

Investigating Peer-to-peer Systems for Resource Sharing within a Small Group of Nodes

L. Lanae Neild
Clemson University
lanae@cs.clemson.edu

Roy P. Pargas
Clemson University
pargas@cs.clemson.edu

Abstract

We present a study of existing peer-to-peer systems and suggest a preliminary solution for implementing a file-sharing application for use by small trusted groups of users, for example 50-200 nodes. We review existing literature on peer-to-peer systems, and application examples of peer-to-peer architectures. We describe two models. One estimates the expected number of query results in a hybrid data-sharing peer-to-peer system, the other determines file replication needs for ensuring file availability for small numbers of users. We use information obtained from these models to design a file sharing system for small numbers of users.

1. Introduction

This paper discusses two related topics: (a) existing hybrid peer-to-peer systems, and (b) models to represent query behavior and resource availability in hybrid peer-to-peer systems. Our goal is to design an application allowing users to create their own private peer-to-peer network for sharing files. For example, such a system would be useful to a group of geographically separate researchers who want to share documents, to family members sharing pictures and home movies, to teachers within a school district or state sharing tests and lesson plans, or to a group of business associates.

We first briefly discuss basic terminology of peer-to-peer systems, specifically, file-sharing systems. We review existing literature on peer-to-peer systems and note their architectural features. We list trade-offs in various services and consider application examples of these architectures. We apply a model for estimating the expected number of query results in a hybrid data-sharing peer-to-peer system to obtain preliminary query results data in a small, unchained architecture system with 50-200 nodes.

We discuss a second model for determining file replication needs for ensuring file availability in the face of small numbers of active nodes. Finally, comparing our findings with our observations of existing applications,

we propose a preliminary design for a file-sharing application for use by small, trusted groups.

2. Applications

The hypothetical peer-to-peer system described in this paper is useful in situations where a geographically dispersed group often shares files. For example, assume a group of researchers are interested in Research Topic X. Most of these researchers know each other personally and collaboration is strong and common among the group. However, they live in various countries and cannot often meet in person. They would like to have a more efficient means of sharing documents than individually requesting and e-mailing files. With the solution proposed in this paper, users can simply place copies of documents in a shared directory on their hard drive. Simply publishing files on the web could lead to plagiarism; the group prefers to have more control over the documents and who has access to them. The proposed solution provides authentication and the ability of existing members to add and remove new members as they wish. Free web services, like Yahoo Groups, exist for moderated e-mail lists, complete with file repositories. However, the group of researchers would like to avoid the ads and unsolicited e-mail that typically come with using such a service. Lastly, the group may not have access to a web server, a database server, and specific middleware required to take advantage of any solutions that require these. They would like an application that they can simply install and configure, with minimal or no assistance from network administrators. In our solution, a server requires a little more configuration, but is installed similarly to the servent. A user with DSL or cable, and dynamic DNS or a static IP, could install the server. This server is actually a modified servent that exists to facilitate creation of an ad-hoc peer-to-peer network and key exchange; an established mesh will continue to exist even if the server goes offline.

Other examples of geographically diverse small groups include a family sharing pictures and home movies, teachers of a specific subject within a school district or

state sharing tests and lesson plans, and colleagues within a business required to share files securely.

Privacy and security are concerns for some of the above groups, this is where cryptography in. Many existing peer-to-peer applications are now including encryption of traffic over the peer network [3], [6]. We believe that the successful peer-to-peer application of the future will be designed to use encryption if it is deemed necessary by users.

Unfortunately, today's peer-to-peer file sharing applications are mainly used for unauthorized sharing of copyrighted material, such that research often entails venturing into a technology underworld [3]. However, the examples we gave show that peer-to-peer technology has many potential applications in business, education, and recreation.

3. Peer-to-peer Terminology

An important characteristic of peer-to-peer systems is equality among *nodes*. Nodes communicate directly with other nodes in the network. In some systems nodes are completely autonomous, in others a server is used for part of the implementation [12].

Peer-to-peer nodes communicate to share data or computing resources. Seti-at-Home is an Internet-connected distributed computing application that uses otherwise idle PCs to analyze radio telescope data [11]. ICQ is a popular instant messaging application that relies on a peer-to-peer system [7]. Some systems focus on the sharing of storage resources. OceanStore, for example, is a highly available and robust data storage system where each user contributes a portion of storage space [9]. Most commonly, peer-to-peer systems are used for file-sharing. Examples include Napster [10], Gnutella [4], and Kazaa [8], which support searching and exchanging files stored on users' PCs. Although file-sharing is their purpose, it may be argued that a secondary function is the sharing of storage space.

In a *pure peer-to-peer system*, there are no centralized functions. In systems like Gnutella, nodes are known as *servents*, combining the roles of both server and client. These nodes can potentially communicate with any node connected to the network; the drawback is that each search by a user can flood the network [12]. Introducing a hierarchy among Gnutella nodes solved this problem, so that higher-bandwidth "super-peers" handle most of the query traffic [4].

The Gnutella system with super-peers, while no longer maintaining true equality among nodes, differs from a *hybrid peer-to-peer system* in that it does not rely on centralized servers for functionality such as indexing, as does Napster. In a hybrid system, servers hold indexes containing searchable *metadata* of users' files, such as filename, keywords, etc. Servers also hold information

on *user connections*, such as IP address and connection speed [12]. Searches are performed at the server.

Given the distributed nature of peer-to-peer systems, nodes send *messages* to communicate over the Internet [2]. Some basic types of services implemented by messages include:

join In a pure peer-to-peer system, nodes must know of each other's existence. A new node is given a list of other nodes to which it can connect, and the new node is added to others' lists.

login In a hybrid peer-to-peer system, a client node connects to a server and uploads metadata on the files that node has, to be included in the server's searchable index.

publish Make a file available for searching and downloading by other nodes. In a hybrid system, file metadata is uploaded to the server's index.

query A query typically searches on keywords or some other criteria, and returns a list of files matching the criteria with identification of the nodes having those files.

get Download a copy of a file from a node.

4. Server Architectures for Hybrid Peer-to-peer Systems

Yang and Garcia-Molina [12] describe four distinct architectures. These differ in how *login*, *query*, and *download* services are handled. The architecture types are *chained*, *full replication*, *hash*, and *unchained*.

Chained architecture servers connect to remote servers and can answer queries from remote nodes via a "chain" of servers. When a user logs in, metadata is indexed only on the local server. If a server does not find files matching a query from one of its local nodes, it then forwards the query to a remote server, which returns any results to be sent to the user. Updating of metadata, which occurs on logins and downloads, is minimized because it only affects the local server. Queries, however, can be very expensive.

In the full replication architecture, each server maintains metadata for all nodes, increasing storage and bandwidth requirements on servers. Logins and downloads are more expensive because updates to metadata must be propagated to remote servers. The trade-off vs. chained architectures is that queries are not as expensive, because they only happen locally.

In the hash architecture, a given server holds an inverted list for a subset of metadata words. Only servers containing the inverted list for a keyword in a query, will be involved in responding to that query. The drawback here is that much bandwidth may be used in sending inverted lists between servers.

The unchained architecture is the most simple and limited architecture. The main disadvantage is that users do not get to query metadata indexes on remote servers,

however, the system scales with the number of servers. This scalability is an advantage in large systems, but is also suitable for our small groups of nodes, because one server per group is more than sufficient. We focus on unchained architectures in the remainder of this paper.

5. A Query Model

Yang and Garcia-Molina developed the following model for estimating the number of query results, and in a chained architecture, the number servers needed to process a query [12]. We use this model to estimate query results for some situations in our small groups scenario.

5.1. Probability density functions

Assuming a universe of possible queries q_1, q_2, q_3, \dots , the following functions are defined. $g(i)$ describes query popularity, specifically, the probability that a submitted query will be query q_i . $f(i)$ describes query “selection power”. Given a particular file in a user’s library, this is the probability with which it will match query q_i . For example, if $f(i)$ is 0.5, then we expect 5 out of every 10 files to match query q_i .

The authors explain that they used exponential distributions for g and f , although other appropriate distributions could be used. They use the function $g(i) = 1/\lambda_g \cdot e^{-i/\lambda_g}$. Here, λ_g is the mean, if λ_g is small popularity decreases as i increases, if it is large, popularity is more evenly distributed. Similarly, $f(i) = 1/\lambda_f \cdot e^{-i/\lambda_f}$, assuming that these two functions are correlated. This is reasonable for a music file-sharing system, because users gain files from each other meaning that the most popular files are searched for more frequently and stored more frequently.

The authors describe caveats and other possible distribution functions. In their example, the authors define r as the ratio λ_g/λ_f . As r decreases, query selection power becomes more evenly distributed over the range of query popularity.

5.2. Calculating expected query results

In most peer-to-peer software designed for data sharing, a limit on the number of query results to be returned is either built in or defined by the user. In this case the limit is $R = 100$. Knowing that a maximum of R results may be returned narrows down the amount of calculation. For a given query q_i , the probability of m results is defined as the probability of m successes in n Bernoulli trials. $M(n)$, the expected number of results returned from a collection of n files, can be calculated using Yang and Garcia-Molina’s equation:

$$M(n) = \sum_{i=0}^{\infty} g(i) \left(R - \sum_{m=0}^{R-1} \binom{n}{m} (f(i))^m (1-f(i))^{n-m} (R-m) \right)$$

Yang and Garcia-Molina had fitted parameters based on observed results for a large-scale music file-sharing system. Their parameters were: $\lambda_f = 400$, $r = 10$, and $\lambda_g = 4,000$. In validating their model, the authors calculated $M(69000) = 14.82$ with these parameters, which matched their observed results on an OpenNap system with a difference of 1.22% [12].

In order to estimate results for smaller numbers of users, fewer files, and possibly different values of λ_g and λ_f , we ran an algorithm to simulate this and calculate $M(n)$. We substituted our own parameters to obtain estimated results for queries on n files in a single-server, hybrid peer-to-peer system for small numbers of nodes. We assumed that users would each share 10-20 files, initially, for 500-4000 total files.

Number of users	Files per user	Expected Results $M(n)$
50	10	0.11
50	20	0.23
100	20	0.45
200	20	0.91

Table 1. Example estimates for query results with $R = 100$, $\lambda_g = 4000$ and $\lambda_f = 400$

While the likelihood of getting even one query result is small, given these results for $M(n)$, we note the parameters tailored for a specific peer-to-peer system, namely the chained architecture, music file-sharing network with an average of 69000 files available at a given time. Yang and Garcia-Molina explain that λ_g and λ_f are functions of typical user behavior in a particular system. For example, a negative rather than positive correlation might exist between query popularity and selection power in an “archive-driven” system, where only older information is in the files but users often search for information not yet placed in the archive.

It is likely that our small group of users will have a much narrower set of interests than a large group of music file swappers, so we would need a revised model if we were to base our architecture choice only on this. We do, however, take the low query success probabilities shown in Table 1 as an indication that we must improve our users’ chance of finding desired files.

6. File Availability in Peer-to-peer Systems

We must consider another important factor not included in the above model: file availability. Above, we assumed that all 50-200 nodes were connected at one time. In reality, the fraction of all users logged into a typical file-sharing system at one time is quite small.

Yang and Garcia-Molina observed only 0.05% of the total user population was active at any given time. [12] While our theoretical users may stay online more than these users did in 2001, given the proliferation of broadband availability in recent years, we cannot assume that most nodes will be active at any one time. Without all nodes active, there is a chance that some files in the network will not be available when users search for them. File-sharing systems with large numbers of users do not suffer from lack of file availability even with only 0.05% of users online, because files are replicated across many nodes. Since we cannot guarantee these conditions, we suggest other measures to ensure that files are replicated and queries have a reasonable chance of returning results. Cooper, Bawa, Daswani, and Garcia-Molina [2] have a simple explanation of failure tolerance that can be applied to our situation. They describe a system designed to protect documents from node failures and from malicious nodes. While their failure is a node that crashes or is otherwise permanently unavailable, the concept can be used to discuss file replication when only a fraction of the member nodes are connected, as in normal Internet peer-to-peer systems. While an unavailable node may come online later, in the instance of a query it can be considered a failure for the purpose of estimating how likely it is that a particular document will be available. The number of times that a file needs to be replicated are discussed by the authors in terms of n total nodes with k failed nodes and $n - k$ nodes that are known to be online at a given time [2]. To have a document reasonably available it will need to be replicated $k + 1$ times, so that there are more copies than there are inactive nodes. For our purposes, k can be large. If our parameters should resemble those of the typical file-sharing system, we can expect that most of the time no nodes will be active given a user base of 200. We might, very optimistically, expect 10 nodes to be active out of a total of 200 for the sake of example. We would need a document replicated on 191 nodes, otherwise it might not be retrievable. Considering that our group might be as small as 50, we would have 2.5 nodes active on average and in that case, we might as well fully replicate all files on each node, as is done in the shared “spaces” of Groove [6], a commercial application designed for secure online collaboration. However, we propose a different solution for keeping files available. Our tentative design includes ideas that we have observed in existing peer-to-peer systems.

7. Designing a Hybrid Peer-to-peer System for Small Groups

We desire the flexibility of an ad-hoc, pure peer-to-peer network like Gnutella, but we are limited by our need to make sure that at least one node remains online.

This indicates the need for a server. However, we also are *not* limited by the scalability problems of Gnutella: our nodes can broadcast without drastically flooding our small network. Therefore we use as a basis for our design the Gnutella protocol [5]. We add new message types to provide for a server node which behaves like a servent with the additional feature of keeping an index of users for authentication, and keeping in its own shared directory a cache of the rarest files that it finds shared on the network. All servents, and the server, broadcast some types of messages and route others.

7.1 Rarest First

Our server shares files just as the servent, so we make use of this to store files that will most likely be unavailable to servents. When a node makes a new file available, it is the only node to have that file, so it will send a message to the server, or if the server is unavailable, to another servent. The server will periodically download and make available all of the rarest files until a user-defined amount of storage space is full. Then it will replace the most common files it has with less common ones as new files are added.

Rarest-first applies to pieces of files in BitTorrent, a swarming download application [1]. BitTorrent involves an ad-hoc, partial mesh of nodes all trying to download the same file at once, where downloader clients get pieces of the file from other downloaders. Clients check to see which pieces of the file they can download from other clients having the partial or entire file, and then download the rarest pieces first to help these stay distributed in the network.

7.2 A Mesh to Mitigate Firewall Limitations

We refer to BitTorrent [1] and WASTE for an example of how peer-to-peer systems can deal with nodes behind firewalls. Since two nodes behind different firewalls cannot have a direct connection [5], firewalled nodes must typically find desired files shared on non-firewalled nodes to download them. A BitTorrent node behind a firewall must depend on downloading file pieces from non-firewalled nodes, if a piece it needs is behind a different firewall it has to wait until a non-firewalled node downloads that piece. Firewalled nodes suffer worse performance in such a system, but they are at least able to download because the mesh structure of the network has possibly many redundant paths. Non-firewalled nodes carry the burden of traffic for firewalled nodes, which is not ideal but has not harmed the functionality of BitTorrent or WASTE. Connections must be routed through non-firewalled nodes to connect two firewalled nodes. Download request and response messages can be routed through non-firewalled servents and the server.

This will require that file downloads happen directly through our application's protocol, rather than through HTTP as in Gnutella [5], and also that servers use an algorithm to select optimal paths for these downloads. Since our system does not need to scale beyond 200 users, most of which will likely not be online concurrently, our file transferring traffic will be light, hopefully not placing an unreasonable burden on the "routing" nodes.

8. Conclusions and Future Work

We reviewed existing literature on peer-to-peer systems for information useful in designing a peer-to-peer system for small groups of users to share files. Describing existing peer-to-peer architectures and application examples of these architectures, we noted the trade-offs in login, download, and query services. We described a model for estimating the expected number of query results in a hybrid data-sharing peer-to-peer system, and used the model to obtain preliminary data showing possible query results in a small, unchained architecture system with 50-200 nodes. We discussed another model for determining file replication needs for ensuring file availability in the face of small numbers of active nodes. Comparing our findings with our study of existing applications, we propose a preliminary solution for a file-sharing application used by small, trusted groups.

Findings based on our implementation of the query model we obtained through our review of the literature are limited, because the distribution functions are determined by researchers' expectations of user behavior in a particular system. This is less of a limitation when studying existing peer-to-peer systems with known usage patterns. Yang and Garcia-Molina describe how different distribution functions can be used to model varying usage patterns with regards to query popularity and selection power. The same sort of limitation applies to the model for file availability; we must assume how many nodes are usually disconnected.

Such an application will be limited to groups of users with the ability to have a specific port on one or more of the nodes *not* blocked by a firewall. The application will not be able to scale to user groups larger than our target 50-200, for reasons of bandwidth and performance.

Our query-simulating algorithm may be a useful tool for further research, giving the ability to simulate query

results by specifying desired parameters for a peer-to-peer system. In this vein, future work could enable a more thorough understanding of query and file replication patterns in different peer-to-peer systems.

For our goal file-sharing system, our research indicates that we should focus on increasing file availability. We will design an application based on our study of query behavior and of existing peer-to-peer systems.

9. References

- [1] B. Cohen, "Incentives Build Robustness in BitTorrent", May 2003. <<http://bitconjurer.org/BitTorrent/bittorrentecon.pdf>>.
- [2] B. Cooper, M. Bawa, N. Daswani, and H. Garcia-Molina, "Protecting the PIPE from Malicious Peers" Technical Report, Stanford University, May 2002. <<http://dbpubs.stanford.edu:8090/pub/2002-27>>
- [3] Fraser, Powell, "Secret networks protect music swappers" CNN, July 2003. <<http://www.cnn.com/2003/TECH/internet/07/29/private.filesshare/>>.
- [4] Gtk-Gnutella Development Home Page, May 2003. <<http://gtk-gnutella.sourceforge.net/>>.
- [5] Gnutella Protocol Specification, May 2003. <http://www9.limewire.com/developer/gnutella_protocol_0.4.pdf>.
- [6] Groove Security Architecture: A White Paper, October 2002. <<http://www.groove.net/products/workspace/securitypdf.gtml>>.
- [7] ICQ Home Page, May 2003. <<http://www.icq.com/>>.
- [8] Kazaa Home Page, May 2003. <<http://www.kazaa.com/>>.
- [9] OceanStore Home Page, May 2003. <<http://oceanstore.cs.berkeley.edu/>>.
- [10] Open Napster Home Page, <http://www.opennap.sourceforge.net/>, May 2003.
- [11] SETI@home Home Page, May 2003. <<http://setiathome.ssl.berkeley.edu/>>.
- [12] B. Yang and H. Garcia-Molina, "Comparing Hybrid Peer-to-Peer Systems", *Proceedings of the 27th International Conference on Very Large Databases (VLDB)*, Roma, Italy, September 2001, pp. 561-570