Cost Efficient Peer-to-Peer File Sharing

Ke (Colin) Zheng Keith Grochow

{kzheng, keithg}@cs.washington.edu
Dept. of Computer Science and Engineering
University of Washington

Abstract

Self-managing peer-to-peer structures have dramatically changed the nature of Internet traffic. Currently a large percentage of network bandwidth in many domains is used for peer-to-peer file sharing. The problem with current peer-to-peer clients is that even though different routes for downloading content can have vastly different costs based on locality of the data and domain peering relationships the clients provide no means to choose and/or enforce cost efficient downloading. This work focuses on Gnutella peer-to-peer file sharing system for its popularity as well as information availability. We give a better understanding of the Gnutella world by monitoring and probing the current file distribution and topology to determine the relative value of locality. Based on this understanding, we propose a set of possible solutions to take advantage of locality in the Gnutella system. Finally we developed a prototype system to implement these solutions and assess their value in an existing network domain.

1 Introduction

In recent years, there has been unprecedented growth in the use of peer-to-peer file sharing systems including Kazaa, Napster, Gnutella and others. These applications allow users to search for specific content among the collections of other users of the system and then share these files with each other. Once a user finds other users that have the content they want, they then choose one or more locations to download the file from. They are expected in turn to make their content available to other users for upload. Finding users to share with is usually through some limited broadcast mechanism. The directory mechanism for these files is usually distributed to provide both a scalable architecture as well as anonymity for the users.

Unfortunately this popularity for sharing such content as music and videos also creates large bandwidth demands on the host networks. A recent study at University of Washington [10] showed that traffic from these systems now eclipses that of the World Wide Web. This increase in traffic has placed an extraordinary load on network infrastructure and in some cases has encouraged network administrators to limit the available bandwidth [3] to combat the file transfer demands. The bandwidth consumed not only causes overloads and poor system performance internally, but huge transmissions in and out of the domain. Bandwidth consumption over the peering and transit links greatly increase the cost of system operation, since transit links are expensive network resources.

If one wants to limit these costs, one simple way is to take advantage of shared files in the local domain whenever possible instead of transferring data over the links out of the domain. Most current peer-to-peer file sharing system have two properties that are drawbacks to this end. First, the decentralized nature of the systems creates randomness in the network topology by design to provide broad distribution and anonymity. Secondly, the only metric for choosing which host to download from is often just the speed of the link. In a cost-efficient system on the other hand, we want to provide a bias for choosing files internally first and then only then using expensive transit links.

Therefore the goal for our research is to determine cost-efficient peer-to-peer file sharing strategies by taking advantage of locality, while maintaining the existing performance and qualities of the system. Our first step was to evaluate the possible limitations and benefits of locality by analyzing the Gnutella peer-to-peer file sharing system. By collecting, monitoring, and probing a large number of Gnutella nodes we were able to determine file distribution and topology of the existing system. This data was then examined to with regard to speed, cost, and availability so that we could determine and evaluate alternative topologies that take advantage of locality.

Based on the data gained from our analysis of the Gnutella network, we propose a host-based solution to improve cost-effectiveness of the system. First, we consider how peer-to-peer clients can be updated to take advantage of locality in finding and selecting nodes to share files with. Then we use the acquired local nodes intelligently to maximize locality without sacrificing search results. Finally for files not found in the local domain, each host determines the relative cost-efficiency of file transfers based on domain specific shared costing information. Based on these solutions we implemented a Gnutella client that is locality aware and analyzed its performance relative to an unmodified client.

This remainder of the paper is organized as follows. Section 2 discusses the related work in the field and presents an overview of peer-to-peer file sharing systems with emphasis on the Gnutella system. In Section 3, we present our analysis of the existing Gnutella system and the possible benefits and limitations of locality. Section 4 describes the solutions we propose that would leverage locality in a peer-to-peer systems and also presents the results from an implementation of these solutions. Section 5 provides a discussion of issues for further investigation and Section 6 concludes and summarizes our study.

2 Related Work

The Internet has experienced an astronomical increase in the use of specialized content delivery systems, including the client/server oriented World Wide Web, content delivery networks, and peer-to-peer file sharing systems. For the last several years based on number of users and network traffic, the World Wide Web has been the most popular of these systems. Based on users the web is in no danger of being overtaken anytime soon, but as a recent study by Saroiu et.al. [10] at the University of Washington showed, peer-to-peer traffic has now overwhelmed web traffic as a leading consumer of Internet bandwidth. This phenomenon is just now getting significant attention.

2.1 Peer-to-peer Overview

The last few years has witnessed the emergence of a number of peer-to-peer(P2P) distributed systems, i.e., systems in which all nodes have identical responsibilities and all communication is symmetric. Successful application of these systems include content sharing networks such as Gnuetella [4] and large-scale storage systems such as Freenet [2]. User interact with these systems in two primary ways: they attempt to locate content of interest by issuing search queries, and once relevant objects have been located, users issue download request for the content.

While the download mechanism is predominantly by direct file transfer from the sharing machine, P2P content sharing systems have differed widely in how they provided search capabilities to clients. Napster [7], which was introduced in mid-1999, maintains a central server storing the index of all files available within the Napster user community. In order to find content, a user queries the central server using key words present in the content's filename. The server returns a list of filename matches to the user's query and the IP address of a user machine storing the file. The file is then downloaded directly from the sharing user's machine. Thus, although Napster uses a P2P communication model for file transfer, the process of locating the file is centralized.

The initial Gnutella system goes one step further in that it de-centralizes the file location process as well. In this system each node can communicate with a predetermined set of neighbors in a store-and-forward fashion. A detailed measurement study comparing the characteristics of these two P2P file sharing systems is presented in [11] taking into account the characteristics of the participating peers.

More recent P2P systems, including KaZaA [6], which uses a hybrid architecture in which some peers are elected as "super-nodes". These super-nodes index content available at nodes in a nearby neighborhood to speed lookup and lower network messaging bandwidth. More discussion on the super-node mechanism is presented below. As far as indexing, it's unknown exactly what scheme KaZaA uses to index its content, but there are several mechanisms in the literature such as the hashing scheme by Ratnasamy et. al.[9] and Chord [16]which associates a file key with location to speed search.

These systems also address other features of specific importance to their users. Freenet additionally protects the anonymity of both authors and readers, and KaZaA empowers super-nodes to be able to broker search requests of weaker clients. A description and evaluation of these and other file sharing applications can be found at [17].

2.2 The Gnutella System

The majority of our research focused on the Gnutella system due to its popularity as well as information availability. We present an overview of that system below.

As pointed out earlier Gnutella has a de-centralized file location process. So at startup each client must locate a set of nodes to query. This list of nodes is created by establishing a connection with some small set of predefined hosts, called hostcaches, that cache the addresses of other hosts and hostcaches in the system. There are several permanent Gnutella hosts in the network whose purpose is provide this service. These hostcaches return a random small set of hostcaches and hosts

from its list. The client then queries these nodes to build its list of hosts and hostcaches further. In this way each client crawls the network to acquire a random selection of host to connect to for content location. Once it has a list of hosts, the client communicates with other hosts by sending and receiving Gnutella protocol messages.

As connected nodes leave the network, a client refreshes its list to maintain some minimal set of connected hosts. If a client is closed, it stores its list of hosts and hostcaches to the disk to be used on restart as a starting point for connecting to other hosts. The very first time a client connects to the system they have no cached set of hosts and hostcaches so they connect to some preset set of hostcaches hard-coded into the client.

As mentioned earlier, originally all Gnutella nodes were connected to each other as equal peers. This design worked fine for users with broadband connections, but not for users with slow modems. This problem was solved by later organizing the network in a more structured form, the Ultrapeer system. The scheme is very similar to KaZaA system described above and categorizes the client-nodes on the network as either ultrapeer and leaf nodes. Ultrapeers are connected to each other and leaf nodes are connected to ultrapeers. Therefore any client-node keeps only a small number of connections open, and these are only to ultrapeers.

An ultrapeer acts as a proxy to the Gnutella network for the leaf nodes connected to it. This has an effect of making the Gnutella network scale, by reducing the number of nodes on the network involved in message handling and routing, as well as reducing the actual traffic among them. An ultrapeer keeps a hash table of all the content available in its leaf nodes and only forwards a query to a leaf-node if it believes to leaf-node can answer it. To provide backward compatibility ultrapeers are connected to "normal" Gnutella hosts (hosts that do not implement the Ultrapeer system) as well.

Query messages are broadcasted when the user initiates a search. Query messages are forwarded to all directly connected hosts before their TTL values expire (usually 7 hops), and the responses to query follow the exact same route as the incoming query. Once a user gets responses to the query and initiates a download, a direct connection between the source and target host is established in order to perform the data transfer.

2.3 Traffic and Costs

A detailed characterization and comparison of content delivery systems, and in particular, the peer-to-peer workloads can be found in [10]. Sen [14] did a study of P2P traffic that considered network-wide patterns by looking at the traffic from several locations on the Internet.

For understanding network domains, Spring et. al. [15] present a set strategies and tool called RocketFuel for externally determining the bounds of a network domain. They also present information on network layouts of several of the major ISP's. Norton [8] provides a thorough overview of peering links versus transit links in network domains and their importance in determining cost.

By restricting a P2P system to the bounds of a LAN, the transit link problem can be solved. Gnutella has provided such an option in the latest releases. In order to provide a corporate P2P solution that is suitable for content management, Intel [5] proposes a P2P approach to determine

the closest host containing the latest version of the requested material, the material is then copied to the requesting host.

For solutions based on network modifications, there has been some research on path selection for improved robustness and/or performance mainly built on overlay networks. Overlay networks has emerged as a powerful and highly flexible method for content delivering, such as RON [1], an architecture that allows distributed Internet applications to detect and recover from path outages and periods of degraded performance quickly by having specialized nodes monitoring path quality among themselves. It's an optimization frame based on specific routing metric. Similarly, the Detour project addressed the sub-optimality of Internet routing based on collected data [13] and showed the potential long-term benefits of detouring packets via other nodes [12].

3 Data Analysis

3.1 Approach

Our primary goal in the data collection and analysis is to determine as much information as possible about the global topology, local topology, and file distribution in a P2P system. With this data we could answer out two primary questions.

- Is it beneficial to enforce locality?
- If it is beneficial to enforce locality, what is the best way to go about it?

The first question is primarily answered with data collected on global topology. Here we were able to acquire lists of nodes in the system and then trace them to collect information on speed, hops, and AS boundaries between nodes. This allowed us to assess the cost of Gnutella traffic on the system.

The second question is answered by information from all three areas. The global topology allows us to see speed and client distribution across domains. Local topology is used to determine how many local nodes we have access to as well as the distribution across egress points from the domain. The file distribution allows us to determine how much we can depend on the locality of content.

3.2 Tools

The three primary objectives of our tool set was to collect a set of nodes, allow passive monitoring of the Gnutella network, and support active probing of the network. Our primary tool to achieve this was a modified Gnutella client. One of the benefits of the Gnutella system is that open source clients are readily available with which to monitor and probe the system. The one we chose was Gnucleus. This is an MFC Win32 client with an extensive feature set. After removing extraneous features, we were able to instrument the client as needed. As well as the modified client we also built a stand alone web crawler to more actively mine nodes from hostcaches.

Acquiring Nodes Based on the fact that Gnutella initializes the first connection by query from hostcaches, we designed our tool to talk to various hostcaches in order to collect active Gnutella nodes. As hostcaches are constantly refreshing themselves with active Gnutella hosts, we are able to collect a large portion of the active nodes in the Gnutella world. Using a standard PC, this application was able to collect about 16000 to 19000 active Gnutella nodes in under an hour. As well as using our stand alone application to collect these nodes, we were also able to instrument the client for collecting smaller node sets.

Active Probing Once we had a large data set of live nodes, we were able to run active probing tests. A trace like functionality was added to our modified client. With this we could ping the entire node set to determine speed, hops and domains crossed. By accessing the domain name server we could also determine what domain a node was currently in. We actively searched the network for content by querying this list of nodes. Limiting our search broadcasts to 1 hope allows us to accurately monitor the percentage of nodes containing certain content. Since the collection of much of this data took hours, all of this data was logged to reports for later analysis.

Passive Monitoring Our final set of functionality allowed us to passively monitor the data messages that were being passed on the network. We initially set out to monitor query and queryhit messages to determine what people were searching for and what clients said they had. Unfortunately we quickly found that while we received a large number of queries, very few queryhits pass back through us. Therefore we had to rely on active probing as explained above to collect information about what users where sharing.

After reviewing data from our initial query monitoring we soon recognized that Gnutella doesn't just send queries from users. It also has an option to automatically sends queries to find other hosts for a file that is being downloaded. By monitoring these queries we were able to collect a large amount of data about the actual files that users were downloading.

3.3 Results

Our first task was to validate our node collection strategy against other data sources. We did this by first identifying the Gnutella nodes within U.W domain and then using available trace data [3] for the domain to compare against our collected nodes. We regularly monitored the domain for a week-long period and determined that the number and activity of active nodes in the UW domain. Our result was in line with the numbers found in the trace data that ran over a similar period. This consistency with the trace implied that our node collecting tools were accurate. One of the conclusions from our node collection was that 20000 active nodes is about the right number for a snapshot of the Gnutella world. Therefore this was the maximum data size we focused on in our tests.

Global Topology The second set of experiments we did was to trace the routes and latency of each node to determine global topology. Since these traces took hours to run it was impractical to run them on one large data set and we therefore broke them into 4 data sets of 5000 nodes and ran them over 4 consecutive nights. Figures 1 and 2 show the results from these tests.

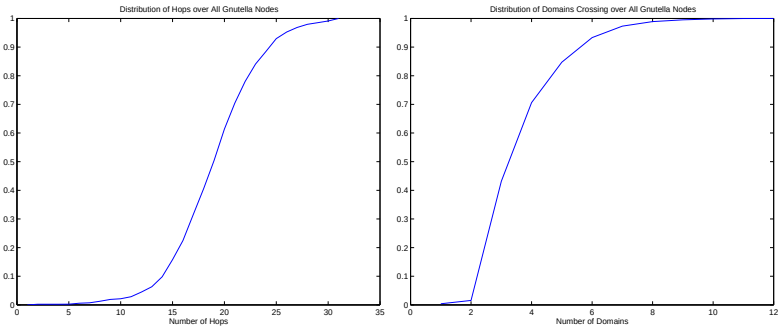

Figure 1: The distribution of Gnutella nodes in terms of hops and domain crossings between nodes

The graphs in Figure 1 display the number of hops and domain boundaries between nodes. The number of hops graph shows unsurprisingly that Gnutella nodes are distributed relatively widely and evenly across the internet. But with this data we were then able to determine the number of domain crossings that occur between clients. This data is key in understanding the extent of the expense incurred by this system. We used a relatively simple method for determining the domain of a node. First we queried the DNS for the name of the node from which we could extract the domain name. This quite often failed since many nodes on the Internet do not publish a name. So when a direct query failed, we backed off 1 or 2 nodes on our trace to see if the slightly closer nodes contained a domain node since these would often be router nodes which often publish names. Failing that we used the simple address class of the node to cluster it with other nodes.

As the Domain distribution graph shows, the majority of nodes cross 2 to 4 domains to talk to each other. This implies that an average Gnutella transfer traverses 1 to 3 transit links with the associated cost needing to be borne by the network. This also implies that several networks are burdened by Gnutella traffic even though they may have no Gnutella nodes on their system.

In Figure 2 the first graph displays how Gnutella nodes are clustered within network domains. In this graph one can see a surprisingly strong clustering within the top 10 domains. In fact the top 10 domains contain 87 percent of the nodes. This would imply that a network solution among these domains may be practical. The caveat with this data is that we were unable to determine the domain of 15.4 percent of the nodes, so ensuring this conclusion would require more testing. This data also shows that most domains have very few active Gnutella nodes. This implies that limiting access to just the local domain will give poor search performance.

The second graph in Figure 2 displays the round trip time between nodes. Less then 1 percent of the nodes can be reached within 10ms, and less than 2 percent of the nodes take over 200ms to reach. But within the range of 10ms to 100ms, the number of nodes distributes quite evenly. About 35 percent of nodes can be reached within 20ms, 60 percent can be reached in 50ms, and 80 percent can be reached in 80ms. This implies the latency between Gnutella nodes is usually small and that the nodes have fast connections to the Internet.

Local Topology After determining a reasonable set of data about the global topology, we then probed two dissimilar network domains to see what generalizations we could make about local topology. Our first domain was the University of Washington. This is a system of approximately

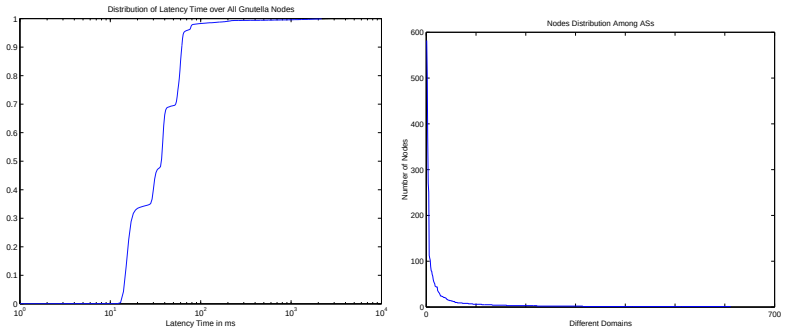

Figure 2: The distribution of Gnutella nodes in terms of Round Trip Time and the Domain they are in.

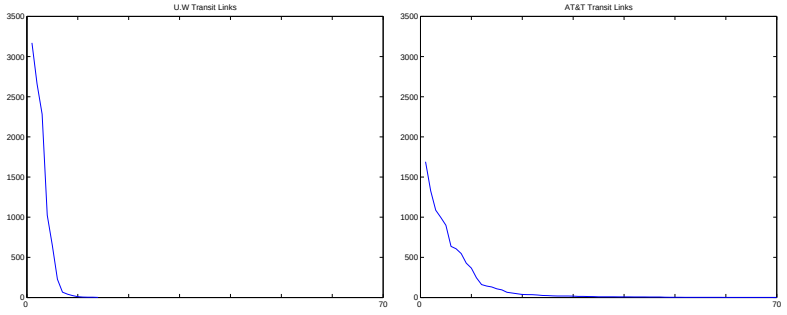

Figure 3: Transit Links for UW Domain

60,000 nodes connected to the northwest Internet hub. The second was the AT&T domain which is a major provider of independent and business user access as well as a tier one provider which provides connectivity and transit for smaller domains. Our goal was to determine how many transit links were connected to the domain. This would allow us to assess the viability of a system that determine which transit links are the most cost efficient.

As one can see in Figure 3, there is a large amount of variance between these two domains. The UW domain has 10 transit links while AT&T has 70. So a one size fits all automated solution does not seem practical. But there is a limited amount of transit links out of both domains. This implies that shared information for picking transit links could be passed and managed efficiently.

Content The final issue we explored with regards to locality of Gnutella nodes is content distibution. We designed our experiments to answer these three questions below, thus giving an answer to the question of content locality as a whole.

- What are the things people are interested and trying to find in Gnutella world?
- What are the things people find and download in Gnutella world?
- What are the things Gnutella nodes store?

File Type	Total Query Hits	Percentage of all Hits %
mp3 Files	210253	77
mpg Files	28757	10.5
avi Files	12072	4.5
jpg Files	6659	2.5
zip Files	2637	1

Table 1: File Type Distribution in Downloading

We collected data on the first two questions by using *passive network monitoring* to collect queries in the system as explained in the tools section. Manual queries reflect what people want while the automated queries show what people are downloading. We put our spy client at various Internet locations for a certain time period for several days. Over this time we monitored a total 374964 queries and 272798 downloads. They share in common that mp3 songs are the most demanding files in Gnutella world with videos in various format(mpg, mpeg, avi, rm) come in second, and JPG images rank third. Also, both queries and downloads show some interesting trends. For example, the newly released 007 movie, "Die Another Day", got over 20000 queries and 5000 downloads. Amazingly, there is an interesting distinction between query and downloads in keywords. For query, the most frequent keywords are almost all porn while for download, the most frequent keywords includes love, dance, and Christmas, leaving porn far behind. This interesting result implies that people search hard for porn, but usually end up not downloading that much porn.

The third question is about what content Gnutella nodes store. We use *active network probing* in which we modify the Gnutella code to have it send queries to all active nodes we collected to see what trends are evident. To get the set of keywords and strings to use for our probe we used the data collected with the passive monitoring. The results are shown in Table below.

The results conveys some interesting insights. Keywords data stored among hosts behaves similar to query hits, in which less than 2 percent of hosts contain porn files. All singers except the last one are on the top in both query and hits results, and most of them are distributed among 3 percent of all Gnutella nodes. And the last artist is actually a band from China, and it's interesting to still find 12 nodes, considering the fact that they are only popular among Chinese. Based on this results, we derive the argument that the most popular artists have a very large number of nodes while unpopular artists still have some nodes which forms a long tail in distribution. But it's not easy to verify this argument by enumerating all artists in terms of popularity.

It's really interesting to find that there lies gap between what users want with what Gnutella nodes store, and further with what users download. But they all show the fact that there exists bias or preference over some files of interest in Gnutella world.

3.4 Data Summary

The topology results imply that the nodes on average are spread over two to four domains, so locality seems like it would have great value in limiting these cross domain transits. But the domains contain a limited amount of nodes, so we still need a system that searches outside the current domain as well. This leads us to believe that we also need some mechanism for determining cost in choosing

Keyword	Total Query Hits	Percentage of all Hits %
Love	15217	5
Christmas	7326	2.5
Live	6817	2.5
Teen	6731	2.5
Boy/Boys	6644	2.5
Girl/Girls	5527	2
Night	4783	1.5
Kiss	4458	1.5
Sex	4034	1.5
Porn	3401	1
XXX	2769	1

Table 2: Top Keywords in Downloading

Keyword	Total Queries	Percentage of all Queries %
Sex	14884	4.5
Porn	13352	4
Girl/Girls	9285	2.5
Teen	8419	2
Love	7912	2
XXX	7568	2
Christmas	5174	1.5
Young	4998	1.5
Pussy	4691	1
Fuck	3612	1
Boy/Boys	4355	1

Table 3: Top Keywords in Searching

Keyword	Total Hosts	Percentage of all Hosts %
Love	2381	12
Girl/Girls	1840	9
Christmas	889	5
Boy/Boys	753	4
Sex	601	3
Porn	531	3
Fuck	469	2.5
Kiss	460	2.5
XXX	411	2
Teen	335	2

Table 4: Top Keywords in Content Storing

Singer	Total Queries	Total Hosts
Barenaked Ladies	4452	752
Beatles	3423	646
Madonna	2719	401
Elvis Presley	2467	553
Queen	2319	542
Nirvana	2113	420
Britney Spears	1986	621
Van Hales	1931	540
Frank Sinatra	1888	363
Christina Aguilera	1876	402
Beyond	N/A	12

Table 5: Top Singers in Gnutella World

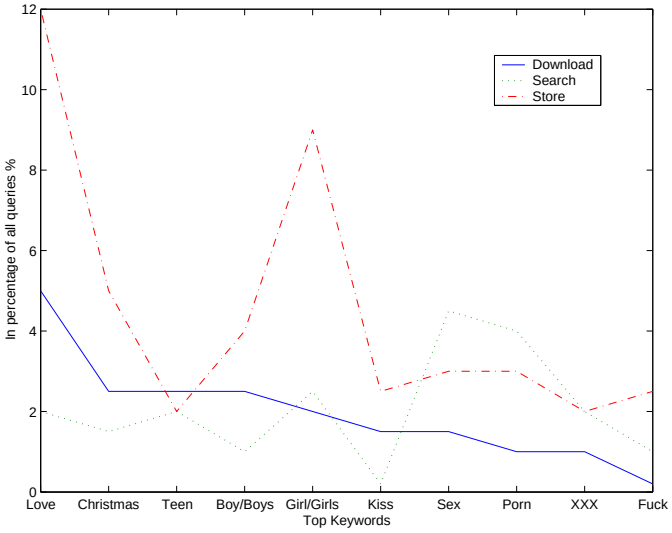

Figure 4: Top Keywords Contents in Gnutella World

transit links.

The content results present a distribution where the most popular keywords seems to be equally prevalent, but there is a long tail for the data. The data also shows a a temporal aspect with certain keywords being as popular as any other query over some short period of time. There is also a constant underlying distribution of requests for more obscure items. So popular stuff dominates but doesn't define the P2P systems.

Given the results of all these experiments, we get some sense of answers to the two questions raised in this section. Local sharing is certainly beneficial, given the fact that preferences do exist when people search and download. But the long tail on the distribution of content implies the need to search globally, so an efficient cost strategy is necessary. Additionally, in order to gain significant benefits from local sharing, it's essential for every client to share their content, especially content downloaded through expensive transit links.

4 Cost Efficient Client

Based on the results of our data analysis we determined that there is definite value in using locality in the P2P system. But we also saw that limiting the solution to only the local domain is not effective enough. So there will be queries that leave the local domain. To most effectively determine which external searches to carry out, some form of domain specific costing data will need to be provided to the clients.

We propose a host-based design to meet these goals for several reasons. The end to end principal implies that since we need user intervention to choose between files, the host level is the lowest level we can completely solve the problem. Also P2P clients and mechanisms change quickly and a network based design would find it difficult to keep up with these changes. In our host, we see a three part solution to meet the needs listed above.

First we want to make it easy for Gnutella clients to get local nodes by default rather than the completely random distribution in the current network. We manage to provide Gnutella nodes with this information about other local nodes under the current Gnutella framework, without making any modification of the protocol itself. We achieve this by setting up our own hostcaches, the novelty here is that we always update the hostcaches with only the local hosts which we determine by continually crawling the network. The client can then access this by adding it to the list of hostcaches to always check. Besides setting up the local hostcache for network initialization, we also build a special Gnutella client in which local nodes information is embedded into its messages, we have this Gnutella client talk to all local nodes with these messages to spread local information among all local nodes.

Next since we want the clients to bias their connections toward local nodes we need to add some mechanism to the clients to manage these nodes separately from the other nodes and balance them with nonlocal nodes. This is necessary since many content searches need to to leave the local domain in order to provide a suitable response to the user. This also avoids the problem of the system becoming closed if all the clients in the domain are biased toward connecting with each other.

Finally even if we could find enough nodes locally to provide a reasonable system, the network is constantly reconfiguring and since there are several different clients available it would be difficult limit all queries to the local domain without some LAN specific solution. So we need a way of providing domain costing data to the clients that they can use to help determine file selection.

4.1 Implementation

Based on the above solutions we implemented a prototype system that provides locality. The system consists of a hostcache as described above, a costing module compiled into the client, and some modifications to to the node cache handling and search interface. The client we used is based on the Gnucleus open source discussed in the Tools section.

The Host Cache we built follows the standard hostcache interface and design published by Gnutella. To this hostchache we then add a web crawler to constantly crawl the existing host caches looking for local nodes that have joined the network. Determining whether a host is in the network is a matter of checking against the address ranges of the local domain. This information is disseminated by the costing module below. The messages to other hosts are a matter of sending a packet at regular intervals using the defined protocol.

Updating the node handling for the client is reasonably straightforward since client maintains several node caches already. Our solution just adds another. For our tests rather than actually implement a new node cache we overloaded the existing permanent node cache. Also since we were the only node in the system we didn't worry about over-using local nodes. We used all that we had all the time and relied on other nodes to disseminate them to non-local nodes.

Distributed Costing Module The route costing is handled in a distributed manner except for the setting of the costing data and determination of the bounds of the current domain. This shared data is entered by someone that knows the current address ranges and costs of the transit links out of the domain and is stored at the host caches described above. This data can then be downloaded when accessing the hostcache. For now our implementation does not worry about this transfer and instead loads the data from a configuration file with the client.

The hosts each maintain two tables for determining route costing. The first table maintains a cache of hosts and transit links they take out of the domain. The transit links are determined by using trace type functionality described below and are updated every time the host receives a set of hosts in response to a content search. Once the transit link is known a second transit link table provides information on the cost for the route based on the shared costing data. The client gets the cost for any given host by making a function call to the route costing module with the host IP and the size of the transfer.

We use a route tracing mechanism for finding the transit links. A UDP broadcast is sent over ICMP with the TTL set to progressively higher numbers. Each expiration returns an error giving the address of the site that the packet reached. In this way a path to the final destination is built up. We terminate when we leave the domain as defined by the shared costing data.

While arbitrarily complex costing can be supported in the system, we currently have implemented

Connections	UW Base	ATT Base	UW New	ATT New
% Local	0	1.3	1.2	23.4
% Peering	28.1	24.2	48.0	68.2
% Total	28.1	27.5	49.2	91.6

Table 6: Cost based connections with base and new Gnutella Clients

a binary costing system. Transit links either cost money or they don't. This makes it easier to carry out analysis and makes the results more applicable to other domains since costs for the transit links vary widely across domains.

4.2 Results

Our system is designed to achieve three goals - find more local nodes to connect to, use these nodes aggressively for searches, and use domain costing data for non-local file selection. To determine the effectiveness of our implementation, we ran a comparison of this system against the standard Gnutella client with no costing functionality. We ran these tests from the same two domains (UW and AT&T) that we used in our data collection and analysis. The first set of tests was to see how well we did with connections. The second set of tests determined our success in searching, and the costing data was used to provide peering link functionality for both tests. The results of these tests are shown in Table 6 and Table 7 respectively.

We carried out our connection test by attempting to connect to 1000 Gnutella nodes and determining what percentage of those nodes are cost efficient. A cost efficient node is determined to be either one in the local domain or one that uses a peering link as opposed to a regular transit link. We do not know the peering links of the 2 domains we tested in, so we instead randomly chose 25 percent of the transit links as peering links.

As Table 6 shows there is disparity between the level of success of our solution in the two domains, but there is significant improvement in cost efficiency in both cases. The percentage of cost efficient nodes for the base Gnutella client is set randomly so it is driven primarily by the peering relationships. For both domains the connection to local nodes is almost non-existent. The new client on the other is able to find all the local clients as well as using the preset peering relationships to fill out the connection list. This nearly doubles the cost effiency of connections for the UW domain and makes over 90 percent of the connections cost efficient in the AT&T domain.

In our search test, we used the connections we made in the connections test and then directed a set of searches over them. Using the top 10 artists we determined in our content analysis, we sent out searches and then looked at the the best 100 responding nodes for each string based on locality. We also kept track of total hits to determine if there was a fall off in number of returned responses.

In Table 7 you can see that our results. The number of total hits is slightly lower in the UW domain and slightly higher in the ATT domain, but both are in line with current Gnutella performance implying that our cost efficient client does not hinder searches. The number of cost efficient nodes that we connected to were in line with the connections data from Table 6. This is encouraging in the context of a client being able to act independently and improve its own cost efficiency.

Search Hits	UW Base	ATT Base	UW New	ATT New
Average per String	1872	1932	1642	2049
% Local	0	2.1	3.8	32.2
% Peering	22.3	27.3	52.1	58.4
% Total	22.3	29.4	55.9	90.6

Table 7: Cost based searches for popular strings with base and new Gnutella Clients

5 Discussion

We chose the Gnutella system since with it's open source model it allowed access to the system protocols and to a modifiable client that we could tailor to our data collection. One of the issues with our study is that we rely so strongly on the Gnutella network for our data. The assumption being that the Gnutella network topology and file distribution are indicative of P2P systems in general. The topology argument is not unreasonable. The distribution schemes of P2P networks are understood and many new P2P clients based their initial design off of the Gnutella source code. The similarity in file distribution is more dubious. The Gnutella client transferred an order of magnitude fewer bytes than the KaZaA client in the University of Washington study. We are assuming that while the numbers and content may be different the file distribution is still similar.

Another issue for future research to calculate the actual dollar amounts saved for the client implementation using the real peering and transit relationships of a domain. Our research uses binary costing for ease of analysis and wider applicability. But in an actual system one very expensive link may may skew the real costs of the system dramatically.

6 Conclusions

P2P file sharing has occupied a large portion of Internet traffic, and it is continuing to grow dramatically. The problem with current P2P systems such as Napster, Gnutella, KaZaA etc, is that none of them support locality as a cost metric, which leads to huge costs for network domains and transit providers. To anyalyze one such system and determine an effective solution, we developed a tool set to collect, probe and monitor the global topology, local topology, and content distribution in Gnutella. From this investigation we discovered that local sharing would be beneficial as long as it integrates a cost efficiency strategy. Based on these conclusions we presented a three part host based solution that includes the following.

- Host Caches that cache and broadcast local node addresses
- A modified client that uses these local addresses aggressively in searches
- Shared domain cost information to choose the most cost efficient connection over transit links when necessary

A Gnutella client with these solutions was implemented and tested over two dissimilar real world network domains. Our results showed that large gains in cost efficiency were achieved in both

domains without sacrificing search quality. We believe that since Gnutella reflects the topology, traffic, and content distribution of many current P2P systems, these systems can benefit from the solutions presented as well.

References

[1] D. Andersen, H. Balakrishnan, M. Kaashoek, and R. Morris. The case for resilient overlay networks, 2001.

[2] Ian Clarke, Oskar Sandberg, Brandon Wiley, and Theodore W. Hong. Freenet: A distributed anonymous information storage and retrieval system. *Lecture Notes in Computer Science*, 2009:46+, 2001.

[3] Steven Friederich. Bandwidth restrictions save almost 1 million. *University of Washington Daily*, October 22, 2002.

[4] Gnutella. http://gnutella.wego.com.

[5] Intel. Peer to peer computing, p2p file-sharing at work in the enterprise, white paper, 2001.

[6] KaZaa. http://www.kazaa.com.

[7] Napster. http://www.napster.com.

[8] William Norton. Internet service providers and peering, 2001.

[9] Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard Karp, and Scott Shenker. A scalable content addressable network. Technical Report TR-00-010, Berkeley, CA, 2000.

[10] Stefan Saroiu, Krishna P. Gummadi, Richard J. Dunn, Steven D. Gribble, and Henry M. Levy. An analysis of internet content delivery systems. volume December, 2002.

[11] Stefan Saroiu, P. Krishna Gummadi, and Steven D. Gribble. A measurement study of peer-to-peer file sharing systems. In *Proceedings of Multimedia Computing and Networking 2002 (MMCN '02)*, San Jose, CA, USA, January 2002.

[12] S. Savage, T. Anderson, A. Aggarwal, D. Becker, N. Cardwell, A. Collins, E. Hoffman, J. Snell, A. Vahdat, G. Voelker, and J. Zahorjan. Detour: a case for informed internet routing and transport. Technical Report TR-98-10-05, 1998.

[13] Stefan Savage, Andy Collins, Eric Hoffman, John Snell, and Thomas E. Anderson. The end-to-end effects of internet path selection. In *SIGCOMM*, pages 289–299, 1999.

[14] Subhabrata Sen. Analyzing peer-to-peer traffic across large networks.

[15] N. Spring, R. Mahajan, and D. Wetherall. Measuring isp topologies with rocketfuel. In *Proceedings of ACM/SIGCOMM '02*, August 2002.

[16] Ion Stoica, Robert Morris, David Karger, M. Frans Kaashoek, and Hari Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *Proceedings of the 2001 conference on applications, technologies, architectures, and protocols for computer communications*, pages 149–160. ACM Press, 2001.

[17] ZeroPaid.com. http://www.zeropaid.com.