows the name and address of the other communication point, an assumption that is not true any more. IPv4's lack of IP numbers, as well as the introduction of intranets and mobile users, resulted in IP numbers that are valid only during a single session. Examples include SLIP, PPP, VPNs, use of firewalls, DHCP, private addresses, network address translators (NATs), split DNS, load sharing optimizations, etc [Carpenter 2000]. This had significant implications for P2P and was also one of the reasons for the introduction of P2P. Because it was no longer possible to rely on DNS to provide an accurate name, P2P systems came up with different naming and discovery schemes (see Section 3.1 as well as end of Section 5.3).

Web naming did not necessarily offer full naming transparency. URLs are widely used instead of URNs, which were supposed to enable *naming transparency* [Berners-Lee et al. 1998]. Beside *naming/addressing transparency* in P2P there is also a requirement for *administration transparency*. Users are typically non-experts and they do not or cannot administer their software and devices.

The P2P software should not require any significant set up or configuration of either networks or devices in order to be able to run. Also, self-updating software is a desirable feature. In addition, P2P systems should be network and device transparent (independent). They should work on the Internet, intranets, and private networks, using high-speed or dial-up links. They should also be device transparent, which means they should work on a variety of devices, such as handheld personal digital assistants (PDAs), desktops, cell phones, and tablets.

Another form of transparency is related to security and mobility. Automatic and transparent authentication of users and delegation to user proxies can significantly

simplify users' actions. Supporting mobile users and disconnection in particular, can enable users to work independently of whether and how they are connected to the Internet or intranets.

P2P applications can be used in the following manners:

- as a user of services, typically through Web interfaces (e.g., content sharing, information gathering)
- wrapped around non-P2P applications, typically on a P2P platform (e.g., Groove, .NET)
- as locally installed P2P software (e.g., distributed computing screensavers and Napster)

4.10 Fault Resilience

One of the primary design goals of a P2P system is to avoid a central point of failure. Although most P2P systems (pure P2P) already do this, they nevertheless are faced with failures commonly associated with systems spanning multiple hosts and networks: disconnections/ unreachability, partitions, and node failures. These failures may be more pronounced in some networks (e.g., wireless) than others (e.g., wired enterprise networks). It would be desirable to continue active collaboration among the still connected peers in the presence of such failures. An example would be an application, such as genome@home [2001] executing a partitioned computation among connected peers. Would it be possible to continue the computation if one of the peers were to disappear because of a network link failure? If the disconnected peer were to reappear, could the completed results (generated during the standalone phase) be integrated into the ongoing computation? Questions similar to these would have to be addressed by P2P systems aiming to provide more than just "best effort" Internet service.

In the past, client-server disconnection has been studied for distributed file systems that consider mobile clients (e.g., Coda [Satyanarayanan 1990]), and a common solution is to have application-specific resolvers to handle any inconsistency on reconnection. Some current P2P systems (e.g., Groove [Groove Networks 2001]) handle this by providing special nodes, called relays, that store any updates or communication temporarily until the destination (in this case another peer) reappears on the network. Others (e.g., Magi [Endeavors Technologies 2001]) queue messages at the source, until the presence of the destination peer is detected.

Disconnection can be result of non-availability of resources. This may occur either because the resource is unreachable due to a network failure or because the peer hosting the resource has crashed/gone offline. While the former may be resolved by routing around the failure and is already supported by the Internet, the latter requires more careful consideration. Replication of crucial resources helps alleviate the problem. P2P networks such as Napster and Gnutella represent systems having both a passive and an uncontrolled replication mechanism based solely on the file's popularity. Depending on the application running over these networks, it may be necessary to provide certain persistence guarantees. This requires a more active and reliable replication policy.

Anonymous publishing systems such as Freenet [Clark et al. 2000] and Publius [Waldman et al. 2000] ensure availability by controlled replication. Oceanstore [Kubiatowicz et al. 2000] maintains a two-layered hierarchy of replicas and through monitoring of administrative domains avoids sending replicas to locations with highly correlated probability of failures. However, because a resource in the P2P system could be more than a just a file – such as a proxy to the Internet, shared storage space, or shared computing power – the concepts of replicated file systems have to be extended to additional types of resources. Grid computing solutions (e.g. Legion [Grimshaw et al. 1997]) provide resilience against node failures by restarting computations on different nodes.

A challenging aspect of P2P systems is that the system maintenance responsibility is completely distributed and needs to be addressed by each peer to ensure availability. This is quite different from client-server systems, where availability is a server-side responsibility.

4.11 Interoperability

Although many P2P systems already exist there is still no support to enable these P2P systems to interoperate. Some of the requirements for interoperability include:

- How do systems determine that they can ineroperate
- How do systems communicate, e.g., what protocol should be used, such as sockets, messages, or HTTP
- How do systems exchange requests and data, and execute tasks at the higher level, e.g., do they exchange files or search for data
- How do systems determine if they are compatible at the higher protocol levels, e.g., can one system rely on another to properly search for a piece of information
- How do systems advertise and maintain the same level of security, QoS, and reliability

In the past, there were different ways to approach interoperability, such as *standards* (e.g., IEEE standards for Ethernet, token ring, and wireless); *common specifications*, (e.g., Object Management Group [OMG 2001]); *common source code*, (e.g., OSF DCE [Rosenberrg

1992]); *open-source* (e.g., Linux); and *de facto standards* (e.g., Windows or Java).

In the P2P space, some efforts have been made towards improved interoperability, even though interoperability is still not supported. The P2P Working Group [p2pwg 2001] is an attempt to gather the community of P2P developers together and establish common ground by writing reports and white papers that would enable common understanding among P2P developers. The P2P Working Group gathers developers from both ad-hoc communication systems and grid systems. The Grid Forum [2002] is a similar effort in the grid computing space. Both efforts represent an approach similar to OMG, in defining specifications and possibly reference implementations.

JXTA [2001] approaches interoperability as an open-source effort, by attempting to impose a de facto standard. A number of developers are invited to contribute to the common source tree with different pieces of functionality. Only a minimal underlying architecture is supported as a base, enabling other systems to contribute parts that may be compatible with their own implementations. A number of existing P2P systems have already been ported to the JXTA base. JXTA is described in more detail in Section 6.7.

4.12 Summary

Decentralization is a key feature of P2P systems. It affects how developers design systems and applications, by influencing algorithms, data structures, security, scalability, and availability assumptions. It affects how users can be connected to a system and the people with whom they can interact. For example, in games, users perceive that other players are remote, and that they can also be disconnected. This implies that they should devise strategies in a decentralized fashion. Distributed P2P applications are written assuming decentralization, and collaborative applications have to handle group management without central naming, authorization, and data repositories.

The ad-hoc nature of P2P systems also affects the way applications and systems are conceived. The fact that any system or user can disappear at time drives the design of these systems as well as user perceptions and expectations. In addition to the classical security issues of traditional distributed systems, P2P is distinguished by the importance of anonymity in certain applications and markets. Scalability, performance, fault resilience, and interoperability have similar importance for P2P as they have in traditional distributed systems.

Distributed computing applications are primarily concerned with scalability (which is derived from decentralization) and performance. Fault resilience is tied in with the ad-hoc nature of connectivity – distributed application designers and users need to account for the possibility of a system going down at any time and being able to resume from a previous checkpoint. P2P systems that use critical or confidential data are concerned with security. Content sharing applications and systems are primarily concerned with the availability of data. Enterprise systems are concerned with security and performance, and public and consumer systems are concerned with transparency (ease of use) and anonymity. Collaborative and communication applications are concerned with connectivity (ad-hoc nature), security, and anonymity, and with interoperability between systems.

5 CATEGORIES OF P2P SYSTEMS

This section describes in more detail the four categories presented in Figure 6: distributed computing, file sharing, collaborative systems, and P2P platforms. In addition, we also present historical P2P systems that predated the recent notion of P2P systems. While this section presents the P2P systems categories in general, Section 6 presents each of the categories in more detail with two case studies.

5.1 Historical Systems

In most cases, early distributed applications were P2P. When most users were from a technical or academic background, and were using either time-sharing systems or engineering workstations, P2P applications seemed the most natural approach. It was the mid-80's to early-90's when client-server architectures became more prevalent because they provided the fastest and most cost-effective means of supporting large numbers of non-technical users. It also allowed the use of less expensive and less functional computers such as desktop PCs.

While most early distributed applications can be considered P2P, e-mail systems built on the Simple Mail Transfer Protocol (SMTP) and Usenet News were probably the most widely used. In each case, local servers that received a message built connections with peer servers to deliver messages into a user's mail file or into a spool file containing messages for the newsgroup. The File Transfer Protocol (FTP) was the precursor to today’s file sharing P2P systems. While FTP is a client-server application, it was very common for individuals to run FTP servers on their workstations to provide files to their peers. Eventually, an indexing system, Archie, was developed to provide a central search mechanism for files on FTP servers. This structure with central search and

distributed files is exactly replicated in the Napster P2P system.

Prior to the establishment of a continuously connected network such as the Internet, decentralized dial-up networks were widely used. The most notable examples include UUNet and Fidonet. These networks were composed of a collection of machines that made periodic dial-up connections to one another. On a typical connection, messages (typically e-mail or news articles) were transferred bi-directionally. Often, a message would be routed through multiple dial-up hops to reach its destination. This multi-hop message routing approach can be seen in current P2P systems such as Gnutella.

In our "modern" era dominated by PCs on workplace LANs or home dial-up connections, the first wider use of P2P seems to have been in instant messaging systems such as AOL Instant Messenger. These are typically hybrid P2P solutions with discovery and brokering handled by a central server followed by direct communication between the peer messaging systems on the PCs. The current phase of interest and activity in P2P was driven by the introduction of Napster [2001] in 1999. It came at a time when computers and their network connections were nearing the level found previously in technical and academic environments, and re-created earlier approaches with an interface more suitable to a non-technical audience.

5.2 Distributed Computing

The idea of using spare computing resources has been addressed for some time by traditional distributed computing systems. The *Beowulf* project from NASA [Becker et al. 1995] was a major milestone that showed that high performance can be obtained by using a number of standard machines. Other efforts, such as *MOSIX* [Barak and Litman 1985, Barak and Wheeler 1989] and *Condor* [Litzkow et al 1988, Litzkow and Solomon 1992], also addressed distributed computing in a community of machines, focusing on the *delegation* or *migration* of computing tasks from machine to machine.

Grid computing is another concept that was first explored in the 1995 I-WAY experiment [Foster 2000], in which high-speed networks were used to connect high-end resources at 17 sites across North America. Out of this activity grew a number of Grid research projects that developed the core technologies for "production" Grids in various communities and scientific disciplines. Grid technology efforts are now focused around the Global Grid forum (http://www.globalgridforum.org) and the Globus project (http://www.globus.org). A computing grid can be seen and used as a single, transparent computer. A user logs in, starts jobs, moves files, and receives results in a standard way.

Derivatives of Grid Computing based on standard Internet-connected PCs began to appear in the late 90's. They achieve processing scalability by aggregating the resources of large number of individual computers. Typically, distributed computing requires applications that are run in a proprietary way by a central controller. Such applications are usually targeting massive multi-parameter systems, with long running jobs (months or years) using P2P foundations. One of the first widely visible distributed computing events occurred in January 1999, where distributed.net, with the help of several tens of thousands of Internet computers, broke the RSA challenge [Cavallar et al. 2000] in less than 24 hours using a distributed computing approach. This made people realize how much power can be available from idle Internet PCs.

More recent projects have been raising interest from many users within the Internet community. For example, SETI@home [2001] now has a consolidated power of about 25 TFlops/s (trillions of floating point operation per second), collected from more than three million registered user machines.

Peer-to-Peer? A commonly raised question is if distributed computing systems such as SETI@home are P2P? The counter-argument is that a central server is required for controlling the resources, the PCs do not operate as servers, and no communication occurs between peers. The pro-argument is that a significant part of the system is executed on the PCs, with high autonomy. We consider such distributed computing systems to be P2P.

How it works. The computational problem to be solved is split into small independent parts. The processing of each of the parts (using a *fork-and-join* mechanism) is done on an individual PC and the results are collected on a central server. This central server is responsible for distributing job items to PCs on the Internet. Each of the registered PCs is equipped with client software. The client software takes advantage of inactivity periods (often characterized by screensaver activation times) in the PC to perform some computation requested by the server. After the computation is finished, the result is sent back to the server, and a new job is allocated to the client.

Current usage. One of the major limitations of distributed computing is that it requires jobs that can be split into *independent* small parts that do not require (significant) cross-peer communication. Internet latencies do not allow demanding communication patterns such as

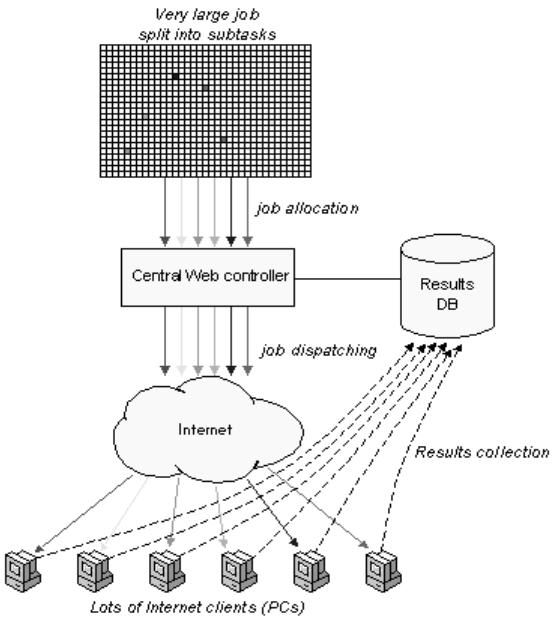

Figure 15: Distributed computing over the Web.

those found in typical cluster computing. Hence, it is not possible to execute supercomputing-like processing such as linear algebra problems (matrix computation). Current applications consist of Single Process Multiple Data (SPMD) problems and multiprogramming problems where a given job has to be run on many different input data sets. This mostly includes simulation and model validation tasks. Because specific applications have to be developed using very specific constraints, paradigms, and environments, their development cost is prohibitive to most users, and the scope remains limited to highly visible research domains, such as the human genome project, SETI, cancer research, and weather model validation.

Application area examples – Financial and Biotechnology. Some financial and biotechnology applications are suitable for distributed computing. Financial institutions, such as banks and credit companies, are executing complex simulations for market calculations. Applications include portfolio pricing, risk hedge calculation, market and credit evaluation, counter-party netting, margin calculations, cash flow, and demographic analysis [Intel 2001]. In the past, financial applications typically ran during the night. As they become more real-time in nature, these requirements will grow even further. So far only big institutions have been able to afford the computing power to automate these simulations, typically using mainframe computers or very powerful workstations. By relying on P2P commodity platforms, smaller banks can also benefit from these applications. Furthermore, as technology favors farms of desktop computers, they become not only more cost effective, but also a more powerful platform. As an example, Intel and DataSynapse claim speed-ups from 15 hours to 30 minutes in the case of interest rate swap modeling when moving to a P2P solution [DataSynapse 2001]. The biggest challenge in using P2P for financial markets is the intrinsic requirement for security. Because most of these applications are executing behind the firewall, security requirements are somewhat relaxed. In particular, parts of the applications can even be executed outside of the firewall.

In the biotechnology sector, the need for advanced computing techniques is being driven by the availability of colossal amounts of data. For instance, genomic research has close to three billion sequences in the human genome database. Applying statistical inference techniques to data of this magnitude requires unprecedented computational power. Traditionally, scientists have used high-performance clustering (HPC) and supercomputing solutions, and have been forced to employ approximating techniques in order to complete studies in an acceptable amount of time. By harnessing idle computing cycles (95%-98% unused) from general purpose machines on the network, and grouping multi-site resources, grid computing makes more computing power available to researchers. Grid solutions partition the problem space among the aggregated resources to speed up completion times. Companies such as Platform Computing (LSF) [Platform Computing 2001, Zhou et al. 1994], Entropia [2001], Avaki [2001] and Grid Computing Bioinformatics [2001] offer complete HPC and grid solutions to biological research organizations and pharmaceutical research and development. Genomics and proteomics projects such as Genome@home [2001] and Folding@home [2001] managed by groups at Stanford make use of the idle cycles of registered clients to compute parts of the complex genome sequencing and protein folding problems [Natarajan 2001].

In search for a business model. Distributed computing requires an expensive infrastructure to host the Web service, and this does not even account for development and maintenance costs. Attempts have been made by several companies (such as Porivo) to pay the user for use of his machine by the system. If your machine computes a lot of jobs, you can get some money for having shared your machine. Other models are based on auctions, where the cheapest available CPU is allocated for a given job. Another model used by Entropia addresses vertical markets, such as Genomics, which is based on huge database look ups. However, none of the existing business models have yet proven successful. The only model that currently seems to work is the dedicated community model, such as astronomy researchers sharing their machines in a "friendly" way.

5.3 File Sharing

Content storage and exchange is one of the areas where P2P technology has been most successful. Multimedia content, for instance, inherently requires large files. Napster and Gnutella have been used by Internet users to circumvent bandwidth limitations that make large file transfers unacceptable. Distributed storage systems based on P2P technologies are taking advantage of the existing infrastructure to offer the following features.

- *File exchange areas.* Systems such as Freenet provide the user with a potentially unlimited storage area by taking advantage of redundancy. A given file is stored on some nodes in the P2P community, but it is made available to any of the peers. A peer requesting a given file just has to know a reference to a file, and is able to retrieve the file from the community by submitting the file reference. Systems such as Freenet, Gnutella, and Kazaa fall in this category.
- *Highly available safe storage.* The duplication and redundancy policies in some projects offer virtual storage places where critical files get replicated multiple times, which helps ensuring their availability. *OceanStore* [Kubiatowicz et al. 2000] and *Chord* [Dabek et al. 2000] are examples of such systems.
- *Anonymity.* Some P2P systems such as Publius [Waldman et al. 2000] mathematically ensure that published documents preserve anonymity for their authors and publishers, while allowing people to access the documents.
- *Manageability.* P2P systems enable easy and fast retrieval of the data by distributing the data to caches located at the edges of the network. In some systems, the location of the data is not known by the retriever, perhaps not even after the data is retrieved. Freenet, for example, stores the data in many locations in the path between the provider and the retriever, so the whole notion of hosting a file becomes meaningless. Files move freely among the peers and are allowed to disappear even if they are being downloaded. This has some important implications. For example, the question is who is accountable for the files (see Section 4.5). Also, how can we ensure that the entire piece of data is being downloaded and cope with the un-reliability of the peers (see Section 4.10).

Technical issues. The major technical issues in file sharing systems are mostly the network bandwidth consumption, security, and search capabilities. Three main models exist today, as discussed in Section 3.2: *Centralized directory* model (Napster), the *flooded request* model (Gnutella), and the *document routing* model (FreeNet). All P2P file sharing systems can be categorized into one of these three families, although variations do exist, such as extensions of the models with *leader election mechanisms* (automatic or manual elections) such as in *KaZaA*, which allow for better scalability of the systems and less stress on the network. In addition, current P2P systems (e.g., Napster, Gnutella, Freenet) have mainly focused on the exchange and sharing of "small" objects such as files and music clips. However, we expect that in future P2P systems the content will be of any form, including audio, video, software, and documents. To do that, we will need intelligent decisions such as from where the content be retrieved and over which network path should the content travel. XDegrees [XDegrees 2001], for example, ensures that information is efficiently routed to users and that multiple document replicas are synchronized across many peers. They provide an eXtensible Resource Name System (XRNS) based on the notion of the URL that creates a location-independent name space. They place frequently accessed information at optimal locations in the network and then select the best route to retrieve that information based on source availability, network topology, and response time.

Application area examples. Napster is first P2P file sharing application to receive widespread use. Napster was originally developed to defeat the copying problem and to enable the sharing of music files over the Internet. Napster uses the centralized directory model (see Section 2.6) to maintain a list of music files, where the files are added and removed as individual users connect and disconnect from the system. Users submit search requests based on keywords such as "title," "artist," etc. Although Napster's search mechanism is centralized, the file sharing mechanism is decentralized; the actual transfer of files is done directly between the peers. Napster's centralized directory model inevitably yields scalability limitations (see Section 3.2). For example, the available bandwidth can be tremendously reduced by users downloading songs from one's machine. Yet, centralization simplifies the problem of obtaining a namespace and enables the realization of security mechanisms (see Section 3.8).

Napster has been quite popular. It has had more than forty million client downloads and has led to numerous variants of file-sharing applications (such as Gnutella and Pointera). OpenNap (http://opennap.sourceforge.net/) is an open-source Napster server that extends the Napster protocol to allow sharing of any media type and add the ability to link Napster servers together. To solve the problem with copyright violations, Napster relaunched a new Napster membership service. Although it still offered the core functions (searching, finding, shar-

ing, and discovering digital music through the Napster community) it also offered to artists the opportunity to register as rights holders and get paid for sharing their music on Napster. Napster set the rules for how their music files are used.

Morpheus is a full-featured P2P file-sharing system introduced by MusicCity (www.comusiccity.com) that tries to overcome some of the limitations of Napster. It includes a powerful search engine that can search for all types of media (including music, videos, documents, and reference files). The results are grouped together so the same file is displayed only once. Its SmartStream mechanism automatically resumes broken content streams by finding another source for the same content and monitoring the network until the whole content stream is downloaded. Morpheus increases the download speed of large files through the simultaneous transfer of content from multiple sources (FastStream mechanism). Its encryption mechanisms protect privacy and transmissions, and prevent unauthorized intrusions. Also, it allows content providers to deploy third-party digital rights management technology to protect the copyrights of their digital content that is distributed through the network.

Kazaa is another example of a P2P file sharing system that uses SuperNodes as local search hubs. These are powerful nodes on fast connections that are generated automatically. Peers connect to their local SuperNode to upload information about the files they share, and to perform searches in the network. Kazaa uses an intelligent download system to improve download speed and reliability. The system automatically finds and downloads files from the fastest connections, failed transfers are automatically resumed, and files are even downloaded from several sources simultaneously to speed up the download. When files are imported, the system automatically extracts meta-data from the contents of the files (such as ID3 tags for mp3 files). This makes for much faster and more accurate searches.

5.4 Collaboration

Collaborative P2P applications aim to allow application-level collaboration between users. The inherently ad-hoc nature of P2P technology makes it a good fit for user-level collaborative applications. These applications range from instant messaging and chat, to online games, to shared applications that can be used in business, educational, and home environments. Unfortunately, a number of technical challenges remain to be solved before pure P2P collaborative implementations become viable.

Overview. Collaborative applications are generally event-based. Peers form a group and begin a given task. The group may include only two peers collaborating directly, or may be a larger group. When a change occurs at one peer (e.g., that peer initiates sending a new chat message), an event is generated and sent to the rest of the group. At the application layer, each peer's interface is updated accordingly.

Technical Challenges. There are a number of technical challenges that make implementation of this type of system difficult. Like other classes of P2P systems, *location* of other peers is a challenge for collaborative systems. Many systems, such as Magi, rely on centralized directories that list all peers who are online. To form a new group, peers consult the directory and select the peers they wish to involve. Other systems, like Microsoft's NetMeeting, can require that peers identify one another by IP address. This is much too restrictive, especially in environments where groups are large.

Fault tolerance is another challenge for collaborative systems. In shared applications, messages often must be delivered reliably to ensure that all peers have the same view of the information. In some cases, message ordering may be important. While many well-known group communication techniques address these challenges in a non-P2P environment, most P2P applications do not require such strict guarantees. The primary solution employed in P2P applications is to queue messages that have been sent and not delivered (i.e., because a given peer is down or offline). The messages can then be delivered to the offline peer when it comes back online.

Realtime constraints are perhaps the most challenging aspect of collaborative implementations. Users are the ultimate end points in a collaborative environment. As such, any delay can be immediately perceived by the user. Unfortunately, the bottleneck in this case is not the P2P technology, but the underlying network. While many collaborative applications may work well in a local-area systems, wide-area latencies limit P2P applications just as they limit client-server applications. Consider a gaming environment. The game DOOM is a so-called First Person Shooter (FPS) game in which multiple players can collaborate or compete in a virtual environment. DOOM uses a P2P structure in which each player's machine sends updates of the state of the environment (such as the player's movement) to each of the other machines. Only when all updates have been received does the game update the view. This was marginally viable in local-area, small-scale games, but did not scale to wide-area games. Long latencies and uneven computing power at the various players machines made this lock-step architecture unusable. All FPS games since DOOM have used a more standard client-server architecture for communication.

5.5 Platforms

Operating systems are becoming decreasingly relevant as environments for applications. Middleware solutions, such as Java Virtual Machines, or Web browsers and servers are the dominant environment that is of interest to users as well as to developers of applications. In that regard, it is likely that future systems will increasingly depend on some other sort of platform that will be a common denominator for users and services connected to the Web or in an ad-hoc network. Examples of such environments include AOL and Yahoo, and .NET is striving toward a similar goal.

As described in Section 3.1, platforms, even more so than other P2P systems, have support for primary P2P components: naming, discovery, communication, security, and resource aggregation. They have an OS dependency even though it is minimal. Most P2P systems are either running on an open-source OS (Linux) or they are based on Windows.

There are a number of candidates competing for future P2P platform. .NET is the most ambitious one, going beyond P2P to encompass all service support on the client and server side. JXTA is another attempt, taking a bottom up and strong interoperability approach. Most other systems also have some level of platform support, such as Groove covering the enterprise domain and Magi, covering the handheld devices domain. Section 6.7 and Section 6.8 contain detailed descriptions of JXTA and .NET respectively.

6 CASE STUDIES

In this section, we compare eight case studies of P2P systems. We selected systems in four different categories, representing the spectrum of different P2P system categories, as well as public domain and proprietary systems (see Table 5).

P2P system	Developer	Technology
Avaki	Avaki	distributed computing
SETI@home	public domain	
Groove	Groove Networks	collaboration
Magi	Endeavors Technologies	
FreeNet	public domain	content distribution
Gnutella	public domain	
JXTA	public domain	platform
.NET	Microsoft	

Table 5. Case Studies for Eight P2P systems.

6.1 Avaki

Avaki provides a single virtual computer view of a heterogeneous network of computing resources. It is a classic example of meta-computing applied to networks ranging from corporate compute and data grids to global application grids of Internet scale.

History. Avaki began as Legion [Grimshaw et al. 1994, Grimshaw et al. 1997], a research project initiated at the University of Virginia in 1993. Passing through various stages of development, and following the first showcase of the Legion technology in 1997 at the Supercomputing conference, the research project emerged as a commercial venture called Applied MetaComputing [2000] in 1998. In mid 2001, it was re-launched as Avaki Corporation, which currently focuses on providing enterprise-level distributed computing solutions.

Goals. The goal is to create one or more grids - the purpose of which is to facilitate collaboration via the creation of virtual organizations and the sharing of resources. Resources are as diverse as computational cycles, data, storage, applications, and specialized devices. Other goals include scalability, robust security, performance management, and failure detection and recovery features to radically simplify grid administration. There is also a focus on integration with existing systems and applications. Avaki is an integrated system, not a toolkit.

Design. Avaki is object-based. Every entity in the system is an individually addressable object with a set of interfaces for interaction. This approach enables uniformity in system design through inheritance and containment through encapsulation. A lot of lessons learned from Mentat [Grimshaw 1993], an object-oriented parallel processing system, were applied to Avaki. High performance, exploitation of heterogeneity, scalability, and masking of complexity were key design criteria in the development of Avaki. The middleware is structured as a layered virtual machine as shown in Figure 16. The stack has three main components.

- *Core Services* provide the basic functionality required to map the system to a network of computing resources. The meta-data and the directory services enable locating files or computing resources.
- *System Management Services* allow the Avaki system to monitor and control the efficiency of the meta-system. The policy management service allows administrators to manage access control for local resources.

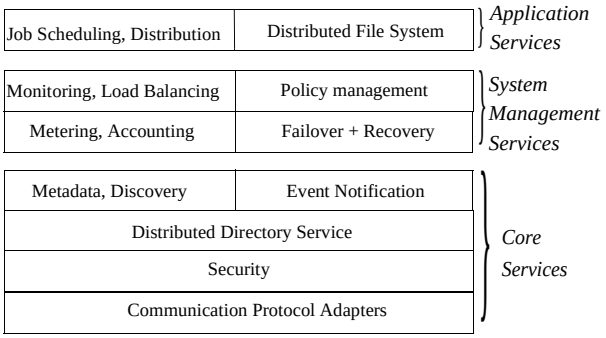

Figure 16: Avaki Architecture.

- *Application Services* are built over the basic services. They enable the construction of applications, such as collaborative and high-performance computing.

Decentralization. Avaki naming and binding are designed to support decentralization. Avaki objects interact with each other in a P2P fashion. They communicate with objects essentially on their own., rather than going through intermediaries. The set of different objects that interact over time may change very rapidly, communication pattern is highly dispersed [Grimshaw 2002].

Cost of ownership. A lot of effort has been made to avoid rewriting applications and to use legacy applications, as well as to reduce the startup cost of using a Grid. There is an initial cost to bring up Grid, which consists of the administrative time to set it up (typically 20 minutes) and the additional cost to bring up applications. There is a marginal complexity of adding a new node. Restarting Grid is automatic. Adding a new organization into a grid requires the manual addition of a local identity into a global identity structure.

Scalability. Avaki is built as a layered virtual machine with scalable, distributed control. It allows for multiple administration domains, allowing system administrators complete control over local resources and their access policies. To make the file system scalable and to not impose a single file system across all machines, Avaki takes a federated file system approach and builds a unified file system over existing ones.

Fault resilience. The scale of the Avaki network enables high redundancy of hardware components. Legion trades off fault tolerance support for higher performance. The scope of the support is limited to stopping failures of hardware components rather than software and Byzantine faults or providing transactional stopping recovery. However, there is a large degree of failure recovery and failure resilience in both the data and computation grid. In the event of internal failures during program execution, the user is notified and can decide on appropriate steps. When a host goes down, Avaki dynamically reconfigures the grid and automatically migrates failed jobs to different locations.

Transparency. Avaki supports location, access, migration, failure, heterogeneity, and transparency. All objects have a global name space and either end-point of a communication link can move around, fail, restart, and recover. Failures and migration take place at the system level; applications are not aware of it [Grimshaw 2002].

Implementation and performance. Avaki gains significantly by the concurrent execution of applications. Applications submitted to the system are executed concurrently on a pool of available computing resources. Avaki came from the high-performance computing community in LANs and tried to keep its properties in WANs, by caching aggressively at the edges of the network. Avaki performance is fundamentally limited by the network's characteristics. Running Avaki over a 56Kb link is not recommended; T1 connection is the entry point communication link [Grimshaw 2002].

Security. Avaki has built-in security, which eliminates the need for any other software-based security controls. Its security is decentralized allowing individual administration domains to control access to local resources. Authentication by users occurs once during sign-in, and Avaki manages all further authentication required to run tasks over resources spread across multiple administration domains. Avaki emphasizes multi-organizational support for industrial rather than consumer applications. Security is policy-neutral, giving Avaki the ability to form virtual organizations in the grid, and limiting applications only to a particular set of resources. Because organizations change frequently, Avaki was designed to put a set of people to work together in a highly secure way and to manage their privileges efficiently and quickly. Companies want to interact with one another and access each other's files without giving remote accounts. They want access that is controlled and auditable [Grimshaw 2002].

Interoperability. Avaki supports servers for the NFS 2 and NFS 3 protocols. The server maps the Avaki global name space onto the local file system, enabling NFS clients to use data without knowing that there is a grid behind it. This offers interoperability with a lot of operating systems and applications. Another form of interoperability is expressed through planned support for the Global Grid Forum's (GGF) Open Grid Service Architecture (OGSA), which is similar to the Legion and Avaki model [Grimshaw 2002].

Lessons learned. Handling heterogeneity and automatic error detection and recovery are key to the success of

systems of this scale. Masking failure in a performance-effective way is a challenge. All this contributes to reducing the activation effort, either the initial effort or after a failure, which is important for grid users. Data grids are raising more interest among the enterprise users than compute grids. They need to integrate data access across a global organization, with smart caching and all data mapped to the local file system. The same applies for relational data bases. A lot of large-scale companies have sites across the world, and there is more need for inter-organizational data grids than for multi-organization grids, across company boundaries. There are two reasons why the Andrew File System (AFS) has not caught on in data Grids. First, Kerberos authentication is required across the whole data grid, which is cumbersome on an organization by organization basis, and it imposes the same trust model. Second, the whole grid would have to support the AFS model, departing from deployed operating environments, such as NFS, Samba, hierarchical storage management, or Avaki. In other words, a federating model is more needed than a unifying model for the file system or authentication, reducing the cost of bringing in the data grid [Grimshaw 2002].

Business model. Avaki is marketing its product to enterprises as a complete data and compute grid. However, it faces competition from established HPC solution vendors such as Platform Computing. Avaki currently has customers in the domain of drug discovery.

Applications. In addition to enterprise solutions, Avaki supports high-end computing power for problems in high-performance computing. Applications that involve parameter-space studies can benefit from Avaki's high-performance parallel execution characteristics. Examples include biochemical simulations; BLAST, a protein and DNA sequence comparator; and CHARMM, a parallel MPI 3D program [Natarajan 2001].

6.2 SETI@home

SETI (Search for Extraterrestrial Intelligence) is a collection of research projects aimed at discovering alien civilizations. One of these projects, SETI@home analyzes radio emissions received from space and collected by the giant Arecibo telescope, using the processing power of millions of unused Internet PCs [Anderson 2002].

History. The SETI@home project was formed in 1995, it began development in 1998, public launch in April 1999. The project has been widely accepted and created a raw processing power of several dozens of TFlops from more than three million Internet-connected computers.

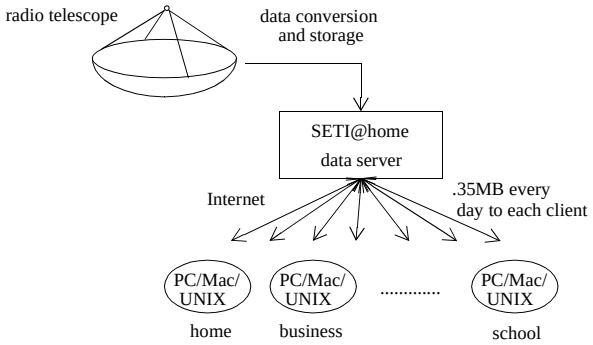

Figure 17: SETI@home Architecture. *Adapted from [Sullivan et al. 1997].*

Goals. The goal is to search for extraterrestrial radio emissions. SETI@home is a scientific research project aimed at building a huge virtual computer based on the aggregation of the compute power offered from internet-connected computers during their idle periods. Two important goals of SETI@home are to demonstrate the viability of public-participation distributed scientific computing, and to increase public knowledge and awareness of SETI and radio astronomy [Anderson 2002].

Design. The project uses two major components: the data server and the client. Clients exist as a screen saver program (although stand-alone operation is available) for several different platforms, such as Windows, Macintosh, Linux, Solaris, and HP-UX (see Figure 17). Some application-specific programming was needed for Mac and Windows, but all other platforms share the same code which uses POSIX and X11.

Decentralization: SETI@home distributes files (350KB large) to its users, requiring a high-speed interconnect at the server end. Currently 30Mb/s is sent 24h/day. For reasons of simplicity, the original design was centralized. Except for the computation itself, there was just one server that was distributing data, and it turned out to be a problem in terms of reliability and scalability. There were long periods when there were problems with connecting to the main server because the University of California network connection was totally saturated. SETI@home clients were not able to connect except in the middle of the night. SETI@home users reverse engineered the protocol between the SETI@home client and server and invented proxy servers. One of them, SETI-queue, acts as a buffer by obtaining the tasks from the main server. Clients can be configured to interact with the proxy server instead of the main server. Proxy servers were able to connect during the middle of the night and distribute during the day.

Scalability. The SETI@home project relies on its server to distribute jobs to each participating peer and to collect results after processing is done, using one of the UC Berkeley 70 Mb/s connections. This turned out to be one of the bottlenecks for scalability. One solution could be to obtain another 100 Mb/s connection, but this is expensive. Another approach is to distribute the bandwidth in the same way they are distributing computation. SETI@home could mail the data on magnetic tape or disks to organizations willing to donate bandwidth. Then the data could be distributed to the end users through the Internet connections donated by the organization. Currently, the SETI@home software architecture does not allow this approach. The existing protocol assumes that the data and control all go through the same server. SETI@home is generalizing software so that the data topology is arbitrary and the network bandwidth will be spread across different connections [Anderson 2002]. There are also a lot of traditional scalability issues in terms of the basic capacity of the servers. Million of computers are running the program at any given time and each of them can try to connect to the server at any time. There is a heavy load that has increased exponentially over time with the Moore's law, as well as linearly with the increased number of users. As a result it was necessary to constantly upgrade the server machines and the database. Approximately half of the project's man power has gone into the system administration of increasing the server capacity [Anderson 2002].

Anonymity. One of the important things to SETI@home users is that they receive credit for what they do, which is the opposite of anonymity. Their Web site has a lot of information about users who have done the most work in a given country or about teams who are competing against each other. It makes SETI@home users unhappy if their computer finishes a work unit but it does not show up in their data base [Anderson 2002].

Cost of Ownership. SETI@home has reduced the cost of ownership by a factor of 100. The commercial cost of the computing power of SETI@home would be about 200 million dollars for purchasing 30 TFlops of computing power and not accounting for administration and management. So far, the project has spent less than a million dollars [Anderson 2002]. The savings are achieved by exploiting a cost of ownership that already has been paid: buying a computer, setting it up, administering it, buying a network connection, etc.

Self-organization is not a factor in SETI@home because the computation has a very simple structure [Anderson 2002].

Ad-hoc connectivity is also not a big problem for SETI@home. The data/compute ratio is low: users only need to be connected a couple of minutes a day to transfer work units. Connectivity would be more of a problem for computations that require more frequent communication. The problem of computers becoming available and unavailable is taken care of through redundant computation. The main issue is computers that are disconnected for several days and run out of work, which is not addressed by the original design of SETI@home. This is improved by proxy servers that allow downloading/uploading of several work units at once.

Performance. SETI@home often had more computing power than it can use. It became more popular than they ever thought it would. Instead of 50,000 users they originally thought they would have, they had one million. Therefore they didn't have enough data to distribute and they would replicate computation eight or ten times instead of three times. Because of this, the project was revised to have more computation per unit data. This benefits their research to have a more thorough search, and it restores the balance between data and computing power [Anderson 2002]. A tradeoff was made between performance and maintainability in not addressing a variety of different types of processors, even within the Intel architecture. Some processors have different instruction sets or special instructions, e.g., for signal processing. AMD also manufactures Intel-compatible chips that have their own, completely separate instructions. A lot of users were eager to have a version of SETI@home that used different architectural extensions to process data faster. SETI@home didn't want to support different versions of software that would have assembly language for specific processors, making it difficult to maintain or change. It was more important for SETI@home to have a common source code, even if that would mean that they would not go as fast as they could.

Security. There are two aspects of security in SETI@home: protecting the system from users and protecting users from the system. SETI@home users have reverse engineered programs and beaten all of the security mechanisms in SETI@home [Anderson 2002]. In any system that gives users (virtual) payment or credit that shows up on the Web site, there is a problem of verifying that the user actually did the work that is recorded. The only remedy SETI@home developers have come up with is the brute force solution of performing redundant computations and comparing three results. In the other direction, one of the big fears when SETI@home started was that it could be used to distribute a virus. This has not happened. The SETI@home client program does not listen for a connection, unlike a file serving program, which

needs to accept the incoming connection. The SETI@home client only serves outgoing connections. The SETI@home communication protocol cannot be used to distribute executable code and that eliminates a big class of security concerns. The remaining security concerns had to do with the security of the servers, so that nobody can hack into the servers from which people download the client program. A lot of effort was invested in securing servers, such as writing scripts that constantly check file checksums.

Transparency and usability. One of the original interests of designers of SETI@home was in making an OS layer that can be used by other applications besides SETI@home. This has not been achieved in initial implementations; SETI@home is a monolithic program that includes the application and infrastructure. A new architecture is being developed that separates the two. This will enable different projects to co-exist, using the same software and sharing the same pool of participating users. For example, other projects might involve genetics or physical simulations. Instead of users choosing from one or the other project, they will be able to support both and divide resources based on their preference.

Fault resilience. The failure of computers is addressed by redundant computing. The SETI@home server sends the work units until it gets three results. If some computer becomes unavailable or takes a long time, the computation gets sent to another user. Because one unit may require as much as days to complete, it is necessary to ensure seamless recovery when the user becomes active (which stops the SETI@home client), as well as when the machine is shut down. The client resilience relies on a *checkpointing* mechanism, where the resumable dataset is saved on the hard disk every ten minutes.

Interoperability has not been addressed until recently. The new design that will enable multiple applications to run using the same infrastructure.

Privacy is less of a factor for SETI@home than it is for other kinds of computation, such as biotechnology, where data might be proprietary, e.g., genome sequencing. For existing commercial hardware, there does not seem to be a way to guarantee that data will not exist unencrypted. While data can be encrypted for storage application, it cannot be encrypted during computation. In SETI@home, there was a concern that over-zealous users would use the radio-telescope data for their own analysis, and announce a false discovery. Therefore, data is transmitted and stored in a weakly encrypted form. This would be easy to break, but it hasn't happened so far [Anderson 2002].

Lessons learned. The major value of the SETI@home project – apart from the scientific results – was to prove that the technology can be applied to real situations. We can envision how it could be used to better utilize the processing power of unused computers and to identify the peak periods of used computers. SETI@home developers were hoping for approximately 100,000 participants, but they surpassed this number within a week. So far, there have been more than 3,000,000 contributors. Security was not introduced until after it was noticed that some users wanted to do inappropriate things. Most of the problems were not caused by malicious users, but as a consequence of competitiveness. For example, people raised the frequency of their CPUs, making them more exposed to failures [Wortheimer 2002].

Business model. SETI@home did not need to have a business model because it was a university project. However, they still needed to get funding because SETI is not supported by the government. In 1992, the government decided not to fund it. This was an economical problem for the SETI@home project, because it was required to raise money from users and companies. This is one of the reasons SETI@home is attempting to build a new infrastructure that will be of interest to the government. It is also being re-developed as an open source program.

6.3 Groove

Groove is a collaborative P2P system, however it can also be considered a platform. Groove is mainly targeted to Internet and intranet users, although it can also be used on mobile devices, such as PDAs, mobile phones, and tablets. It is intended to enable communication, content sharing, and tools for joint activities [Leigh and Benyola 2001, Groove Networks 2000]. For example, all Groove applications automatically inherit security without the application developer needing to invoke security-related calls, thus making the development of secure applications trivial [Tuvell 2002].

History. Groove was founded in 1997 by Ray Ozzie, the developer of Lotus Notes. The first version of the product was released in the beginning of 2001.

Goals. Groove's main goal is to allow users to communicate directly with other users without relying on a server [Suthar and Ozzie 2000]. Other important goals include *security, asychronicity* (intermittent connectivity), *admin-free*, and *flexibility*. Groove users are authenticated, data is secured both on disk and on the wire, and data confidentiality and integrity are enabled by secure key technology. User data residing on the server is opaque and private data is not shared with third parties. Groove supports the Internet, intranets, and private net-

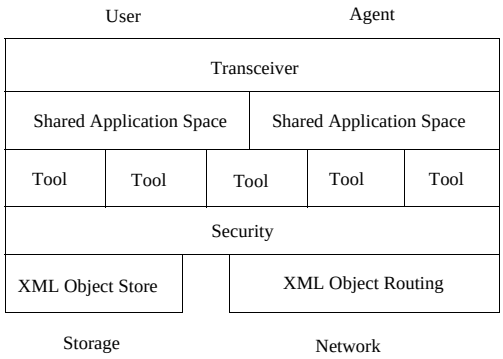

Figure 18: Groove Layers. *Adapted from Groove Product Backgrounder White Paper, http://www.groove.net/pdf/backgrounder-product.pdf.*

works, and allows users to transparently handle disconnection. Groove supports a number of reusable components and a flexible platform for building collaborative applications.

Applications. Groove is intended for communication, collaboration, and content sharing.

- *communication*: voice over Internet, instant messaging, chat, threaded discussions
- *content sharing*: shared files, images, and contacts
- *collaboration*: group calendaring, group drawing and editing, and co-browsing

Groove presents itself to users or to agents representing users in the form of shared spaces (see Figure 18). Shared spaces offer users benefits, such as *spontaneity* – users act without an administrator, *security* – shared spaces act as virtual private networks, *context* – spaces provide and maintain the persistent context for users, *synchronization* – all spaces of the same user are synchronized across all users devices, and *granularity* – modified (delta) parts of documents are exchanged rather than whole documents. Only the deltas of the changed documents are being sent because of bandwidth issues. The definition of a delta is application-specific, and not always obvious. In the initial implementation of chat, a delta per keystroke was sent, and in later implementations a delta per chat-message is used. In the initial implementation of MS Word integration (co-editing), the whole document was used for the delta but just "binary diffs" in later implementations [Tuvell 2002]. Delta/synchronization is used only within shared spaces and not in non-shared-space modes of Groove communication, such as instant messaging and voice chat.

Design. Groove is P2P by design. The Groove synchronization layer is inserted between the application logic and command execution layers (see Figure 19) [Groove Networks 2001a]. This enables the command requests or

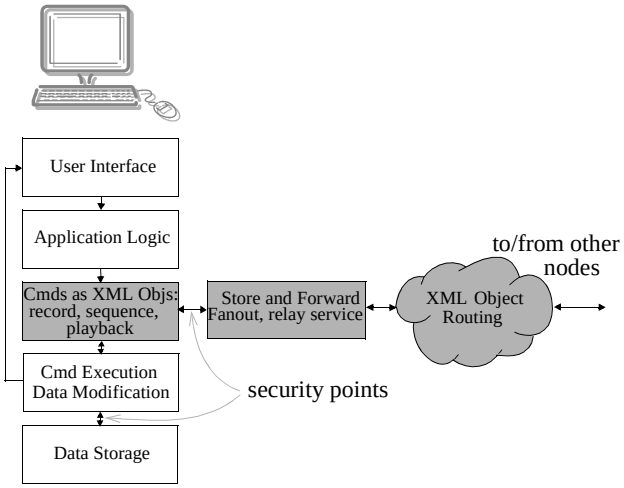

Figure 19: Groove Application Structure. *Adapted from www.groove.net/developers/presentations/architecture.exe.*

data to be stored locally or forwarded to the network, as appropriate. Commands are transformed into XML objects, recorded, and sequenced, which enables them to be played back and relayed in case of disconnection. Security layers are executed before storing objects to disk or sending them out to the network, preventing unauthorized access. Groove supports peer-based authentication and end-to-end encryption. Groove supports system and centralized services [Groove Networks 2001]. *System services* include:

- *security*, such as automatic handling of public/private key management, encryption, and authentication
- *storage* based on the XML object store, enabling disconnection from the network
- *synchronization* of the multiple local and remote replicas of shared spaces to which users belong
- *peer connection* to support transparent administration, such as IP addresses, bandwidth selection, and firewall translators

Within a managed administrative domain, *centralized services*, perform management of resources:

- *licence and policy management* which enables viewing all the devices using certain software as well as managing the use of this software
- *component management,* to enable managing Groove and the third party software/components on the fly without requiring user interaction for upgrades
- *relays and transparent cross-firewall interaction* to optimize the communication bandwidth and offline work, while still enabling direct P2P communication

- *usage reporting* to track how shared spaces are used, enabling administrators to optimize resource distribution while respecting user privacy
- *enterprise management server (EMS)* supports user management by importing an LDAP database and binding Groove identities via certificate authorities.

Security. Groove fully supports end-to-end crypto, reducing the security problem to authentication. In early versions, it only supported peer-to-peer authentication, using out-of-band fingerprint-checking. In later versions, PKI is introduced as an optional authentication mechanism. The Groove PKI solution is implemented as a CA co-resident on the Groove enterprise management server. Interoperability between Groove-PKI and third-party PKIs (such as Entrust, etc.) is being developed. Groove has a native support for access control, via an authorization framework. This framework is integrated into standard tools shipped with Groove, and it is exposed to application developers.

Anonymity. Groove was not designed to protect anonymity. On contrary, Groove designers believe that meaningful communication/collaboration cannot happen unless users know who they are communicating with. Nevertheless, Groove communicators do not have to authenticate one another. An invitation to a Groove-space can be "open", i.e., anybody can accept the invitation without knowing who the inviter is, and the inviter can confirm the acceptance without knowing who the acceptor is). However, Groove does not obscure the underlying network communications, so an attacker with a network sniffer could track down the endpoints. They have also explored the idea of an "anonymous identity" per shared-space (implemented by a public private/public key pair per shared-space), but so far it has not found practical use [Tuvell 2002].

Ad Hoc Connection. Because all peers are not online at any instant, Groove supports a Relay Server as a service at groove.net as well as a separate product, which the transceivers automatically use. The relay's main purpose is to enable intermittent connection to the network. The groove.net hosts a directory-like service for users who opt to register their contact info with Groove, but it is also possible for two users to invite/accept/join a Groove-space using email. Thus, any users who have email-connectivity (i.e., everybody, presuming they know one another's email address) automatically has Groove-connectivity [Tuvell 2002].

Performance. Groove computes the speed of a user's connection to the Internet, and if it is slow a single bulky delta/message is sent to the Relay, where it will get fanned-out, instead of sending multiple copies of the data directly from the source peer to the various targets. In addition, sending deltas and binary diffs of the shared information significantly improves performance of collaborative applications.

Fault Resilience. Groove supports fault resilience through interaction of transceivers as a part of their synchronization protocol. Transceivers retain copies of all deltas until they receive acknowledgement from all targets. Thus targets can realize when a delta has been lost, and recover the necessary data.

Lessons learned. As Groove was maturing, the best features of centralization and client/server were incorporated as necessary to meet the customer needs. Starting from a P2P architecture enabled that, because if they had started from a centralized or client/server base it would be much harder to incorporate P2P features [Tuvell 2002].

Business model. Groove Networks licenses its infrastructure platform to corporations and third-party integrators [Rice and Mahon 2001]. They are also investigating and outsourced services approach, by hosting a relay server farm and charging by quotas. Their competitors are primarily other collaborative P2P systems, such as Magi. Some of the enablers for adoption of Groove include elimination of the network administration costs, minimization of dependences on server infrastructure, and availability.

6.4 Magi

Magi is a P2P infrastructure platform for building secure, cross-platform, collaborative applications. It employes Web-based standards such as HTTP, WebDAV, and XML to enable communication among applications within an enterprise network or over the Internet. Magi Enterprise, their end product, builds over the infrastructure to link office and project teams so they can share files, do instant messaging, and chat [Endeavors Technologies 2001].

History. Magi evolved from a research project headed by Greg Bolcer at the University of California, Irvine. His team was funded by grants from DARPA, and at the time, it was believed to be the largest, non-Sun Java project in the country. Endeavors Technology was founded in 1998, and was later acquired by Tadpole Technology. The first version of Magi, which is their P2P infrastructure, was released in late 2000. This was followed by their enterprise edition in 2001. In the beginning Endeavors focused on mobile devices in consumer space, but lately it has focused on enterprises.

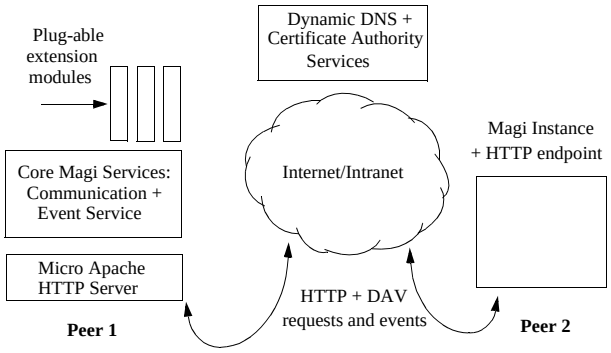

Figure 20: Magi Architecture.

Goals. The goal of Magi is to enable information sharing and messaging on any device using popular Web-based standards. The architectural framework runs on various platforms including PDAs and Pocket PCs. It aims to make development of XML and Java-based distributed applications easier by providing event notification to any device running an instance of Magi.

Design. The key components of the Magi framework include a micro-Apache HTTP server which provides a link to every instance of Magi, a set of core services, and a generic extensible interface. The modular architecture of an Apache server allows easy reconfiguration and the addition of interfaces only if they are required. The core services include:

- *Communication Service:* messaging interface between Magi services and the HTTP server
- *Event Service:* local modules register with the event service to receive specific event notifications; also enables communication among local modules
- *Buddy Manager:* maintains a list of active buddies and their most recent communication endpoints
- *Access Controller:* allows restricted access to resources on a Magi peer

Pluggable service modules such as MagiChat, MagiDAV, and instant messaging, could be built over the core infrastructure as shown in Figure 20. Each instance of Magi acts as both a client as well as a server and hence can participate in P2P communication with other Magi servers. Magi supports the following Web protocols: HTTP1.1; WebDAV, for remote authoring; and SWAP/Wf-XML, for remote process monitoring and control. The modules supporting these protocols can be loaded when required to achieve interoperability with applications compliant with these protocols. These features reinforce Magi's primary design goal to achieve interoperability with existing Web access standards.

Decentralization. Magi attempts to achieve a balance between centralized management and decentralized content in enterprise environments. State and functionality is centralized for ease of management, while there is a lot of data that is decentralized by the nature of computer deployment. In the enterprise world the IT is overburdened, but it still needs to be in control. Magi gives IT ability to revoke certificates and solve the permissions, but it transfers the responsibility for the management of buddies, group, and passwords to the end user. Magi supports different trust, relationships, IT policies, QoS, social, and organization policies that surround intellectual properties and the content. A lot of cross-organization data sharing, collaboration and communication, distributed computing does not work unless all sites can agree on the same peer or architectural platform. This is often not possible unless participants are interoperable at all levels. Magi provides a universal security and access to decentralized nodes, but the peers are meant to be unequal.

Scalability. Magi's scalability is somewhere between centralized Web serving and email [Bolcer 2002]. If an email is used for collaboration, it will require changes by a number of people, resulting in more copies of the email message. Magi bandwidth usage is better than email protocols and a little worse than Web protocols, because it requires the round trip, but the scaling part is the same. There are various Magi components, such as MDNS (registration server that keeps track of dynamic IPs), search server, connection broker (helps different people behind two firewalls to connect), and a certificate server (PKI server). Some of these components can be hosted on one computer, however, to scale it up to an enterprise, components need to be split on multiple servers. It is well known how many users each of the components can sustain. For example, one connection server is required per approximately every 5,000 users. In an enterprise with 50,000 people, there is approximately 40% (20,000) cross-firewall traffic, requiring approximately 4 connection servers. With the mid-range PC, with filters turned off, Magi can support 400 connections per second and several thousand users from that machine. Scalability is impacted by doing search filtering so either commercial search engines are plugged in or Magi light filters are used which can filter up to 200-300 file sites.

Anonymity and Privacy. Magi does not deal with anonymous users at all. All users are strongly authenticated. Not even anonymous browser requests are permitted on Magi. The audit trail is performed, such as which documents have been passed and who they came from. In Magi enterprise there is neither anonymous reading nor writing. That requirement comes from the enterprise cus-

tomers. There are certificate pairs for every user and device. For example, a user's work machine and laptop have different certificate pairs, but they are both part of the same user name space. All authors are tied to their certificates. Users can not access network without a certificate. Unlocking certificates is achieved using passwords and using passport [Bolcer 2002]. Currently enterprises can only be searched on portals and application servers. Magi enables searching content of individual peers on desktops, laptops, and PDAs within intranet. If all the content is indexed on these machines, no private data or meta-data need to be available on the central search server. With the support of connection brokers Magi can index all devices even when they move inside or outside firewalls and behind two or more firewalls. Using buddies and group permissions, Magi filters the search results and indexes all the private and semi-private information that is on individual machines. However, it will not return information to users who do not have permission. For people on the buddy list search will return results on the shared folder of buddies, for people in the same group, the search will return results from the group folder.

Self-Organization. Every member of an organization can have buddies and send invitations for them and accept. This way, the formal organization supported by IT staff is extended to an informal one, governed by individuals. Buddies and group members automatically get permissions to shared file areas for direct access, as well as for searches, which generate results for those file that individual has permission to and no results for those files that she has no permission to.

Cost of Ownership. The primary cost of ownership benefits in Magi stem from the lower cost of management and deployment, which are similar to traditional Web services deployment but with less central management cost and less deployment cost. Users are comfortable deploying Magi manually and IT staff is comfortable to control certificates and not necessarily entire software.

Ad-hoc Connectivity. Magi addresses ad-hoc VPNs without having to set up firewall openings or name and password permissions on each edge of a site [Bolcer 2002]. Users can form an ad-hoc connection on each edge of the site, by forming an ad-hoc VPN using two different connections. Magi also manages presence, whether users are offline or online. Instant messaging supports store and forward of short messages or files in a P2P fashion. Group invitations are handled centrally, because it is possible that both the source and destination of the invitation are offline, slowing down distribution of group permissions. This is one instance of centralization, introduced by design. The centralized server can also be a shared buddy that is online all time. Another reason for doing it centrally is because the IT staff wants the ability to cancel group invitations when needed.

Performance. Magi addresses two performance issues: individual desktop performance and enterprise-wide performance. A minimum configuration desktop of 400MHz PC running an up-to-date Windows OS, runs without performance constraints when 30-50 documents are edited by a remote group at the same time. However, Magi is optimized for less than that, because there are typically 5 people working on a document. Because it is based on Apache, it is possible to tune for specific requirements. Magi distinguishes the bandwidth between the cheap one (internal or intranet) and an expensive one (external or extranet), which has to go through gateways or proxy. The goal is to use cheap bandwidth as much as possible and expensive bandwidth as little as possible. Magi interposes network applications and does network routing, e.g. for AOL instant messenger or for Outlook, there is a local proxy sitting on the desktop. Instead of having an AIM message going to the AOL AIM server through a firewall, Magi can interpose this traffic and send it directly. The same applies to mail messages.

Security. Magi supports everything that secure Web server does. Magi supports simple PKI implementation, as well as integration with certificates from VeriSign and RSA. Typically, a key pair exists for every user/device instance. All connections and all traffic go through SSL (encrypted); all authentication takes place in a P2P fashion. The second security level is at the buddy and group level. The third is access level: if user has permissions to perform the HTTP or WebDAV method on the individual resource. There is a set of methods on individual peers which are end-user configurable. Magi does not support shared keys for group definitions. It crosses firewalls by exporting network installations. IT has the ability to revoke keys both at the user and firewall level.

Transparency and Usability. In the resources folder (shared folder), Magi provides location transparent information about resources. However, users can also specify particular machine to store information on. Magi uses the URL naming mechanism for resources, which proved useful when machines change IP address or locations. URLs used for indexing search results do not break when users move around. It is almost like a URN except that it is not a naming token, but a fully qualified URL.

Fault Resilience. Magi has a hybrid model with traditional techniques, such as failover, round-robin and proxies. If some central components go down there is a graceful degradation for users. For example, there is a failover for certificate authority (CA). Even if all CAs go

away, a user can authenticate other users, but it is not possible to create new users. If MDNS goes down and user with a dynamic IP, she can use a person with a static IP to find a person with dynamic IP. As long as user knows someone on static IP he can exchange credentials. The only problem is if two peers use dynamic IPs, and MDNS is down, they will not be able to find each other. If connection machines go down, there is no visible effect except if too many go down in which case there is some performance degradation. Magi also queues messages that cannot be delivered to buddies who are currently offline and enables delivery when the destination buddy eventually comes online. Buddy crashes are not detected, as there is no mechanism to check the liveliness of each buddy. However, the core Magi interface can be extended as desired to support any kind of fault resilience mechanism among the buddies.

Interoperability. Magi emphasizes the use of existing standards. This makes Magi a highly interoperable platform, and its Web-based design makes its deployment on a range of devices easy. Magi supports WebDAV and any WebDAV enabled desktop tool can read/write content across the firewall. In addition a lot of tools are made WebDAV compliant by giving an illusion to write to a local file which is really a proxy for the real file across the firewall and network. This is protocol-based interoperability.

Lessons learned. The most important lesson learned is that enterprises are typically willing to let go of centralized control. Magi supports the hybrid model with the right amount of central control and end user empowerment. This enables the same amount of IT control, but it reduces the mundane, tedious things for end-users. Search and network publishing are two biggest application areas for Magi.

Implementation. The entire infrastructure is in Java and the Web interface is through servlets. This makes Magi as fast as the underlying virtual machine. Because each Magi instance is able to both receive and send messages over HTTP, a minimal Web server must be running. This may not be the best solution for a Pocket PC or a handheld with limited memory resources. Magi uses Tomcat, which supports modular loading of essential components, coupled with independent service-oriented design of the infrastructure to target constrained and embedded devices. Magi's services are accessible through a Web-based interface and is the access mechanism in their implementations for the Compaq iPaq and the HP Jornada pocket PCs.

Applications. Magi was designed primarily with the paper-based workflow scenario in mind and is targeted at any type of collaborative environment. It supports file sharing and collaborative tools such as chat and messaging. As a platform, it can be used to embed collaborative processes into enterprise-wide applications and B2B products. An SDK has also been released to ease the integration of Magi into applications.

6.5 FreeNet

Freenet is a P2P file-sharing system based on an initial design by Ian Clarke [Clark 1999, Clark et al. 2001, FreeNet 2001]. The primary mission of Freenet is to make use of the system anonymous. That is, upon entering the system, a user should be able to make requests for files without anyone being able to determine who is making these requests. Likewise, if a user stores a file in the system, it should be impossible to determine who placed the file into the system. Finally, the operator of a FreeNet node should have no knowledge of what data is stored on the local disk. These forms of anonymity make it possible to provide storage and use the system with less concern of being tracked down or potentially held liable.

History. The Freenet system was conceptualized by Ian Clarke in 1999 while at the University of Edinburgh. In his introduction to Freenet, he cites the following quotation by Mike Godwin: “I worry about my child and the Internet all the time even though she's too young to have logged on yet. Here's what I worry about. I worry that 10 or 15 years from now, she will come to me and say “Daddy, where were you when they took freedom of the press away from the Internet”. Public development of the Open Source Freenet reference implementation began in early 2000.

Goals. The principle goal of Freenet is to provide an anonymous method for storing and retrieving information. Freenet permits no compromise on these goals. However, within these bounds, Freenet also strives to be as reliable, easy to use, and responsive as possible.

Design. One of the key design points of the Freenet system is to remain completely decentralized. Therefore, Freenet represents the purest form of P2P system. Freenet's basic unit of storage is a file. Every node in the Freenet network maintains a set of files locally up to the maximum disk space allocated by the node operator. When all disk space is consumed, files are replaced in accordance with a least recently used (LRU) replacement strategy.

Each file in the Freenet system is identified by a key. These are typically generated using the SHA-1 [1997] hash function. A variety of mechanisms are used to generate the desired hashes, but typically a user starts by pro-

viding a short text description of the file. This description is then hashed to generate a key pair. The public key becomes the file identifier. The private key is used to sign the file to provide some form of file integrity check. Other schemes for generating keys can be used as well permitting users to create hierarchical file structures or to generate disjoint name spaces. The file's key is then made available to users of the system by out-of-band mechanisms such as a Web site. Because the key can be computed from the description of the file, it is common to publish only the description and not the actual key.

The only operations in the Freenet system are inserting and searching for files. In either case, it is essential to find the proper location for the file. In general, Freenet nodes form a network in which they pass and forward messages to find the location of an existing file or the proper location to store a new file. The keys are used to assist in the routing of these messages. Freenet attempts to cluster files with similar keys on a single node. By clustering, Freenet is able to optimize searches by creating routing tables. When a file is successfully located by a search, the file's key is inserted into a local routing table. When another search message is received, it is first forwarded to the peer node with the most similar key in the routing table. When a search is received by a node that contains the desired file, it returns the entire file as a successful result. This is done recursively until the file is returned to the initial requester. As a side effect, the file becomes replicated at each node in the search path. In this way, popular files become highly replicated.

Inserts operate much the same as searches. First, a search operation is performed on the file's key. If a file with that key is found, it is returned to the inserter as a form of key collision indication. When no existing file is found, the file is propagated forward along the search path. This accomplishes both the replication of the file that occurs during a search as well as preserving the integrity of the routing tables by placing the new file in a cluster of files with similar keys.

New nodes announce their presence in the network by performing a search operation. This search basically accomplishes the function of announcing the new node to other existing nodes. Prior to sending the message, the node must first discover at least one other node in the network to which it can connect. Freenet explicitly does not help in this problem because doing so typically leads to centralized solutions. User's are required to bootstrap themselves into the network using out-of-band means for finding at least one peer.

Scalability. The scalability of Freenet has been studied by its authors using extensive simulation studies [Clarke et. al. 2001]. Their studies support the hypothetical notion that route lengths grow logarithmically with the number of users. In their experiments, they start with a network of 1000 nodes in a ring topology. As the experiment runs, random keys are inserted and removed from the network, and the path length to find a file reduces from approximately 500 to less than ten. This is consistent with the logarithmic curve expected.

Implementation and performance. Freenet is available in an Open Source reference implementation. The protocol is also well defined allowing others to create their own implementations. One side-effect of the focus on anonymity in Freenet is the difficulty in observing its behavior. Obscured identities and probabilistic choices make measurements difficult. For these reasons, no real world performance studies seem to be available. Only the simulation results described above can be used to evaluate the system.

Lessons learned. The key lesson of Freenet is both the importance and difficulty of maintaining anonymity. Anonymity opens the door to freer discourse on the network because users need not be concerned with the ramifications of their actions. Anonymity also runs contrary to many intellectual property notions such as copyright. The Freenet team argues that preserving freedom and eliminating censorship are more important than intellectual property concerns.

Business model. Freenet operates as a non-profit Open Source project. There are as yet no commercial ventures building on top of Freenet. The Freenet project does solicit donations to help fund continued development, but does not appear to be dependent on these donations.

Applications. Freenet's only real application is as an information storage and retrieval system. The heavy use of cryptography as well as anonymization of requests may lead to other related uses. For example, it may be possible to use Freenet as a form of distributed backup system with some guarantees that data can only be retrieved by the owner of the data. However, Freenet only provides probabilistic guarantees about the persistence of data, so this likely would not make for a high-confidence back-up solution.

6.6 Gnutella

Gnutella is a file sharing protocol. Applications that implement the Gnutella protocol allow users to search for and download files from other users connected to the Internet.

History. Gnutella file sharing technology [Gnutella 2001] was introduced in March of 2000 by two employ-

ees of AOL's Nullsoft division. Touted as an open-source program with functionality similar to that of Napster [2001], the Gnutella servant program was taken offline the following day because of a possible threat to Warner Music and EMI. AOL was rumored to be in the midst of merger talks with the record companies at that time. However, the open-source program remained online long enough for eager hackers to discover the Gnutella protocol and produce a series of clones to communicate using the Gnutella communication protocol. Soon after, versions of the original Gnutella servant were communicating with Gnutella clones to search and trade files over the Gnutella Network. FastTrack [2001] is probably the most popular early Gnutella-based technology. Many popular peer-to-peer clients today, such as Kazaa, Grokster [2001] and the older version of Morpheus operate using the FastTrack technology.

Goals. The goal of Gnutella is to provide a purely distributed file sharing solution. Users can run software that implements the Gnutella protocol to share files and search for new files. The decentralized nature of Gnutella provides a level of anonymity for users, but also introduces a degree of uncertainty.

Design. Gnutella is not a system or a piece of software. Gnutella is the communication protocol used to search for and share files among users. A user must first know the IP address of another Gnutella node in the network. This can be discovered by going to a well-known Web site where a number of Gnutella users are posted. When a user wishes to find a file, the user issues a query for the file to the Gnutella users about which it knows. Those users may or may not respond with results, and will forward the query request to any other Gnutella nodes they know about. A query contains a Time-To-Live (TTL) field and will be forwarded until the TTL has been reached.

Scalability. While the Gnutella model has managed to succeed thus far, in theory it does not scale well. The number of queries and the number of potential responses increases exponentially with each hop. For example, if each node is connected to only two others and the TTL of a query is 7 (the default for most Gnutella queries), the number of queries sent will be 128 and the number of responses may be substantially more depending on the popularity of the item.

To avoid broadcasting queries in the network, Limewire has introduced the use of super-peers called *ultrapeers.* In their scheme, hosts create query route tables by hashing file keywords and regularly exchanging them with their neighbors [Rohrs 2002]. In particular, leaf nodes send query route table update messages to ultrapeers by communicating a compressed bitvector of the hash values of every matching keyword for all files and metadata that they have. Ultrapeers never propagate these tables. In this scheme each ultrapeer can easily handle 100 leaf nodes and greatly reduce the bandwidth requirements. Currently they are extending it towards 500 and 1000 leaf nodes [Bildson 2002].

Fault resilience. The Gnutella protocol itself does not provide a fault tolerance mechanism. The hope is that enough nodes will be connected to the network at a given time such that a query will propagate far enough to find a result. However, the distributed nature of the protocol does not guarantee this behavior. In fact, studies have shown [Adar and Huberman 2000, Saroiu et al. 2002] that only a small fraction of Gnutella users actually remain online long enough to respond to queries from other users. More robust software has begun to alleviate this problem. Limewire, for example, has retry logic built-in for files [Bildson 2002]. However no guaranteed solution exists. The only real solution at this point is to rely on users to retry when their queries or downloads fail.

Implementation and performance. Since the first Gnutella servant was posted by Nullsoft, a number of companies have implemented clone software and made efforts to overcome many of the limitations not addressed by the protocol. Among the most popular are Limewire [2001], BearShare [2001], Morpheus [2001] and ToadNode [2001]. Kotzen claims that while in late 2000 only 10% of download attempts were successful, by March of 2001 the number grew to over 25% [Kotzen, 2001]. This was mainly attributable to browser blocking. Web sites that crawled the Gnutella network used to allow direct access to Gnutella uploaders since uploads are purely HTTP requests. These web users overloaded the Gnutella network and when they were blocked (by detecting their User-Agent), the network improved. Furthermore, the number of Gnutella users has increased. According to Redshift Research [2001], over 300,000 users were logged on to the Gnutella network over a typical weekend in March of 2002, a substantial increase from an average of only 19,000 simultaneous users in December 2001. While this represents an increase over previous findings [Clip 2 Marketing 2000], it does not provide any proof of the scalability or performance of the Gnutella network for the targeted thousands or millions of nodes.

Business model. Gnutella is not a company or a piece of software, but rather an open protocol. As such, there is no Gnutella business model. Many of the companies developing Gnutella-compatible software are open-source or offer free download. Additionally, some companies are building software for the enterprise that offer content-specific content sharing. Ultimately this will rise busi-

ness models similar to television with built-in advertising support, cable with subscription models and pay-per-view models. Gnutella however is pioneering many decentralized techniques in peer-to-peer searching and storage. This raises questions of how much control can be exerted in a fully decentralized system.

Lessons learned. Perhaps the most important lesson is that the true measure of success for a system is user acceptance. Gnutella is widely adopted and together with SETI@Home, they are the most deployed peer-to-peer systems. As a result, the Gnutella model has raised a number of questions in the research and development community. Well-known solutions to traditional computer systems problems cannot be applied in a straightforward manner to Gnutella's decentralized network. Researchers are busily trying to apply distributed computing and database problems to ad-hoc, unorganized, decentralized networks of nodes. The benefit of a Gnutella-style network is the use of an unbounded set of diverse resources. However, solutions often trade reliability for the potential of discovering an otherwise unavailable resource. Furthermore, having a pre-established design is good in that there is interoperability but the downside is that there is less upgrade flexibility. This is particularly true in a multi-vendor environment.

Applications. The primary application for this technology has been the sharing of music files. A second potential application is chat [Kan 2002]. While this continues to be the main motivating application for companies developing Gnutella-compatible software, Gnutella is more a paradigm than a given application. A number of companies are developing visions of using the Gnutella protocol for enterprise software including project-management-style applications as well as academic software for classroom-based communication.

6.7 JXTA

The vision of the JXTA project is to provide an open, innovative collaboration platform that supports a wide range of distributed computing applications and enables them to run on any device connected on a network [Gong 2001]. JXTA provides core functionality in multiple layers, including basic mechanisms and concepts, higher level services that expand the capabilities of the core, and a wide range of applications that demonstrate the broad applicability of the platform.

History. The JXTA project was unveiled by Sun on April 25, 2001 [JXTA 2001]. Despite its recent introduction, JXTA has been quite popular. Statistics show that on the week of 5/4/02, JXTA had over 400,000 downloads, close to 10,000 members, and 71 projects.

Goals. The goal of JXTA is to provide a "general-purpose" network programming and computing infrastructure. Its goals are:

- *Interoperability*: enable inter-connected peers to easily identify each other, participate in community-based activities, and offer services to each other seamlessly across different P2P systems and communities
- *Platform independence*: from programming languages (such as C or Java), system platforms (such as Microsoft Windows and UNIX operating systems), and networking platforms (such as TCP/IP or Bluetooth)
- *Ubiquity:* implemented on every device with a digital heartbeat, including appliances, desktop computers, and storage systems

Design. It is important to note that the JXTA project is approaching the P2P space from the lower level as they are proposing an entirely new infrastructure with no direct relation to other, existing P2P systems (e.g., Gnutella and Napster). For example, they have built their own distributed search service, called JXTA search. There are three core abstractions in JXTA: peers, groups, and pipes. Peers are the first level objects in JXTA and they can be organized in groups. Peer groups are the core of JXTA's infrastructure. A peer group is essentially a partitioning of the world of peers for communication, security, performance, "logical locality" and other reasons. A single participant can be in multiple groups at one time. JXTA provides core protocols for peer discovery, peer group memberships, and peer monitoring. JXTA uses asynchronous uni-directional communication channels, called *pipes*, for sending and receiving messages. Pipes represent a virtual port of the descriptor used for communication [Yeager 2002]. The description of peers, pipes, peer groups, and rendezvous are published in XML formatted documents. For performance reasons the exchange of the XML documents is based on JXTA binary messages which are payload-neutral.

Services. The JXTA project has defined core and optional services that run on the JXTA platform. Examples of core services include authentication, discovery, and management. To address the need for security, they have implemented a cryptography toolkit that enables message privacy, ensures authentication and integrity, and permits transactions between peers that cannot be repudiated (see paragraph on Security below). Another core service is the Content Manager Service (CMS) that allows JXTA applications to share, retrieve, and transfer content within a peer group. However, in the current version of the CMS, there is no support for automatic propagation of content requests; each peer sends content requests directly to the other peers. Examples of optional

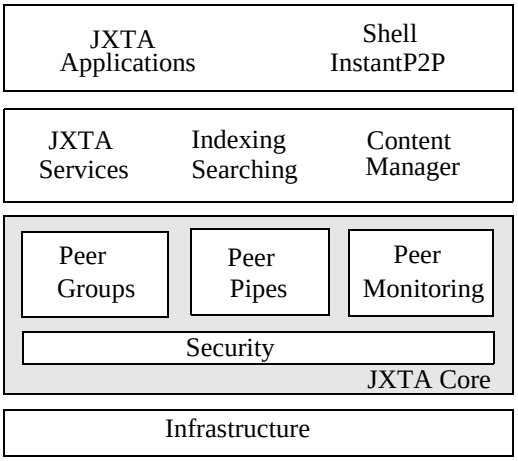

Figure 21: JXTA Architecture.

services are naming, routing, indexing, and searching. For example, the JXTA distributed indexing service implements a general-purpose, fully distributed index service by using a distributed inverted index to map keywords to a list of postings.

Applications. The JXTA Shell was the first application developed that gives a command-line interface to the core JXTA services, such as peer and group discovery and messaging. Since then, the JXTA project has developed more applications that allow interactive access to the JXTA platform. For example, the InstantP2P application enhances the JXTA project with chat capabilities. It includes functionality that enables users to chat one-on-one, chat with a group, and share files. Another application is JuxtaProse, which is a discussion application that allows creating, viewing, and replying to HTML documents over JXTA's content management system. Other applications that the JXTA community is currently designing include event notification and P2P email. Some other applications developed by third parties include real-time code editing and collaborative graphics design.

Decentralization. The original JXTA core was entirely decentralized. There was no intention to use DNS, LDAP, or any of other centralized name server, although JXTA applications can use it. However, in order to do long-term discovery there was a need for rendezvous bulletin boards for publish/subscribe advertisements of peers. These rendezvous points are the only centralized aspect of JXTA. Rendezvous are much different than DNS because registration is ad-hoc. A peer can arrive and register. If it goes away, it disappears, there is no dependency on other peers. Rendezvous is also used for firewall/NAT traversal. Peers both inside and outside of firewall-ed peer communities can discover one another using the rendezvous mechanism. A rendezvous is currently a combination of a bulletin board, publish-subscribe service, a router, a relay to do store and forward message exchange, and a message propagation service for group chat on multicast pipes. Any peer can be configured as a rendezvous [Yeager 2002].

Scalability. The JXTA project has a layered architecture, which consists of three building blocks: JXTA core (for peer group management), JXTA services (such as searching and indexing), and JXTA applications (e.g., JXTA Shell). This layered architecture enables JXTA to easily incorporate new protocols and services to support a growing number of users. JXTA attaches a unique ID (generated by the users) with each peer in the group but does not solve the global naming problem because it does not guarantee uniqueness across the entire community of millions of peers. Even if a peer has a unique ID within its group there is no guarantee that its name is unique across multiple groups. However, the probability of collision is very small. The Universal Unique Identifier is 64 bytes of which 32 are generated by a SHA-1 based pseudo random number generator. In addition, JXTA does not prohibit the use of a name registry. Another potential scalability problem is of an implementation character. The Java threading model is not as versatile as the Solaris POSIX threading model. This will be addressed with a C implementation of JXTA. There are no scalability limitation with respect to local clients. For example, average TLS can transfer 1.5-2Mb/s in Java, with the handshake latency of 1-2 seconds. It is believed that there is no current bottleneck even if there are no current targets [Yeager 2002]. There is an in-house expertise with supporting up to 100,000 transactions every couple of seconds for email support. Advertisements in JXTA are similar to small emails; scalability is limited by the number of threads that can be run on Solaris.

Anonymity. JXTA decouples semantics of naming from mechanism. Anonymity can be built on top of JXTA as a service and JXTA protocols do not prevent users from doing so [Gong 2002]. This is a general JXTA design principal: not to prevent users from providing anything on top of JXTA. Every peer has a 64 bit identifier and a name, but the name need not have anything to do with the user. It is something that is readable, but others still may not have any idea who the user is [Yeager 2002].

Self-Organization. JXTA is self-organized; JXTA users are volunteering to deploy relays and rendezvous. The cost of ownership depends on who is buying these machines. Rendezvous systems can come and go, but the whole JXTA system is self-adaptive to these changes. Upon bootstrap, the configuration specifies if the system is a rendezvous or not, the peer name, the list of other rendezvous systems, security, user name and password.

Fault resilience. In JXTA, fault resilience is built into the discovery mechanism. Instances of three JXTA core abstractions are advertised at rendezvous points: peers, groups, and pipes. If they go away because of disconnection, reboot, or failures, the priorities deal with it. This enable much simpler assumptions; it is up to application developers to assert if they need more. However, for presence detection they can use lower level protocols to discover whether pipes are active. Traditional fault tolerance mechanisms and approaches (for example virtual synchrony) would not be applicable in such a peer-to-peer environment [Yeager 2002].

Security. JXTA supports distributed PKI and an implementation of transport layer security (TLS) with both client and server authentication [Sun Microsystems 2002]. Most existing TLS are based on sockets. Because in JXTA there are no sockets, but rather a virtual network, the end communication points are identified with 64-bit peer IDs. Virtual transport is a lower level communication mechanism for pipes (not UNIX pipes), which can be open as secure pipe, supporting RSA 1024, 3DES, SHA1 standards for encryption. The pipes are multiplexed through a same TLS communication, so there is amortization of handshakes [Yeager 2002].

Transparency: Application programmers need only know about pipes and obtain the list of pipes. Pipes are like phone numbers (users need not be aware whether it is wireless or physical, link) or are like virtual communication ports [Gong 2002].

Interoperability. Interoperability is one of the goals for the C port of JXTA.[1] A proof of protocol interoperability is one of the goals for the completed C port of JXTA. One can run applications written in any language with common underlying JXTA protocols. Siemens has ported JXTA to Java 1.1.8 for its personal Java PDA, and a JXTA J2ME/MIDP implementation is also finished. It has been demonstrated that all of these implementations are protocol interoperable, and all data maintains its integrity from edge device to edge device. This means that secure data that is encrypted at the source peer is decrypted at the ultimate destination peer.

Implementation and performance. The first version of the JXTA core has currently been released, running on four different platforms: Solaris, Linux, Microsoft Windows, and a Siemens PDA. A prototype of the JXTA Shell that gives a command-line interface to the core services is also available. In addition to that, there have been many independent efforts that build services and applications on top of JXTA.

1. Other goals include a) the proof of concept, b) some devices support only C, and c) to prove that JXTA is not Java or Jini [Yeager 2002].

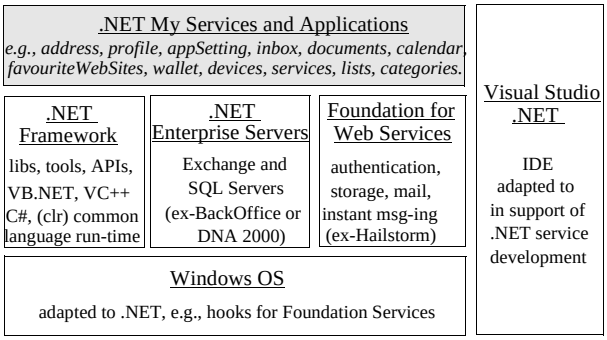

Figure 22: .NET Architecture.

Lessons learned. Users do not need all the centralized support in order to get the service. For example, email and instant messaging can be supported by a peer-to-peer infrastructure. Today there exist superhighways, but no small streets are there pipelining into highways. P2P is like small streets. Local content aggregation is motivating force, a lot can be done locally [Yeager 2002].

Business model. JXTA has been released as open-source code and has already attracted many developers to build applications (e.g., event notification, file sharing, P2P email) on top of it. Sun is also working with other P2P companies that are committed to work with the JXTA open-source model and contribute their P2P technology to it.

6.8 .NET My Services

.NET My Services and .NET in a more global form do not represent a P2P system in its traditional form, as do the other P2P systems described in this paper. However, .NET My Services encompass a lot of P2P architecture and as such they are presented here. .NET also introduces a new language called C#, development tools, a framework, a foundation for Web services, and the underlying operating systems. This paper addresses only the .NET My Services part of it (the shaded part of Figure 22).

History. Microsoft officially announced .NET for the first time in June 2000. Shortly thereafter, certain components were available, such as operating systems, parts of the .NET framework and enterprise servers, and passport. During 2001 and 2002, subsequent versions of operating systems, frameworks, and some key services have been introduced.

Goals. The goals of .NET My Services and .NET in the broader context are to enable users to access Web services on the Internet from available devices, including desk-

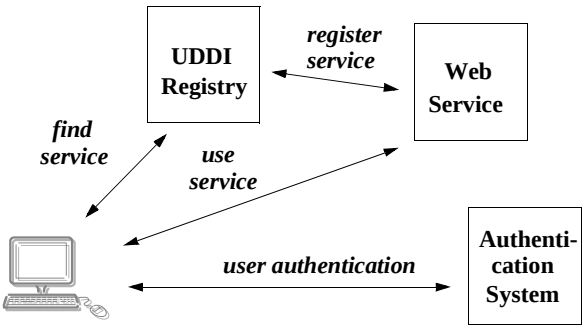

Figure 23: User Interaction with .NET Components.

tops, laptops, and handhelds, using existing standards, such as XML [Bray et al. 2000], UDDI [Ariba et al. 2000], SOAP [Box et al. 2000], and WSDL [Christensen et al. 2001]. The goal of .NET is to shift focus from applications, platforms and, devices to user-focused data. Therefore, security and privacy are one of the major goals of .NET.

Design. The design of .NET is focused around de-componentization and decentralization of distributed services. For example, a user will locate the desired service through the UDDI registry, then she will access service through the Web Services provider location, and she will be authenticated through a separate passport service (see Figure 23). The .NET Services programming model is built around Web standards and standard protocols, such as SOAP, UDDI, and WSDL. The .NET System programming model is built around a new common language run-time, which is intended to complement the Java run-time. Passport (see Security and Privacy below) is an authentication method for .NET Services. Other Web services can communicate with .NET My Services without disclosing a user's identity and personal information.

Decentralization in .NET My Services is not as much about logic, as it is about the data being managed [Lucovsky 2002]. For example in Calendar application, the calendar data for an individual is the point of centralization. Data is set up to be easily cacheable and it supports multi-master replication. There is a notion of definitive store, which can move from one back-end cluster to another or from one data center to another. It is also possible that the edge PC is the definitive store in case of the real distributed deployment. However, there is a concept of one reference copy of the data and that's where the Web services are going to be directed to. That is what is centralized about the .NET model. It is not different than UNIX home directory which could be on any file server node but it is not replicated on any computing device or disk that user currently uses.

Scalability is of ultimate importance for operating large Web services of the magnitude of *msn.com* or *ms.com*. Hotmail is one of the biggest services on the planet. The traditional conservative multi-tier design is applied where the network capacity is tuned by cluster, and the architecture is highly partitioned. Each cluster has a known bandwidth needs at both stages of the cluster, inbound at the front-end and between the front-end and the data-tier. Approximately fifty front ends exist per one back-end. There is steering into the protocol to directly find a front end cluster from the service request, and once there, to find the back end that is connected to it. It was a conscious design decision to make the partitioning of the system visible to the programmer. When user requests a URL to send request to, the client application does the directory lookup to find specific cluster including the front and back-end machines serving the particular user. This is hotspot on MS Hotmail, for example. Cluster information is stored in cookies so that it is not required to do the lookup again. By partitioning the data along the type boundaries (e.g. calendar data v. contacts in MS Outlook), there is no need to partition lookups in the database core or in the data access layers. Flat data architectures could also have been done, but that would limit scalability [Lucovsky 2002]. It is possible to run .NET My Services on a laptop or on a rack that has 50 front and two back end multi terabyte data base machines.

Cost of ownership. On the client side it is low and on the server it is a function of the complexity and size of the user base. If a service is run for 100 people then it is no different than a small server application. If a service is running for a hundred million people, then cost of ownership is similar to that of traditional enterprise services.

Anonymity. There are no anonymous messages coming into the system. Any message has to carry the authenticated id of the requester of the service. There could be an anonymous *id*, but the message has to carry an authenticated *id*. One can visit a Web site and turn off the cookies and not have anything identify you as coming from certain company. In .NET My Services this is not the case.

Security. The original security architecture was based on Passport, an online authentication service. Passport is used for accessing Web pages and services. Once authenticated, the user can roam across passport-participating Web sites. Web sites in turn are required to implement the single sign-on. Passport was released in July 1999 [Microsoft 2001] and it had over 160 million accounts by July 2001. Users of passport are required to create an account that contains: *passport unique identifier (puid), user profile* (email, phone, name, address), *credential* (password and a four-digit key), and *wallet*

(optional, credit card numbers). Of the passport information, only the puid is shared with participating Web sites; wallet is used for purchases only. Passport protects the privacy of users by playing the intermediary in user authentication. Passport is not an exclusive authentication service in .NET My Services. In case of federated distributed deployment of .NET My Services in an enterprise, the active directory or the authentication system in that enterprise becomes the authenticator for users in that domain. These services become peer to *passport* and traditional trust hierarchies are used. The deployment of .NET My Services authentication system is similar to a Key Distribution Center (KDC) of Kerberos [Kaufman et al 1995]. There is a KDC of a base of users that are authenticated and a set of services that are in the realm covered by that KDC. This is like an island of .NET My Services that can run in the *msn.com* domain and an equivalent island can run in corporate MS, corporate Boeing and corporate HP. These islands can trust one another trough traditional trust mechanisms and use Kerberos cross-realm service tickets [Lucovsky 2002].

Privacy. In .NET My Services, privacy is of high importance because of the potential access to personal data. No data is shared unless there is explicit permission. Encryption of request/response on the wire prevents from snooping and data is also encrypted in the data base in case someone steals the disk drives.

Ad-hoc connectivity is difficult in general in any Web service architecture. .NET My Services have been designed to be highly cacheable and easy to resync with definitive store once connectivity is restored. A user can take her address book offline simply by making an interaction with the system when connectivity exists and keeping the answer in an XML file. The API is modeled like an XML DOM (Document Object Model). It is a restricted DOM, with no navigation with X-path and with the XML's proper name space, it is highly tailored to the person who will take it offline. When user makes changes to it, it is pushed into the .NET My Services which take care of the conflict resolution.

Performance. .NET Services were designed to execute at very-high scale (millions of users). Therefore, some tradeoffs were made in performance in the multi-tier architecture. The performance issues are addressed, by adding capability on the data tier on the front end. Because Microsoft also owns application side and service, it is possible to better understand the performance requirements of the address book, hotmail, all messenger, all calendar, all *msn* to serve the aggregate needs of 150 million users. On a lower scale, it is possible to estimate the resources offered by a minimum base line cluster.

Fault resilience. Similar to scalability, further decomponentization of .NET enables fewer single points of failure and more opportunities for failover among components. The services in the context of a public data center environment like an *msn* environment, have traditional fault resilience support, such as backbone providers, multiple network paths into the physical data center, router hierarchy, multiple paths inside the data network, redundant switching, redundant stateless load balancers in the front-end. What ever applies to traditional Web services applies to .NET My Services. The designers of .NET My Services are not trying to hide the protocol behind the object layer. The fact that the protocol has visible redirects means that the clients participates in the fault recover. The protocol itself is stateless: there are no handles or sessions and everything is either a request or a request/response [Lucovsky 2002].

Transparency and usability. As already discussed in the scalability section, .NET My Services design has pushed information into the client side, similarly to traditional Web serving redirects. When there is a redirect on the Web, it is transparent to the user. However, the browser, or in the case of the layered stack a WinInet, might get redirected. Similarly, DNS supports client and server side resolution, however for scalability reasons the client resolution is used. .NET My Services follow the same basic pattern. It is transparent at one level of API, but it is visible at the core protocol. This design saves cycles on the back-end. It is less transparent at the protocol level, but user non-transparency is hidden at the software layer above the protocol.

Business model. .NET expands Microsoft's strategy from a license-based one to include a subscription-based [Lucovsky 2002]. For example, an Exchange server can be offered to large corporations through licenses, or email service can be hosted as businesses. Both approaches are valid for any software, service, or for any server that runs them. .NET is also one of the first times that Microsoft embraced the Internet model in its entirety including open-source standards, such as XML, SOAP, WSDL, and UDDI, as well as MS submission of the C# and Common Language Infrastructure to standards body European Computer Manufacturers Association (ECMA).

Interoperability. Because they are based on Web service protocols, .NET My Services are very interoperable.

Applications. The major advantage of .NET My Services compared to other solutions is the wealth of applications. Microsoft will port all of the Microsoft Office applications to .NET, including the development tools.

P2P System	System Feature					
	System Category	Alternative Solution	Platform	Languages and Tools	Distinctive Features	Target Networks
Avaki	distributed computing	single installation HPC and supercomputers	Linux, Solaris, MS Windows	OO, paral. lang. obj wrap., legacy app x-comp. (Fortran, Ada, C)	dist. admin ctrl, heterogeneity, secure access, high paral. exec	Internet, intranet
SETI@home		distributed objects	all common OS supported	closed source	large scale	Internet
Groove	collaboration	Web-based collaboration	MS Windows	JavaScript, VB, Perl, SOAP, C++, XML	Synchronization, security, self-updating, multi-identities	Internet, intranet
Magi		publishing & authoring dist file sys, chat, messaging	Windows, Linux, Solaris, HPUX, Mac	Java, XML, HTTP, WebDAV	HTTP based, platform indep.	Internet, Extraprise, ad-hoc
FreeNet	content distribution	anonymity: none. others centralized & single point of trust	any with Java	Java implementation and APIs	Preservation of anonymity	Internet
Gnutella		central servers	Windows, Linux	Java, C	protocol	Internet
JXTA	platform	client-server	Solaris, Linux, MS Windows	Java , C, Perl	open-source effort	Internet
.NET / My Services		Web-based	Windows (backend) any front-end	C#, VC++, JScript, VBScript, VB	widespread MS apps base	Internet and mobile

Table 6. Summary of Case Studies.

Lessons learned. .NET and .NET Services are fairly new to be able to derive any substantial lessons learned without a deployed product. Nevertheless, some lessons can already be derived. First, getting clarity on the deployment is very important. MS was not explicit on the business model from the very beginning: was it targeted to a private individual, corporate running on its own, or as a set of services. Next, the passport model raised concerns about security problems, but it has also attracted a huge number of users so far. Centralized authentication through passport was eventually decentralized and federated, but this is not enough, it is required to go to the next level and have a true edge-based authentication. Third, it is the applications that attract users; .NET seems to follow the same model as with MS DOS, where it is really the applications, such as Excel and Word that are of interest to users, not the infrastructure itself. Finally, going further down the P2P rout is something that any scalable service has to be able to run at a data center, in an enterprise, hosted for a small businesses, or at a home. A full stack of capabilities is required to run on a full range [Lucovsky 2002].

6.9 Summary

This Section summarizes the case studies. Table 6 summarizes the general characteristics of all of the case studies. The entries are self explanatory. P2P systems are deployed on a variety of systems and support different languages and standard protocols. They are largely targeted for the Internet and some are targeted for mobile settings.

Table 7 summarizes the case studies from the perspective of the characteristics described in Section 4. There are various decentralization models, including pure (FreeNet, Gnutella, JXTA), hybrid (Groove, Magi), master-slave (SETI@home), and mixed (.NET). Few systems support a high degree of anonymity (FreeNet). A number of them support some sort of self-organization and most of them improve cost of ownership. Ad-hoc nature is predominantly supported through the ability of the peers to join and leave. Performance is improved primarily in the distributed computing type of P2P systems (Avaki, SETI@home). Standard security mechanisms are supported, such as encryption, authentication, and authorization, and some of them protect against denial of service attacks (Freenet).

Transparency is primarily supported for the benefit of disconnection and to hide communication/computation heterogeneity (Groove, Avaki). Fault resilience is supported through checkpoint-restart mechanisms (Avaki, SETI@home), queued messages (Groove, Magi), and in general by the redundancy of components and the elimination of a single point of failure. Most systems interoperate by relying on standard communication stacks (IP) and on Web standards (SOAP, XML, UDDI, WSDL, and WebDAV). There are a only a couple of exceptions of interoperability with other P2P systems, such as Avaki with Sun's Grid, and Magi with JXTA.

Table 8 summarizes the business aspect of the case studies. The surveyed case studies belong both to proprietary (Groove, .NET) and open source systems (FreeNet, Gnutella, JXTA), or they are offered as either (Avaki, Magi). They support applications in their respective domains, such as distributed computing, communication/collaboration, and file sharing. P2P systems classified as platforms (.NET My Services, JXT, and to a certain extent Groove and Magi) support a wider base of applications.

The case studies we selected have a closed base of customers with the exception of .NET My Services, which relies on its earlier customer base. Also, they are competitors within each class of P2P systems. These systems

P2P System	System Feature										
	Decentralization	**Scalability**	**Anonymity**	**Self-Organization**	**Cost of Ownership**	**Ad-hoc**	**Performance**	**Security**	**Transparency**	**Fault Resilience**	**Interoperability**
Avaki	distributed, no central mgr	scale to 1000s 2.5-3k tested	not permitted	restructures around failure	low	join/leave compute resources	speedups	encryp, authentication, adm. domains	location; HW/SW heterog.	chckpt/restart reliable msg	interoperates with Sun Grid
SETI@home	master-slave	millions	medium	low	very low	join/leave compute resources	huge speedups	proprietary	high	timed chckpt	IP?
Groove	hybrid P2P	~25 shared spaces Web scale globally	no support but possible	high	low	join/leave of collaborators	medium	shared-spaces authn/authr, encryp	high	queued messages	IP-based
Magi	hybrid P2P	Web scalability	not permitted	buddies & groups	low	join/leave of group/buddies	N/A	simple PKI & 3rd party plugins	location & naming	hybrid server failover	JXTA, WebDAV, HTTP, Java, XML
FreeNet	pure P2P	theoret. scales ~ log(size_network)	high	high	low	join/leave of peers	medium	f anonymity & preventing DoS	high	No 1 point of failure, replic.	low
Gnutella	pure P2P	thousands	low	high	low	join/leave of peers	low	not addressed	medium	resume download	IP?
JXTA	pure P2P	also addresses embedded systems	N/A	N/A	low	join/leave of peers	N/A	crypto algor. distr. trust model	low	low	low
.NET / My Services	mixed	world-scale	not permitted	medium	low	join/leave of peers	high	Kerberos-based authen, encryp, sign	high	replication	SOAP, XML, UDDI, WSDL

Table 7. Comparison of Characteristics of Case Studies.

have different funding model: startups (Magi, Freenet, government funded (SETI@home), public domain efforts (Gnutella, JXTA), or privately owned companies (.NET My Services).

Finally, they have different business models, ranging from becoming an all-encompassing pervasive platform of choice (.NET My Services), or becoming a collaborative platform of choice (Groove, Magi), to selling more computers and using ads on screen savers (SETI@home) and becoming a P2P algorithm of choice.

7 LESSONS LEARNED

Peer-to-peer is a relatively new technology that is not yet proven in the research community and in industry. Therefore, it may be too early to make a firm statement on some of the lessons learned. Nevertheless, we can make some preliminary observations. Some of them are intuitive, while others are not obvious. We divide the observations into P2P strengths and weaknesses, P2P non-technical challenges, and implications on users, developers, and IT.

7.1 Strengths and Weaknesses

A comparison of P2P with its alternatives, centralized and client-server systems, is summarized in Table 9. P2P systems are designed with the goals of decentralization, ad-hoc connectivity, reduced cost of ownership, and anonymity. P2P has more decentralized control and data compared to alternatives; it supports systems whose parts can come and go and can communicate in an ad-hoc manner; the cost of ownership is distributed among the peers; and the peers can be anonymous. Compared to P2P, centralized systems are inherently centralized and client-server systems have centralized points of control and data at the servers.

It is harder to compare P2P with alternatives in terms of scalability, performance, security, self-organization, and fault-tolerance. It is our speculation that because of high-

P2PSystem	System Feature					
	Revenue Model	**Supported Applications**	**Known Customers**	**Competitors**	**Funding**	**Business Model**
Avaki	product and open-source	computation grid, shared secure data access	none. evaluated at several life sciences labs	Platform Computing Globus	startup	solution for hi-tech companies (distributed computing)
SETI@home	Academic	Closed	Academic	cancer@home fight_aids@home	government	sell more computers ads on screen savers
Groove	product & platform	purchasing, inventory, auctions, etc.	Dell, Department of Defense, GlaxoSmithKline and Veridian	Magi	startup	x-enterprise decentr. collab. get bus. processes out of email
Magi	product	Search, shared files, messaging, chat	Altavista	Groove, eRooms, XDegrees	funding through revenues from other company	enterprise sales (end-user product & embedded with apps)
FreeNet	open-source	file sharing	public	N/A	startup	enterprise sales
Gnutella	open-source	file sharing	public	N/A	public domain	algorithm of choice for P2P
JXTA	open-source proprietary extensions	file sharing, messaging, event notific., email	Many P2P systems ported to JXTA	.NET/My Services	public domain, supported by Sun Microsystems	de-facto common P2P platform
.NET / .NET My Services	proprietary & open-source standards	MIcrosoft Office and more	large base of MS customers	AOL, Sun J2EE/JXTA, Ximian MONO	MS internally	de-facto pervasive platform

Table 8. Business Comparison of Case Studies.

System/App Requirements	Type of System		
	Centralized	**Client-Server**	**Peer-to-Peer**
decentralization	low (none)	high	very high
ad-hoc connectivity	no	medium	high
cost of ownership	very high	high	low
anonymity	low (none)	medium	very high
scalability	low	high	high
performance	individual high aggregate low	medium	individual low aggregate high
fault resilience	individual high aggregate low	medium	individual low aggregate high
self-organization	medium	medium	medium
transparency	low	medium	medium
security	very high	high	low
interoperability	standardized	standardized	in progress

Table 9. Comparison of Solutions. *Darker shading represents inherent capabilities or potential. Actual implementation and user requirements determine level of achieved capability. For example, anonymity is typically not a user requirement in enterprises, even if potential for supporting it is high. Similarly, security requirements and implementation can be very high in enterprises even if they are low compared to centralized solutions.*

er decentralization P2P can better satisfy these requirements as well. P2P has the potential to be more scalable than centralized and client-server solutions. However, P2P systems are highly dependent on the underlying topology and the types of applications. For example, mainframe computers are still competitive for transaction-based computing because they are optimized for this type of applications. On the other hand, P2P system are very suitable for large number of computers on the Internet, for mobile wireless users, or for device sensor networks. A similar reasoning applies to performance and fault-resilience. Individually, centralized systems have highly optimized performance and fault-tolerance, followed by client-server systems, and only then by P2P systems. The opposite is true for aggregate systems where P2P has higher aggregate performance and fault resilience by avoiding the need to interact with servers and having fewer points of failure.

It is unclear whether any of the systems has a clear advantage in self-organization and transparency. There has been long-standing work in improving these characteristics in both centralized systems and client-server systems. Self-organization is especially critical in Internet Data Centers, and mainframe computers also implement various forms of adaptation. Transparency has been the focus of client-server systems, in particular location transparency. While also a goal of P2P, transparency has not been sufficiently developed in P2P systems. Many actions require user intervention, such as connecting to a specific network of P2P systems, providing knowledge about the files and users, and understanding the failure model.

Finally, we believe that in two regards P2P lags behind its alternatives. Inherently, P2P systems expose more security threats than centralized and client-server models. Because of the nature of peer-to-peer interaction, sufficient guarantees or additional trust has to be established among the peers. Interoperability is matter of investment and as development proceeds more interoperability may evolve. At the time of writing, client-server systems support the most interoperability.

7.2 Non-Technical Challenges

In addition to all of the technical motivations and difficulties with developing P2P systems, there are also non-technical challenges to the success of P2P. In spite of excellent technology, if these challenges are not overcome, P2P will likely remain an approach used only in small niches or by specific communities.

The chief challenge of P2P systems is *acceptance and use*. Because peers rely on one another to provide service, it is essential that numerous peers be available for the service to be useful. By comparison, centralized or client-server environments are potentially useful as long as the service provider keeps the service running. This is not the case in P2P systems. If the peers abandon the system, there are no services available to anyone.

P2P systems live and die on their network effects, which draw in more and more users. The value of network effects was informally specified by Metcalfe's Law, which states that "the utility of a network grows with the square of the number of users." While we cannot truly quantify utility to prove the law, it resonates clearly with the idea that more users make a system more useful.

Studies have also shown that all users are not alike, and that some users can actually damage a system. Adar and Huberman [2000] showed that in the Gnutella system, many users download files, but few actual provide files. Further, those with poor resources, such as bandwidth, can actually slow the network down by becoming bottlenecks for the system as a whole. This situation has been likened to the tragedy of the commons where poorly behaving users of a free, shared resource make the resource unusable by all.

Solutions to this problem revolve around building an economy around the use of the shared resource. One example of this is MojoNation [2001]. In MojoNation, users accumulate a form of currency called "mojo" by providing service to others. In the MojoNation scheme, this currency would be redeemable for other services or potentially from real-world vendors in the form of gift certificates or related benefits. However, implementing a MojoNation type scheme requires accounting to be performed, and this can limit anonymity and other potential benefits of P2P systems.

Related to acceptance and use is the *danger of fragmentation* of the user base. Typically, individuals are only going to participate in one or a very few different P2P systems because they simply don't have the resources to support multiple systems at the same time. Because of this, as each new system is introduced it fragments the user base and can potentially damage the value of all P2P systems. Consider the success of Napster, which was arguably not the most technically sound P2P system developed, but was the first to gather a large user base. Therefore, until it was forcibly shutdown, it thrived because the network effects continued to draw more and more users.

Since Napster's demise, a number of music sharing systems, such as KaZaa, have been developed and are both free and technically sound. However, neither has yet become the one truly dominant system, so neither has become as truly useful as Napster was in its prime.

Instant messaging faces a similar difficulty. While there are numerous IM systems, each with a significant user base, the overall utility of IM is limited because of fragmentation. Typically, to communicate with everyone a user wishes to reach, the user must maintain accounts and run clients from many IM systems at the same time. Interoperability, as discussed in Section 4.11, seems to be the only solution to this problem.

While P2P systems rely on scale for success, *scale* is also a significant challenge. In centralized systems, problems related to scale are relatively well understood and solutions are pretty well known. In the worst case, bigger, faster computing platforms are an option for improving scale. Decentralized P2P systems more often require algorithmic solutions to problems of scale. There is no central place to simply throw more computing resources. These distributed algorithms tend to be some of the most difficult to develop because they require decisions to be made at each local peer, usually with little global knowledge.

Target	Criteria	Type of System		
		Centralized	Client-Server	P2P
User	Pervasiveness	low	medium	high
	State-of-the-art	low	high	medium
	Complexity	high	low	medium
	Trust & Reputation	high	medium	low
Developer	Complexity	high	straightforward	typical - no atypical - yes
	Sustainability	low	high	medium
	Tools	medium (proprietary)	high (standardized)	low (few tools)
	Compatibility	medium	high	low
IT	Accountability	high	medium	low
	Being in control	high (fully)	medium	low
	Manageability	medium	high	low
	Standards	medium (proprietary)	high	low (inexistent)

Table 10. Comparison of Solutions. *Darker shading represents inherent capabilities and potential.*

A final challenge involves the *release of control*. In centralized systems, the operator of the system has some control of the system. For example, an operator can monitor each transaction and potentially determine who initiated the transaction and how long it took. In a decentralized system, no such monitoring is possible. This scares many providers of traditional services so they resist P2P schemes. Certainly, the release of control can be seen as part of the reason for the Recording Industry Association of America's (RIAA) lawsuit against Napster. Napster provides an alternative method for music distribution. While copyright, payment, and fair-use issues are certainly at the center of the case, embracing a P2P approach to distribution, such as Napster, also implies a release of control of the distribution channel. This may explain why the RIAA and Napster have not been able to reach an agreement.

7.3 Implications for Users, Developers, and IT

P2P is not a solution for every future system and application. As we have seen in the previous two subsections, it has both strengths and weaknesses. In this section, we evaluate the implications that P2P has for users, developers, and IT departments. In Table 10, we compare P2P with its alternatives. P2P has the following implications for the users of P2P systems and applications: pervasiveness, complexity of use, state of the art, and trust and reputation.

Pervasiveness is becoming more and more important. While it may not be possible to access traditional services at any point on Earth at any time, P2P offers opportunities in this regard by relying on peers to provide services when there is no other infrastructure or means of access available. For example, cellular connectivity may not be available to the server, but a peer can offer the service locally.

Traditional centralized solutions have complex support compared to client-server solutions. P2P solutions are not as simple or as well-understood as the client-server model, yet wide deployment and use of Napster, Gnutella-based solutions, and other startups offers potential in this regard.

Currently, the client-server model represents the state of the art, but there is a lot of ongoing development in P2P research – second generation P2P systems (CAN, Pastry, Chord, Tapestry, etc.); open-source – JXTA; proprietary systems – .NET; and standards – P2PWG.

From the user perspective, P2P is probably weakest in the sense trust and reputation. Owners want to have trust in the service they are using, so they will use only reputable sites unless price is the main objective or the service is free. Centralized systems and the client-server model have traditionally built up trust and reputation, and this is a concern for users of P2P.

Developers are also concerned with complexity, as well as with the sustainability of solutions, with the availability of the tools, and the compatibility of the systems. In this comparison, the client-server dominate over the other solutions.

Client-server systems are well-understood and documented for developers. P2P systems offer promise in simple cases, such as document exchange, but for more complex requirements, such as collaborative applications, they require more complex algorithms and understanding. It is a similar case of sustainability. In the long term, centralized solutions may not be as sustainable as peer-to-peer solutions, but the client-server model will continue to be supported. Finally, the weakest aspect of P2P is the lack of tools and compatibility across various P2P systems.

P2P has the following implications for IT: accountability, being in control, manageability, and standards. The first three are very closely tied. Accountability is emphasized in centralized systems where access is monitored through logins, accounts, and the logging of activities. Accountability is more difficult to achieve in client-server systems, because of interactions with multiple clients. It is weakest in P2P systems, because of equal rights and functionality among the peers. Similar reasoning applies for being in control. In centralized and client-server systems, control is exercised at one or more well-defined points, whereas it is harder to achieve in P2P systems, where control is entirely distributed.

A similar situation exists for manageability. There are a lot of tools for the management of client-server systems, somewhat fewer for centralized systems, and fewest for P2P. Similar applies for standards. Most standards have been developed for client-server systems and very few for P2P systems.

While Table 9, compares the potential of P2P versus its alternatives in terms of the characteristics, Table 10 summarizes the existing implications of P2P on users, developers, and IT. In summary, there is a lot of potential for P2P, but it has not yet been realized.

8 SUMMARY AND FUTURE WORK

In this paper, we surveyed the field of P2P systems. We defined the field through terminology, architectures, goals, components, and challenges. We also introduced taxonomies for P2P systems, applications, and markets. Based on this information, we summarized P2P system characteristics. Then, we surveyed different P2P system categories, as well as P2P markets. Out of systems presented in Section 5, we selected eight case studies and described them in more detail. We also compared them based on the characteristics we introduced in Section 4. Based on this information, we derived some lessons about P2P applications and systems.

In the rest of this section, we revisit what P2P is, we explain why we think that P2P is an important technology, and finally we present the outlook for the P2P future.

8.1 Final Thoughts on What P2P Is

One of the most contentious aspects in writing this paper was to define what P2P is and what it is not. Even after completing this effort, we do not feel compelled to offer a concise definition and a recipe of what P2P is and what it is not. A simple answer is that P2P is many things to many people and it is not possible to come up with a simplified answer. P2P is a mind set, a model, an implementation choice, and property of a system or an environment.

- **A mind set.** As a mind set, P2P is a system and/or application that either (1) takes advantage of resources at the edge of the system or (2) supports direct interaction among its users. Such a system and/or application remains P2P regardless of its model or implementation. Examples include SETI@home, which is considered

to have a client-server model, but displays the first mind set property, and Slashdot [2002], which enables the second mind set property, but really has a centralized implementation.

- **A model.** A system and/or application supporting the model presented in Figure 1 is P2P. In its purest form, P2P is represented by Gnutella, Freenet, and Groove. According to this, SETI@home does not have a P2P model, whereas Napster has a hybrid model.
- **An implementation choice.** P2P systems and applications can be implemented in a P2P way, such as JXTA or Magi. However, a non-P2P application can also be implemented in a P2P way. For example, application-layer multicast can have a P2P implementation, and parts of the ORBs or DNS servers are also implemented in a P2P way.
- **A property of a system or an environment.** .NET as well as environments, such as small device sensor networks, may require P2P implementation solutions while not necessarily supporting a P2P application. P2P solutions may be required for scalability, performance, or simply because of the lack of any kind of infrastructure, making P2P the only way to communicate. This is similar to looking at P2P as an implementation choice, however in this case P2P is the *forced* implementation choice.

8.2 Why We Think P2P is Important

As P2P becomes more mature, its future infrastructures will improve. There will be increased interoperability, more connections to the (Internet) world, and more robust software and hardware. Nevertheless, some inherent problems will remain. P2P will remain an important approach for the following reasons.

- Scalability will always be a problem at certain levels (network, system, and application), especially with global connectivity, much of it wireless. It will be hard to predict and guarantee all service-level agreements. P2P can contribute to each area.
- Certain parts of the world will not be covered by (sufficient) connectivity, requiring ad-hoc, decentralized groups to be formed. P2P is a well-suited alternative when there is a lack of infrastructure.
- Certain configurations of systems and applications will inherently be P2P and will lend themselves to P2P solutions.

8.3 P2P in the Future

The authors of this paper believe that there are at least three ways in which P2P may have impact in the future:

- **P2P algorithms** probably have the biggest chance of making impact. As the world becomes increasingly decentralized and connected, there will be a growing need for P2P algorithms to overcome the scalability, anonymity, and connectivity problems.
- **P2P applications** are the next most likely to succeed in the future. Examples, such as Napster are a convincing proof of such a possibility.
- **P2P platforms** are the third possible scenario for P2P. Platforms such as JXTA may be widely adopted, in which case many other P2P systems can also gain wide adoption.

8.4 Summary

We believe that P2P is an important technology that has already found its way into existing products and research projects. It will remain an important solution to certain inherent problems in distributed systems. P2P is not a solution to every problem in the future of computing. Alternatives to P2P are traditional technologies, such as centralized systems and the client-server model. Systems and applications do not necessarily have to be monolithic, they can participate in different degrees in the centralized/client-server/P2P paradigms. P2P will continue to be a strong alternative for scalability, anonymity, and fault resilience requirements. P2P algorithms, applications, and platforms have an opportunity for deployment in the future. From the market perspective, cost of ownership may be the driving factor for P2P. The strong presence of P2P products indicates that P2P is not only an interesting research technology but also a promising product base.

ACKNOWLEDGMENTS

The developers of eight case studies spent the time with us in detailed analysis of their systems. We are indebted to David Anderson, Greg Bolcer, Andrew Grimshaw, Li Gong, Mark Lucovsky, Jack Ozzie, Walt Tuvell, Dan Wortheimer, and Bill Yeager. We are also very thankful to Fabio Casati, Denis Chalon, Fred Douglis, Ira Greenberg, Martin Griss, Tim Kindberg, Alan Karp, Raj Kumar, Mark Lillibridge, Alan Messer, Jeff Morgan, Chandrakant Patel, Todd Poynor, Steve Richardson, Stéphanie Riché, Sumit Roy, Jim Rowson, Yasushi Saito, Patrick Scaglia, Ming-Chien Shan, and Zheng Zhang for reviewing various versions of this paper. Their comments significantly improved the content and presentation.

REFERENCES

Adamic, L. The Small World Web. 2000. *Technical Report*, Xerox Palo Alto Research Center.

Adar, E. and Huberman, B. 2000. Free Riding on Gnutella.

First Monday, vol 5, no 10 (October). (see also www.firstmonday.dk/issues/issue5_10/adar.)

Akamai 2001. www.akamai.com.

Albitz, P, and Liu. C. 1995. DNS and BIND. O'Reilly and Associates.

Anderson, D., 2002. Personal Communication.

Anderson, D., Cobb, J., Korpela, E., Lebofsky, M., Werthimer, D. 2002. SETI@home: An Experiment in Public-Resource Computing. To appear in CACM, November 2002.

AOL 2001. www.aol.com.

Applied MetaComputing. 2000. Legion - An Integrated Architecture for Secure Resource Sharing. Applied Meta Computing *Peer-to-Peer Architectural Proposal*.

Ariba, IBM, and Microsoft. 2000. UDDI Technical White Paper, available at www.uddi.org, September 2000.

Avaki Corporation. 2001. Avaki 2.0 Concepts and Architecture. White paper. www.avaki.com/papers/AVAKI_concepts_architecture.pdf.

BearShare 2001. www.bearshare.com.

Ball, T. and Rajamani, S. K. (2001) Automatically validating temporal safety properties of interfaces. In the Proceedings of the International SPIN Workshop on Model Checking of Software, pp. 333-344, May 2001.

Barak, A. and Litman, A. 1985. MOS: a Multicomputer Distributed Operating System. *Software - Practice and Experience*, 15(8):725–737. August.

Barak, A. and Wheeler, R. 1989. MOSIX: An Integrated Multiprocessor UNIX. *Proceedings of the Winter 1989 USENIX Conference*, pages 101–112. February.

Barkai, D., 2001. Peer-to-Peer Computing, Technologies for Sharing and Collaborating on the Net. Intel Corporation.

Becker D.J., Sterling T., Savarese D., Dorband J.E., Ranawak U.A., Packer C.V. 1995. "Beowulf: A Parallel Workstation for Scientific Computation", Proceedings of ICPP.

Berners-Lee T. Fielding, R., Masinter, L. 1998. Uniform Resource Identifiers (URI): Generic Syntax. IETF Request for Comments. August 1998

Bernstein, P. A. 1996. Middleware: A Model for Distributed System Services. *CACM*, 39(2):86–98.

Bildson G. 2002. Personal Communication.

Bolcer, G. 2002. Personal communication.

Bolcer, G., et al. 2000. Peer-to-Peer Architectures and the Magi Open Source Infrastructure. *Endeavors technologies White Paper*.

Bolcer, G. 2001. Magi: An Architecture for Mobile and Disconnected Workflow. *White paper* (www.endeavors.com)

Bolosky, W., Douceur, J., Ely, D., and Theimer, M. 2000. Feasibility of a Serverless Distributed File System Deployed on an Existing Set of Desktop PCs. *Proceedings of SIGMETRICS*, Santa Clara, CA, USA, June, 2000.

Box, D. Ehnebuske, D., Kakivaya, G., Layman, A. Mendelsohn, N., Nielsen, H.F., Thatte, S., Winer. D. 2000. Simple Object Access Protocol (SOAP) 1.1. W3C Note 08 May 2000.

Bray, T., Paoli, J., Sperberg-McQueen, C. M., Maler, E. (Editors). 2000. Extensible Markup Language (XML) 1.0 (Second Edition) W3C Recommendation 6 October 2000.

Britton, C. 2000. IT Architectures and Middleware: Strategies for Building Large, Integrated Systems. Addison Wesley.

Carpenter. B. 2000. Internet Transparency. Internet Society Request for Comments 2775. www.faqs.org/rfcs/rfc2775.html.

Castro, M. and Liskov, B. 1999. Practical Byzantine Fault Tolerance. *Proc. Usenix OSDI*, Berkeley, California.

Cavallar, S., Dodson, B., Lenstra, A.K., Lioen, W., Montgomery, P.L., Murphy, B., te Riele, H., Aardal, K., Gilchrist, J., Guillerm, G., Leyland, P. 2000. Factorization of a 512-bit RSA Modulus. Proceedings of EuroCrypt 2000, Bruges (Brugge), Belgium.Chaum, D. 1981. Untraceable Electronic Mail Return Addresses and Digital Pseudonyms, Communication of the ACM 24, 2, Feb. 1981, pp. 84-88.

Christensen, E., Curbera, F., Meredith, G., Weerawarana, S. 2001. Web Services Description Language (WSDL) 1.1, W3C Note 15 March 2001. www.w3.org/TR/wsdl.

Clarke, I. 1999. A Distributed Decentralized Information Storage and Retrieval System, unpublished report, Division of Informatics, University of Edinburgh. freenet.sourceforge.net/freenet.pdf.

Clarke, I., Sandberg, O, Wiley, B., Hong, T.W. 2001. Freenet: A Distributed Anonymous Information Storage and Retrieval System. *Designing Privacy Enhancing Technologies: International Workshop on Design Issues in Anonymity and Unobservability*, LNCS 2009, ed. by H. Federrath. Springer: New York.

Clip 2 Marketing 2000. Gnutella: To the Bandwidth Barrier and Beyond. www.clip2.com/gnutella.html.

Couloris, G., Dollimore, J., and Kindberg, T., 2001. Distributed Systems. Concepts and Design. Addison Wesley.

Dabek, F., Brunskill, E. Kaashoek, F., Karger, D., Morris, R., Stoica, I., and Balakrishnan, H. 2001. Building Peer-to-Peer Systems With Chord, a Distributed Lookup Service. *Proc. of the 8th Workshop on Hot Topics in Operating Systems (HotOS-VIII)*, Schloss Elmau, Germany, May 2001.

DataSynapse 2001. The Next Generation of High Performance Computing For Financial Services. White Paper.

Denning. D. E. 1976. A Lattice Model of Secure Information Flow. Communications of the ACM 19(5):236-243, May 1976.

Dingledine, R., Freedman, M., Rubin, A. 2001. Free Haven. In Oram, A., 2001 Peer-to-Peer, Harnessing the Power of Disruptive Technologies, pp 159-187.

Druschel P. and Rowstron, A. 2001. PAST: A Large-Scale, Persistent Peer-to-Peer Storage Utility, HotOS VIII, Schloss Elmau, Germany, May 2001.

Endeavors Technology 2001. www.endeavors.com.

Entropia 2002. www.entropia.com.

FastTrack. 2001. Product Description. www.fasttrack.nu/index_int.html.

Fattah, H.M., Fattah, H.M. 2002. P2P: How Peer-to-Peer Tech-

nology Is Revolutionizing the Way We Do Business, Drabber Trade

Foster, I. 2000. Internet Computing and the emerging Grid. Nature, Dec. 7.

Foster, I., Kesselman, C. 1999. The Grid: Blueprint for a New Computing Infrastructure. Morgan Kaufman Publishers, Inc. San Francisco, California.

Folding@Home 2001. foldingathome.stanford.edu.

FreeNet. 2001. The FreeNet home page, freenet.sourceforge.net. www.freenetproject.org.

Gabber, E., Gibbons, P., Kristol, D., Matias, Y., and Mayer, A. 1999. Consistent, yet anonymous, Web access with LPWA. Communications of the ACM. 42, 2. February 1999. pp. 42-47.

Genome@Home. 2001. genomeathome.stanford.edu.

Gnutella. 2001. The Gnutella home page, gnutella.wego.com.

Gong, L. 2001. JXTA: A Network Programming Environment. IEEE Internet Computing, (5)3:88--95, May/June.

Gong, L., 2002. Personal Communication.

Gribble, S., Halevy, A., Ives, Z., Rodrig, M., and Suciu, D. 2001. What Can Peer-to-Peer Do for Databases and Vice Versa? *Proceedings of the WebDB: Workshop on Databases and the Web, Santa Barbara, CA, USA.*

Grid Computing Biotechnology. 2001. www.grid-computing.net.

Grid Forum. 2002. www.gridforum.org.

Grimshaw, A., Wulf, W.A., and the Legion Team. 1997. The Legion Vision of a Worldwide Virtual Computer. *Communications of the ACM, 40(1):39-45.*

Grimshaw, A., Easy to use Object-oriented Parallel Programming with Mentat. *IEEE Computer, pp39-51, May 1993.*

Grimshaw, A. S., Wulf, W. A., French, J. C., Weaver, A. C., Reynolds Jr., P. F. 1994. Legion: The Next Logical Step Toward a Nation-wide Virtual Computer. UVa CS Technical Report CS-94-21, June 8, 1994.

Grokster 2001. www.grokster.com

Groove Networks. 2000. Introduction to Groove. *Groove Networks White Paper*.

Groove Networks. 2000a. Connection Age. *Groove Networks White Paper*.

Groove Networks. 2001. Groove Networks Product Backgrounder. *Groove Networks White Paper*. www.groove.net/pdf/groove_product_backgrounder.pdf.

Heylighen, F. 1997. Principa Cybernetica Web. pespmc1.vub.ac.be/SELFORG.html.

Howard, J.H., Kazar, M.L., Menees, S.G., Nichols, D.A., Satyanarayanan, M., Sidebotham, R.N., West, M.J. 1988. Scale and Performance in a Distributed File System. *ACM Transactions on Computer Systems, Feb. 1988, Vol. 6, No. 1, pp. 51-81.*

Howes, T., Smith, M. 1997. LDAP. Programming Directory-Enabled Applications with Lightweight Directory Access Protocol. MacMillan Technical Publishing, USA.

IEEE 1990. Institute of Electrical and Electronics Engineers. IEEE Standard Computer Dictionary: A Compilation of IEEE Standard Computer Glossaries. New York, NY: 1990.

IEEE 2001. www.ieee.org.

Intel 2001. Peer-to-Peer-Enabled Distributed Computing. *Intel White Paper*.

JXTA 2001.The JXTA home page www.jxta.org.

Kan, G. 2002 Personal Communication.

Katzenbeisser, S. 1999. Information hiding techniques for steganography and digital watermarking. *Artech House Books.*

Kaufman, C., Perlman, R., Spencer, M. 1995. Network Security. Prentice Hall.

KaZaA. www.kazaa.com. 2001.

Kindberg, T. 2002. Personal Communication.

Kotzen, M. 2001. Dramatic Improvements in the Gnutella Network Since Late 2000. www.limewire.com/index.jsp/net_improvements.

Kubiatowicz, J., Bindel, D., Chen, Y., Czerwinski, S., Eaton, P., Geels, D., Gummadi, R., Rhea, R., Weatherspoon, H., Weimer, W., Wells, C., and Zhao, B. 2000. OceanStore: An Architecture for Global-Scale Persistent Storage. *Proceedings of the Ninth international Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS 2000)*, November 2000.

Leigh, P. and Benyola, P., 2001. Future Developments in Peer Networking. Equity Research, *White Paper*, Raymond James & Associates, INC.

Limewire 2001. www.limewire.com.

Litzkow, M. and Solomon, M. 1992. Supporting Checkpointing and Process Migration outside the UNIX Kernel. *Proceedings of the USENIX Winter Conference, pages 283–290.*

Litzkow, M., Livny, M., and Mutka, M. June 1988. Condor - A Hunter of Idle Workstations. *Proceedings of the 8th International Conference on Distributed Computing Systems*, pages 104–111.

Lucovsky, M., 2002. Personal Communication.

Microsoft. 2001. Microsoft .NET Passport Technical Overview. September 2001.

Milgram, S. 1967. The Small World Problem. Psychology today. 1, 61.

Miller, M. 2001. Discovering P2P. Sybex, to appear.

Mockapetris, P. 1989. DNS Encoding of Network Names and Other Types. *Internet Request For Comment 1101* Network Information Center, SRI International, Menlo Park, California.

MojoNation. 2001. http://mojonation.net.

Moore, D., Hebeler, J. 2001. Peer-to-Peer: Building Secure, Scalable, and Manageable Networks. McGraw Hill.

Morgan, J. 2002. Personal Communication.

Morpheus 2001. www.musiccity.com

Napster. 2001. The Napster home page, www.napster.com.

Natarajan. A., et al, Studying Protein Folding on the Grid: Ex-

periences Using CHARMM on NPACI under Legion. *High Performance Distributed Computing 10, August 7-9, 2001*

Necula, G. 1997. Proof-Carrying Code. Proceedings of the 24th Annual ACM Symposium on Principles of Programming Languages, pp 106-119. Paris France, January 1997.

Necula, G., and Lee, P. 1998. The Design and Implementation of a Certifying Compiler. Proceedings of the ACM SIGPLAN Conference on Programming Language Design and Implementation, pp 106-119. Montreal, Canada, pp. 333-344, June 1998.

Nethisinghe, S., 2001. TV or the PC? Consumer Perspective. Phillips Presentation.

Nowitz, D. 1978. UUCP Implementation Description. UNIX Programmer's Manual, Bell Laboratories, October, 1978.

OMG. 1996. Common Object Request Broker Architecture and Specification. *Object Management Group Document Number 96.03.04*.

OpenCola 2001. www.opencola.com.

Oram, A. 2001. Peer-To-Peer, Harnessing the Power of Disruptive Technology. O'Reilly.

Perkins, C., 2001. Ad Hoc Networking. Addison Wesley.

p2pwg. 2001. Peer-to-peer Working Group. www.p2pwg.org.

Peer-to-peer Working Group. 2001. Bidirectional Peer-to-Peer Communication with Interposing Firewalls and NATs. p2pwg White Paper, Revision 0.091. May 23, 2001. http://www.peer-to-peerwg.org/tech/nat/.

Pfitzmann, A., Waidner, M. 1987. Networks without User Observability. Computer Security 2, 6, pp.158-166.

Platform Computing. 2001. www.platform.com.

Ramanathan, M. K., Kalogeraki, V. Pruyne, J. 2001. Finding Good Peers in the Peer-to-Peer Networks. International Parallel and Distributed Computing Symposium, Fort Lauderdale, Florida, April 2002.

Ratnasamy, S., Francis, P., Handley, M., Karp, R., Shenker, S. 2001. A Scalable Content-Addressable Network. *Proceedings of the SIGCOMM, pp 161-172.*

Redshift Research. 2002. www.redshiftresearch.com.

Reiter, M. K., Rubin, A. D. 1998. Crowds: Anonymity for Web Transactions. ACM Transaction on Information and System Security 1, 1, November 1998, pp 66-92.

Rhea, S., Wells, C., Eaton, P., Geels, D., Zhao, B., Weatherspoon, H., and Kubiatowicz, J. 2001. Maintenance-Free Global Data Storage. Internet Computing, vol. vol 5, no 4. pp 40-49. September 2001

Rice, W.A. and Mahon B. 2000. Peer Networking. *Deutsche Banc Alex. Brown White Paper.*

Rosenberg, W., Kenney, D., and Fisher, G. 1992. Understanding DCE. O'Reilly & Associates, Inc.

Roman, G., Huang, Q., Hazemi, A. 2001. Consistent Group Membership in Ad Hoc Networks. Proceedings of ICSE, Toronto, Canada.

Rowstron, A. and Druschel, P. 2001. Storage Management and Caching in PAST, a Large-Scale, Persistent, Peer-to-Peer Storage Utility. *Proceedings of SOSP*, pp 188-201.

Saltzer, J.H., Reed, D.P., Clark, D.D. 1984. End-To-End Arguments in System Design, ACM TOCS, Vol 2, Number 4, November 1984, pp 277-288.

Sandberg, R., D. Goldberg, S. Kleiman, D. Walsh, B. Lyon. 1985. Design and Implementation of the Sun Network Filesystem. *USENIX Conference Proceedings*, USENIX Association, Berkeley, CA, Summer 1985.

Saroiu, S., Gummadi, P., and Gribble, S. 2002. A Measurement Study of Peer-to-Peer File Sharing Systems. MMCN, San Jose, CA, USA January 2002.

Satyanarayanan, M., Kistler, J.J., Kumar, P., Okasaki, M.E., Siegel, E.H., and Steere, D.C. 1990. Coda: A Highly Available File System for a Distributed Workstation Environment. IEEE Trans. on Computers, pp 447-459,vol 39 1990.

Scarlata, V., Levine, B. N., Shields, C. 2001. Responder Anonymity and Anonymous Peer-to-Peer File Sharing. ICNP'01.

Schwartz, E. 2001. 'Parasitic Grid' Wireless Movement May Threaten Telecom Profits. August 24, 2001. www.infoworld.com/articles/hn/xml/01/08/24/010824hnfreewireless.xml?0827mnam.

SHA-1. 1997. American National Standards Institute, American National Standard X9.30.2-1997: Public Key Cryptography for the Financial Services Industry - Part 2: The Secure Hash Algorithm (SHA-1).

Shamir, A. 1979. How to share a secret. Communications of the ACM, vol. 22, n. 11, pp. 612-613, Nov. 1979.

Shields, C., Levine, B. N. 2000. A protocol for Anonymous Communication Over the Internet. 7th ACM Conference on Computer and Communication Security, Nov. 2000.

Shirky, C. 2001. What is P2P... and what Isn't. *An article published on O'Reilly Network.* www.openp2p.com/lpt/a//p2p/2000/11/24/shirky1-whatisp2p.html.

SearchLing. 2000. SearchLing Peer-to-Peer Networking. White Paper. October 2000.

SETI@home 2001. setiathome.ssl.berkeley.edu.

Stoica, I., Morris, R., Karger, D., Kaashoek, F., and Balakrishnan, H. 2001. Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications. *Proc. SIGCOMM, pp 149-160.*

Strom, D. 2001. Businesses Embrace Instant Messaging. January 2001 enterprise.cnet.com/enterprise/0-9534-7-4403317.html.

Sullivan, W. T. III, Wertheimer, D., Bowyer, S., Cobb, J., Gedye, D., Anderson, D. 1997. A new major SETI project based on Project Serendip data and 100,000 personal computers. Astronomical and Biochemical Origins and the Search for Life in the Universe. Proc. of the Fifth Intl. Conf. on Bioastronomy, Colloq. No. 161, eds. C.B. Cosmovici, S. Bowyer, and D. Wortheimer (Publisher: Editrice Compositori, Bologna, Italy.

Sun Microsystems. 2002. Security and Project JXTA. Sun Microsystems Technical Report.

Suthar, P. and Ozzie, J. 2000. The Groove Platform Architecture. *Groove Networks Presentation.* devzone.groove.net/library/Presentations/GrooveApplicationArchitecture.ppt.

Syverson, P. F., Goldschlag, D. M., Reed M. G. 1997. Anony-

mous Connections and Onion Routing, 1997 IEEE Symposium on Security and Privacy. pp. 44-53.

Tanenbaum, A. S. 1981. Computer Networks, Prentice-Hall International.

ToadNode 2001. www.toadnode.com.

Tuvell, W. 2002. Personal Communication.

Veytsel, A. 2001. There is no P-to-P Market... But There is a Market for P-to-P. *Aberdeen Group Presentation* at the P2PWG, May 2001.

XDegrees 2001. www.xdegrees.com.

Waldman, M., Rubin, A. and Cranor, L 2000. Publius: A Robust, Tamper-Evident, Censorship-Resistant Web Publishing System. Proceedings of the USENIX Security Symposium. Denver, Colorado, USA. Aug, 2000.

Waldo, J., Wyant, G., Wollrath, A., Kendall, S. 1997. "A Note on Distributed Computing", Mobile Object Systems, Lecture Notes in Computer Science, No. 1222, pp. 49-64, Springer-Verlag, Berlin (D).

Waterhouse, S., Doolin, D.M., Kan G., Faybishenko, Y. 2002. Distributed Search in P2P Networks. IEEE Internet Computing 6(1):68-72. January-February.

Wollrath, A., et al. 1996. A Distributed Object Model for the Java System. *Proceedings of the USENIX 1996 Conf. on Object-Oriented Technologies (COOTS)*, pp. 219-231.

Xu, Z., Miller, B., and Reps, T., 2000. Safety Checking of Machine Code. *Proceedings of the SIGPLAN Conference on Programming Language Design and Implementation*, pp 70-82. Vancouver B.C., Canada. June 2000.

Xu, Z., Reps, T., and Miller, B., 2001. Typestate Checking of Machine Code. *Proceedings of the 10th European Symposium on Programming*, pp 335-351. Genova, Italy, April 2001. Lecture Notes in Computer Science 2028, G. Goos, J. Hartmanis and J. van Leeuwen (Eds.)

Yahoo! 2001. www.yahoo.com.

Yang, B and Garcia-Molina, H. (2001) Comparing Hybrid Peer-to-Peer Systems. The VLDB Journal, pp 561-570, Sept. 2001.

Yeager, B., 2002 Personal communication.

Zhao, B., Kubiatowicz, J., Joseph, A. 2001. Tapestry: An Infrastructure for Fault-Tolerant Wide-area Location and Routing. Computer Science Division, University of California, Berkeley Tech. Report no UCB/CSD-01-1141, April 2001.

Zhou, S., Zheng, X., Wang, J., and Delisle, P. 1994. Utopia: A Load Sharing Facility for Large, Heterogeneous Distributed Computer Systems. *Software-Practice and Experience*, 23(2):1305-1336, December.