The Future of the Decentralized Model of P2P File-Sharing

Ryan Sit Mike Semanko Raymond Kim Emir Basic

CSE 222, Winter 2000
Computer Science and Engineering Department
University of California, San Diego

1 Introduction

> *"I don't think as a concept peer-to-peer applications are inherently flawed. I think current implementations are flawed." [1]*
>
> Kevin Smilak
> Chief Technology Officer
> Scour

Currently Napster's future is very uncertain. What does this mean to us? If Napster does in fact lose its case, this will prove that a centralized model of peer-to-peer (P2P) file-sharing will not work for the uncontrolled distribution of data. The use of this type of network may be controversial since it is popularly used to share copyrighted or illegal materials, but the sharing of these materials will nonetheless persist.

The only alternative for the free sharing of data is a decentralized model of P2P file-sharing. The current most popular implementation of this is Gnutella, but its present implementation is flawed. This is because of the inherently hard problem of creating a decentralized P2P system that can reach a similar performance to Napster's system. In particular, the scalability of this type of network is very low because of its exponential traffic growth. At this time a great deal of work is being put into optimizations that create unequal nodes where nodes with more bandwidth carry more traffic. But the outcome of these optimizations are questionable and to this day the main problems of Gnutella still persist: low responsiveness, download failures, slow search and download speed, instabilities, and low search results. The problem becomes even easier to see when you realize that file-sharing networks only become useful after reaching a critical mass where there is a high probability that someone on the network has what you want. With Gnutella's support for any type of file, not just MP3s, Gnutella's needed critical mass is much larger. This becomes a problem since its performance greatly decreases as the users increase. Since Gnutella's user size should be multiple times larger than Napster's to reach critical mass, the future of Gnutella remains in doubt.

If Napster and Napster type systems can't bypass outside intervention and Gnutella ultimately doesn't overcome its technical threats that come from mass usage, what system can? This paper will describe an alternative named OurNet where decentralized P2P file-sharing is added to chat groups to help provide categorizing and an implicit hierarchy. This combination will ultimately provide a scenario for significant optimizations on the decentralized model of P2P file-sharing. In section 2 we give an analysis of current popular distributed P2P file sharing programs and speculate on the future of file-sharing P2P if Napster (centralized P2P file-sharing) fails. The description of our new P2P file sharing protocol, OurNet, is presented in section 3. In section 4 we will discuss the following additional topics: analysis of scalability, possible implementation paths, avoidance of legal issues and shutdown, a possible business model, and related work on P2P groups concerning multicast and security. Section 5 concludes with a summary and the possible future impact of a successful, uncontrollable, and usable P2P file-sharing network.

2 Background

Let us first begin with a general overview of the two currently most popular P2P file-sharing schemes, Napster as an example of an centralized system, and Gnutella as a decentralized system. The protocols will be described, the run-time anaylized, the fatal weaknesses will be given, and their current state will be discussed.

2.1 The Napster Protocol

Napster's file sharing protocol is based around a central index server that contains the files being shared by Napster clients. In order to find a file, a client accesses this central index, finds the file they desire, and then locates the machine on which this file resides. Once the machine that has the file is found, the client initiates a connection with the machine and downloads the file directly from there.

The central index server provides a number of useful features in the Napster system. First, it allows easy and quick searches of data that resides on the network. Second, it reduces the amount of network bandwidth taken up by the algorithm because search requests do not need to be forwarded amongst clients. Finally, the central server makes it easy to connect to the Napster network. Because there is a single known address for the central server, this connection can be done automatically by installation software.

Unfortunately, the centralization of the Napster index causes problems as well. Because the algorithm requires this index server in order for clients to initiate connections, the Napster service is easy to shut down. The central index server becomes a point of failure, and its removal causes all of the Napster clients to cease communication. The ease of shutdown is not really a problem with the protocol, but more a problem of avoiding third party intervention because of legality or other issues.

In order to make an analysis of the Napster protocol, we must make a few assumptions. Assume that the average latency of the network is A, and that the average bandwidth is B. Assume also that a machine operation takes time M. Then, if we assume that the Napster index server is using a hash table, that the time for a song request is going to be in $O(A + 1/B + M)$.

Currently the Recording Industry Association of America (RIAA) is suing Napster Inc. for massive copyright infringement. The final outlook does not favor Napster and recently Napster has voluntarily turned on a filtering system meant to block copyrighted songs from being traded. In addition, other centralized P2P file-sharing systems are being attacked or have been attacked by the RIAA including OpenNap servers and the successful, but currently shutdown, Scour Exchange. Clearly the outlook for centralized P2P systems does not seem positive, but new work is being done to avoid these recent setbacks. This includes the scrambling of filenames to bypass filters and setting up OpenNap servers in places where U.S. copyright laws cannot be enforced.

2.2 The Gnutella Protocol

Gnutella, unlike Napster, has no servers. Thus, all of the machines, refered to as servents (a mixture between client and server), are treated in the same way. The lack of a server slows down the Gnutella protocol significantly as searching becomes much more difficult. However, this protocol benefits from the fact that it is very difficult to shut down the network, since there are no servers. Also, this network can be used to share any type of file (which makes it more difficult for someone to say that this program was designed to promote the infringing of copyrights).

Once a Gnutella servent is given the address of one other servent, it connects through that servent to a number of other servents in the network. Thus, the beginning servent suddenly gains a number of methods to access the network, and this makes Gnutella fairly robust in the face of servent failure. Querying is less efficient, however, as each query gets flooded through the Gnutella network, each individual computer must search its data for this song. Hits are returned to the requesting computer, and a download is then set up outside the Gnutella network through a normal connection.

Performing an analysis of Gnutella requires even more assumptions to be made. Taking the same variables as in the Napster case, we also assume that there are n Gnutella servents in the network, and that each servent is connected to k other servents. We assume as well that a time-to-live (TTL) of t is put upon any song requests. Then a Gnutella song query requires time in $O(min((A + k/B + M) * t, (A + M) * log_k n + n/B))$. Note that the time to find a song is not an exponential algorithm, as is commonly believed. Much of the exponential features of this algorithm are hidden by the fact that the operations can all be done in parallel. The network data transmitted, however is exponential in growth, and namely is in $O(min(k^t, n))$ bits. For example, if you send out a request with a time-to-live of 7 and each peer contacts an average of 5 other peers, over $((k^{t+1}-1)/(k-1))-1$ or about 100,000 messages could be sent out by one query. One problem with the Gnutella algorithm, however, is that the latency and bandwidth are not so evenly distributed as assumed above, and one slow link can cause the query results to take a much longer time than expected.

The fundamental algorithm of a decentralized P2P system is slow. This is because queries need to be broadcasted to all hosts within view. Optimizations can be made, but the exponential traffic growth cannot be overcome without breaking the concepts of a single decentralized P2P file-sharing network like Gnutella. As Napster's free sharing of music nears an end, more interest is being directed towards Gnutella. Many new optimizations [2] have been made to the Gnutella network. For example, applications like BearShare [3] and LimeWire [4] have started enforcing connection-preferencing rules that help peers with similar bandwidth connect to each other. In addition, Clip2.com, Inc. [5] has released Reflector 1.3 Beta, an application that makes high-bandwidth peers act as a proxy and index server for low-bandwidth peers. Even with these recent developments, the currently small Gnutella network has still been said to "exist in an intermediate state between scaling and collapsing". Now imagine if Napster's estimated 5 million unique users weren't able to download their MP3s anymore and decided to finally try Gnutella. The outcome does not look good.

3 Algorithm & Specification

In this section, we will present our alternative protocol, OurNet, which adds the decentralized model of P2P file sharing to chat groups to improve upon Gnutella's performance. In Gnutella, a servent searches for files by broadcasting a search request to all of the other servents it knows about. This technique is very inefficient. Our protocol provides flexible mechanisms that allow an implicit hierarchy and categorizing, which will allow us to improve the efficiency of the network and the relevance of search results. The improvements are possible because of two main properties of our new system. First, instead of the *entire* Gnutella system scaling with the number of users in Gnutella, our system scales with the number of users in any particular OurNet chat channel. Second, instead of searching a large random part of the network, our system gives you the option of searching a subset of the network with relevance to the query, therefore giving a query result with a high precision (high probability of relevance).

In this section we will describe the two main components of our system. The OurNet channel component facilitates practically sized network groups with similar interests to give us an efficient network. The OurNet Group Manager component helps with the searching and publishing of channels, and the categorization used for the searching of files.

3.1 OurNet Chat Channels

The main idea is to just add decentralized file-sharing to chat groups. As it turns out, this simple idea changes the scenario significantly enough to gain notable improvements on the efficiency of traditional decentralized P2P file sharing systems.

This combination of file-sharing and chat groups is done by having separate mutually exclusive file-sharing groups. This means that instead of sending messages to everyone possible, messages are directed and limited to members of a particular group. To illustrate, in Gnutella everyone is connected as one big network. Queries are sent to everyone reachable with a given packet TTL. If one group of say three connected computers wanted to just query each other, they would have to make sure that none of three computers connected outside of their group. If just one of the computers were connected to the main Gnutella network, their exclusive group would be broken. This means the group's queries would now be relayed to the main Gnutella network and the Gnutella network could query them. Our model is decentralized like Gnutella, but has independent chat channels where users with similar interests can meet and share files. Independent meaning that each group is like their own small Gnutella network. To join the group, a peer must connect directly with one of the active members of the group. In other words, the idea of one large network is not part of our system. Channels should not be maintained by anyone not in the group.

The reason for the functionality of chatting is to help with the management and organization of the channel, and to create a feeling of a community. Chatting would be implemented using the existing decentralized P2P file-sharing channel network. This is unlike the current popular implementation with a central server. Ideally each chat channel would have similar features to the Internet Relay Chat (IRC) type chatting system. That way valuable functionality such as kicking, banning, setting topics, having different channel modes, and size limits could be enforced. This functionality would make the administration of the channel possible to ensure quality members and subside chaos. Administrator privileges would ideally be given to channel operators (ops) who are possibly elected by positive interaction in the chat channel. As an additional by product of this scenario, a hierarchy is formed by the use of ops. Other optimizations could

be possible by this feature such as requiring ops have act as "super peers" that carry more traffic, like a tradeoff of power for bandwidth.

One last point to be made on the subject of chat channels is about the likely self-management of channel size. This is because of the natural tendency for chat groups to split or branch when the group size grows too large. Likewise in our scheme chat channels could be pressured to split because of too many people talking at once or if there is too much traffic for one channel to handle. Other reasons could also be possible for a channel split.

The other component of our scheme is the group manager. The group manager functions as a directory to find chat channel and as a place to publish chat channels to gain members. In addition, for a centralized implementation of the group manager we can use the group manager for many other tasks considering the information it knows.

It is quite simple to add a chat channel capability to the Gnutella Algorithm to implement OurNet. Just add the channel name you wish to search to any queries, hits, and pings while a particular machine resides within a channel. However, there are complicated issues that arise when you follow this simple addition to the protocol. The first consideration is that the OurNet program must be able to accept queries over multiple channels if it stores files on it that reside in multiple channels. This is, in fact, a very likely situation. A second consideration is that OurNet machines must ignore channel messages for a channel that they are not in. Otherwise, if they must deal with these messages, the network will face the same scaling issues as normal Gnutella.

The biggest problem with this technique, however, is identifying which machine to connect to initially for any given channel and identifying which channels are currently available. It is no longer sufficient to have a one simple host list to connect to initially because in our scheme a peer must directly connect to the chat channel. This issue is dealt with in the next section by providing a channel directory that gives these initial hosts.

3.2 OurNet Group Manager

It is not useful to categorize Gnutella into a number of channels without some method to find out what channels are available to search. We thus must provide a directory for the currently available channels. There are a number of possible implementations for this directory, and they all have important implications for our system. We provide two possible implementations below.

3.3.1 Fixed Channel Directory Machines

The simplest method would be to provide a single or multiple machines that act as directory servers. Whenever someone wanted to find a particular channel, they would access one of these servers to search for the channel name, and then use that channel name in their Gnutella packets to only speak with machines in that channel.

This method has a number of advantages. Fixed directory machines are time and bandwidth efficient. They are intuitive, and easy to implement. Finally, there could be money to be made in the owning of these directory servers.

Unfortunately, this method goes against grain of the Gnutella movement. The purpose of Gnutella was to distribute the information over multiple machines to make the network difficult to shutdown. If there were a single or multiple fixed directory servers, someone interested in seeing Gnutella shutdown would shutdown these directories first. Once the directories are gone, the channels could no longer be found. The Gnutella network would go back to being an overcrowded mess.

3.3.2 A Distributed Channel Directory over Gnutella

With the knowledge that fixed machines are probably not a viable solution, we should consider distributing the directory somehow. In this section, we present an interesting approach for doing this, using the Gnutella network itself. We assume that the Gnutella network is divided into channels and the channel names are strings.

We create a special Gnutella channel named *#channels*. In this group, the filenames of these files are channel names that the servent knows about, and the contents of the file are the addresses of a few servents within that channel. We refer to these special files as channel files. So, if a user queried the *#channel* group for everything, she would receive all of the channel names currently being used. By selecting one of those channels, she receives the addresses of which servers to contact in that channel via

the channel file, and now she can query that channel directly. Selecting the file also causes the channel file to be downloaded to the users computer, where it becomes available for other users to access.

Once a user's machine has the channel file, the user may now search the specified Gnutella sub-network for that channel without accessing the directory server again. The channel file which now resides on the user's computer offers access. If all of the systems within the channel file prove unreachable, then the Gnutella system should query for that channel in the #channel group in an attempt to find more servents to contact.

When a file is shared, a user may select a channel to place the file in. Sharing this file in that channel causes the system to add a channel file (or append to the current one) for that particular channel. Thus, channel queries may now turn up this particular computer as a resource. At any time, the Gnutella system keeps track of which channels are on the current system. If any file queries are received from any of these channels, the system will respond as if it was a member of that channel. Thus, a Gnutella system now must act as a member of multiple groups if there are files residing in multiple channels. Files that are not part of the respective machine's channel are ignored (they are not even forwarded along the Gnutella network).

Channels can be placed within channels to form a hierarchy of categories. Because of this, channel files require a file designator to distinguish them from music or other types of downloads. A file extension will work for this. All channels are simply labeled ????????.chl.

The advantage to this system is that now, in order to shut down the Gnutella directory service, one must shut down Gnutella itself. Assuming Gnutella had a large acceptance, this would be infeasible. The disadvantage is that this dispersion of channels may cause user searches to take twice as long and may waste a lot of network bandwidth with duplicated channel files being sent to various computers. If the channel group turns out to be as large as the music group already in existence, we know that this won't scale very well. However, there is the benefit that if a user tends to search in a particular channel, they will be able to search more quickly and without having to access a directory. Thus, caching methods can be attempted here to speed up the user's access time when they actually use the program.

4 Additional Topics

In this section we will discuss additional topics related to our new OurNet. First, the scalability of OurNet will be shown to be much greater than other decentralized P2P file-sharing programs because of optimizations using the inherent properties of OurNet. Next, possible directions in implementation will be proposed using the available open source code of Gnutella and IRC. Then issues relating to legality will be brought up. This will mainly focus on the avoidance of legal problems and shutdown, and protecting users from legal responsibilities. Also a business model exploiting unique features of our system will be proposed. This model will be based on intelligent advertisement targeting. Lastly, papers on security and multicast relating to P2P groups will be presented. The related work in P2P groups will be shown to be a plausible and useful addition to OurNet.

4.1 Scalability and Performance

We know there are many reasons why Napster (central index server) has the potential to scale quickly, but because of the many complicated legal issues, its pace in the race has come closer to a slow-down. For argument sake, we shall label Napster as the 1st Generation P2P system. Next comes along Gnutella and its problems with scalability. We will argue how ours scales better than Gnutella and why it has a high probability of success.

Because of Napster's near-death trend, Gnutella technology is becoming more predominantly researched as the next possible standard. But there still exists many issues which hold Gnutella back from becoming widely scalable. Our transition from Gnutella to our protocol, OurNet, is one that points out two clear advantages that make OurNet highly scalable.

The Gnutella network can be challenging to manage and secure. As its size grows, limits of performance and fairness are affected. Our method requires even less effort in managing nodes (the network). Our protocol can use the idea of channels in order to categorize items. These channels represent sub-networks which hold the structure of a hierarchical tree. Each level of the tree represents a new sub-network which manages its children nodes only. Not having to worry about parent nodes or grandparent nodes makes managing traffic and congestion a lot easier and simpler. A possible limit of users per

channel end-node gives us ground to saying that this channel structure will maintain network stability and fairness, encouraging high scalability.

We have already mentioned how a decentralized P2P system, like Gnutella, is slow because queries need to be broadcast to all hosts within view. Take note that OurNet could possibly resolve such delayed queries. Again, with our plug-in of channels into a P2P system, we see how queries could take much less time by using an efficient sub-network search method. This not only avoids the huge overhead of broadcasting queries to all hosts, but also retrieves results in just a fraction of the time. A quick example would be searching for a particular mp3, say Michael Jackson's hit song "Thriller". A sample sub-network search would most likely start within the 'media' channel and work its way down to 'music'. From 'music' it finds 'mp3s' which includes '80's' as a child channel. Knowing Jackson's music is pop, it searches within the 'pop' channel and finds it way toward 'Jackson'. Quickly, the query finds 'Thriller' and notifies the sender that a hit has been made. With this example, we are assuming the user is sitting within the 'media' channel. Say the user at the time happened to be sitting within the 'pop' channel, her search query would cost that much less time in finding that particular file. So, clearly, the use of channels can significantly increase the quality of searches, proving performance scalability.

Now, we attempt to give an analysis of our algorithm, and we show why it is superior to the Gnutella algorithm. Unfortunately, there are a number of features to our algorithm that make it very difficult to analyze. First, we have that any particular servent can be in multiple channels. For the sake of analysis, we are going to assume that any OurNet servent is only in one channel at a time. The second difficulty is that the number of channels is going to be dependent upon the number of users in the network. Just as in a chat room situation, if there are more users, there will be more content and there will be more channels. The assumption we make is that there are m channels and the number of servents in each channel is b. Thus, $n = bm$. The final difficulty in analysis comes from the fact that when the Gnutella network is broken up into smaller and smaller channels, the connectivity is going to decrease. We assume that there are k connections in the n user case, and the number of connections in the group remains proportional to the size of the channel in which you reside. Thus, there are k/m connections in a particular channel. These assumptions, combined with those from the Gnutella analysis, give us a time for a song request in $O(min((A + k/mB + M) * t, (A + M) * log_k b + b/B))$. The amount of transmitted data goes down significantly, to being in $O(min((k/m)^t, b))$ bits. We feel that these two improvements are enough to allow Gnutella to scale to hundreds of thousands or even millions of users.

4.2 Related Work

In this section, we mention a few related projects that show us how OurNet can be optimized for system and user enhancements. One project in particular offers an alternative approach to OurNet involving a multicast application resulting in efficient data throughput to multiple hosts. The details of the research stretch further into end-host mulicast (EM), where hosts are connected to each other, forming a virtual network. With this structure, routes are discovered sending multicast packets to each end-host. EM proves to solve many IP Multicast problems, but also introduces new problems. One of which results in terrible scalability issues for groups larger than a few hundred users. Others involve requiring bootstraps inorder to join the groups, while some involve issues with firewalls, etc.

The name of this alternative protocol is Jungle Monkey (JM) [6]. It is a peer-to-peer file sharing program with support for channels with users of the same interest group. Each channel in JM is a multicast group, which is capable of sending out announcements of newly created channels or newly shared files. Whenever the JM search server receives an announcement of a new channel, it sends a join message creating a new node within the multicast group. Similarly, when an announcement of a new file-share is made, an addition is made to the multicast list.

Some of the technical details of this protocol are similar to other known protocols. Users, for example, must be part of a channel in order to share files. If necessary, they can also create channels in the channel directory, which is hosted by junglemonkey.net (Monkey Central). When a user wants to download a file, the host must connect to the main server and retrieve the list of hosts which hold the desired file. Then, EM makes it possible for the download to occur between the closest hosts.

JM's EM protocol is called Banana Tree Protocol (BTP), which is a layer above TCP/UDP. The basic idea with BTP is to build and manage multicast trees. The advantage with BTP is that no routing tables are required and routing within a tree is simple and loop-free. The only disadvantage with BTP would be having a single point of failure (root). Having this tree-lie structure is basically good idea, but optimizing it can be difficult.

The BTP authors argue that nodes can change (switch) the parents without creating a loop in order to optimize the tree. The concept of switching is to find the closest sibling (host) possible. To find a sibling ("potential parent") closer than its parent, the node can simply ping each sibling ("virtual someping") or use Internet distance service like IDMaps. So, when downloading the file, the host joins the tree and switches parents to find the closest hosts, which then become its parent. Finally, file is sent from parent to child.

Above this layer of BTP is the Banana Tree File Transfer Protocol (BTFTP), which provides one-to-many file transfers. There is also the Banana Tree Simple Multicast Protocol (BTSMP), which provides many-to-many group communication. The goal of BTFTP is to download popular files from the closest host possible. Channels are built on top of BTSMP.

OurNet, like other decentralized P2P applications, provides no security or privacy. [7] presents a secure P2P file sharing implemented in Java. The protocol is based around group sharing where the main goal is to allow any user to create a group and specify what files are to be shared with other members of that group (like in Unix where the owner of the file set permission to control access to the shared file). Every group contains group authorities (GA) and members. Group membership has to be obtained in order to join a group. All file transfers are done via secure channels established after peers mutually authenticate their group membership using a unique session key type encryption. The basis for security involves two protocols, "login" protocol based on the Encrypted Key Exchange and "mutual authentication" protocol, which establishes a secret session key between the peers. One advantage with this protocol is its extensibility, which makes it possible to add additional features. Turning off encryption allows peers to use other authorities than its home authority when obtaining a group membership. Furthermore, the protocol is operating system independent, meaning there is no reconfiguration needed when adding new users while also including immediate sharing of resources.

The protocol described above could be possibly built on top of the OurNet protocol in order to achieve security.

4.3 Avoidance of Legal Issues and Shutdown

As the knowledge and technology of peer-to-peer networking matures at a rapid incline, the probability of avoiding legal issues worsens. We know the main reason why P2P protocols, like Napster, face legal battles is because they simplify the act of file-sharing copyrighted material between a multitude of users. Furthermore, who will determine that Gnutella will not face the same battles involving its decentralized implementation? Our main goal in this section is not to present detailed legal issues, but to examine the different options in hopes of avoiding such issues and possible shutdown. These options will also tie into the necessity of protecting users from legal responsibilities.

Changes may need to be done in order to sidestep legal problems. One speculation starts with our implementation of group managers. These managers act as directories where users are able to find what channels exist. They help in managing sub-networks when possible and keep in memory the channel hierarchy as an optimization. Some may argue that these managers act as servers and may be items to point at when considering a forced shutdown. That is why we proposed two ways to implement the group manager. Of course the first implementation with a central powerful group manager would be the most efficient, but this implementation can be arguably illegal. Our backup plan of using our existing network is still very usable, but not as efficient and controllable. Potentially both implementations could be implemented in parallel. Therefore if the central group manager didn't work out, we could easily transition to the group manager using our existing network.

One other feature that we have added into our implementation is the possible existence of ops. Channel ops can do optimization such as manage chat channel peers and keep an index of files. This characteristic of ops could allow for fast query searches within channels and better quality of possible download hosts. Obviously, we would want to protect these group managers and channel ops, but ops could represent individual sources of liability. If any type of legal action is taken against our protocol, ops are easy targets. A quick solution to avoiding any legal issue would be to eliminate the use of channel ops and force peers within each channel to have similar properties.

4.4 Business Model

As we know, most of the programmers who have implemented variations in P2P systems have done it without any intentions of making money. But we propose a business model which will exploit unique features and characteristics or our system and of our users.

The first step in our proposal is based on intelligent advertisement targeting. In essence, because we already know the channel id and content, we have confident knowledge of the kind of users within them. With this knowledge we are able to spread advertisements specifically related to that channel, aimed in getting the attention of those specific users.

There is also the option of requiring subscription type fees for individual users, but the success rate of subscription based sites have continued to remain at its lowest. Studies have shown that common consumer will not go out of his way to pay for something that he could possibly get free somewhere else. Therefore, the second step in our proposal, with our channel like technology, is to implement an automated system that analyzes each user's likes and dislikes. This would keep information like which channels she tends to exist most often in, what file extensions are used most often in searching, and the frequency of file-sharing. This sort of information can in fact be very valuable to us. In the money-making aspect, the information can be shared to third party companies within the media industry for a cost in hopes of building stronger public and legal relations.

The last idea involves surveying users monthly with specific questions related to what chat channels they visit. This would at least be better than monthly fees. For example, all users within the PC games chat group could be asked what their favorite games are or what their favorite magazines are. This could obviously give third party companies an edge on what they think is hot for the current market. More general type questions could also be asked, like what their hobbies and interests are. Again, we could use these personal surveys to help build a database consisting of a collection of music interests, software title interests, TV interests, movie interests, computer game interests, and more. Partner companies will know the value of this type of data after they realize the number of hits for each particular interest group.

In any case, similarly to Napster, we could have retail links for areas like music or movies. For example, within the channels involving music files like mp3's or wav's, we would have advertisement buttons of some music retail company like CDNOW. Within the movies channel involving MPEG files, we could have a movie retail company link like Amazon.com or Buy.com. In essence, holding categorical items within separate channels allows us to focus in on a particular business area, leaving us multiple options for incorporating third party companies.

5 Conclusion

In this paper, we have presented a new approach for decentralized peer-to-peer file sharing. Using a categorized Gnutella algorithm, we show how it is possible to not only scale more efficiently in terms of the amount of time required in the search and the amount of traffic generated. Our algorithm also provides a more accurate search to the user than was previously possible with the standard Gnutella implementation. We discuss additional issues such as the legal issues behind the running of this service and the possible business models that our algorithm enables.

Although many people are in support of a decentralized peer-to-peer file sharing algorithm, it is important to consider what the possible impact of such an algorithm would be. What are the consequences of providing free music, books, movies, and data to whoever might be interested? While musicians have additional avenues for making money (concerts, radio royalties, etc.), not all those in entertainment are so fortunate. How can the publishing industry and authors survive when their books are released free of charge on the Internet? How can the movie industry survive, if you are able to watch any movie in the comfort of your own home?

Besides the question of copyright issues, distributed file sharing raises a number of questions about criminal activities. If we cannot shut down the network, and (as is true in some systems) we cannot determine who is downloading or storing what information, then distributed file sharing is the perfect medium for dissemination of terrorist information and child pornography. What happens when a credit card database is posted to services like Freenet or Gnutella? Should all information really be freely available to everyone without exception?

The correct answer to these questions is of course, that it doesn't matter. Technology is advancing, and there is very little that we can do to stop it short of banning peer-to-peer file sharing altogether. Whether or not this is the way things should be, it seems that this is the way things are going to be. We must be prepared to deal with the consequences.

References

[1] Borland, J. “Democracy’s Traffic Jams.” CNET news, Oct 26, 2000.
http://news.cnet.com/news/0-1005-201-3248711-2.html

[2] McCannell, S. “The Second Coming of Gnutella.” Webreview.com.
http://www.webreview.com/mmedia/2001/03_02_01.shtml

drscholl@users.sourceforge.net. “Napster Messages.” March 12, 2000.
http://opennap.sourceforge.net/napster.txt

“Gnutella: To the Bandwidth Barrier and Beyond.” Clip2.com.
http://www.clip2.com/gnutella.html

“The Gnutella Protocol Specification v0.4.” Clip2.com.
http://dss.clip2.com.

[3] Bearshare. http://www.bearshare.com/

[4] Limewire. http://www.limewire.com/

[5] Clip2 Inc. http://www.clip2.com/

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

[6] Jungle Monkey (JM) homepage. http://junglemonkey.net/

[7] Matthew R. Delco Mihut D.Ionescu, "Secure Peer-to-Peer File Sharing within Dynamic Groups"