Finding Good Peers in Peer-to-Peer Networks

Murali Krishna Ramanathan, Vana Kalogeraki, Jim Pruyne

Abstract

As computing and communication capabilities have continued to increase, more and more activity is taking place at the edges of the network, typically in homes or on workers desktops. This trend has been demonstrated by the increasing popularity and usability of "peer-to-peer" systems such as Napster and Gnutella. Unfortunately, this popularity has quickly shown the limitations of these systems, particularly in terms of scale. Because the networks form in an ad-hoc manner, they typically make inefficient use of resources. We propose a mechanism, using only local knowledge, to improve the overall performance of peer-to-peer networks based on interests. Peers monitor which other peers frequently respond successfully to their requests for information. When a peer is discovered to frequently provide good results, the peer attempts to move closer to it in the network by creating a new connection with that peer. This leads to clusters of peers with similar interests, and in turn allows us to limit the depth of searches required to find good results. We have implemented our algorithm in the context of a distributed encyclopedia-style information sharing application which is built on top of the gnutella network. In our testing environment, we have shown the ability to greatly reduce the amount of communication resources required to find the desired articles in the encyclopedia.

1 Introduction

As computers become more pervasive and communication technologies advance, a new generation of communication models will be deployed over the Internet. Peer-to-peer models such as Napster [11] and Gnutella [10] are increasingly becoming popular for sharing information and data through direct exchange. These models offer the important advantages of decentralization by distributing the storage capacity and load across a network of peers and scalability by enabling direct and real-time communication. In fully decentralized peer-to-peer networks there is also no need for a central coordinator; communication is handled individually by each peer. This has the added benefit of eliminating a possible bottleneck in terms of scalability or reliability.

In these networks, a node becomes a member by establishing a connection with at least one peer currently in the network. Each node maintains a small number of connections with its peers. Messages are sent over multiple hops from one peer to another with each peer

Murali Krishna Ramanathan, CS Department, Purdue University, IN, 47906, rmk@cs.purdue.edu
Vana Kalogeraki, Hewlett-Packard Laboratories, Palo Alto, CA, 94304, vana_kalogeraki@hp.com
Jim Pruyne, Hewlett-Packard Laboratories, Palo Alto, CA, 94304, pruyne@hpl.hp.com

responding to queries for information it has stored locally. For example, to search for a file, a node broadcasts a search request to its peers, its peers propagate the requests to their own peers and so on.

Many disadvantages with this approach have been pointed out. Among them is the arbitrary nature in which the networks are formed and maintained. Scale becomes limited because messages are propagated across all nodes in the network, including those with high latencies. Each hop contributes to an increase in the bandwidth on the communication links and furthermore to the time required to get results for the queries. The bandwidth for a search query is proportional to the number of messages sent which in turn is proportional to the number of peers that must process the request before finding the data.

In this paper we propose an automatic mechanism for establishing connections between "good" peers, by taking into consideration the *interests* of the peers. We capture the interests of the peers through the number and types of files maintained and provided by the peers in the network. We consider that two peers have similar interests if they are able to provide files to their each other's requests. Nodes learn about the interests of their peers by monitoring the replies they receive to their requests. Therefore, nodes decide whom to connect to or when to add or drop a connection based on this local information. Nodes with high degree of similar interests are considered "good" peers. By manipulating the connections between the peers, our mechanism tries to guarantee that nodes with a high degree of similar interests are close to one another on the network and that as distance between the peers grows the similarity of the peers' interests decreases.

Our mechanism has the following advantages:

- Reduces the number of messages in the network and also the number of peers that propagate the messages (search requests) to find relevant information.
- Allocates resources (networking, processing and storage) efficiently as each peer keeps only a small number of connections only to peers with the same interests.
- Scales well with respect to the number of peers.

To illustrate our mechanism, we have built a decentralized online encyclopedia. The encyclopedia is organized as a network of peers, each peer maintains a set of files (articles). There is no centralized directory that holds all the files, the files are kept at the users' machines. The files can be updated very frequently as in the case of daily news or can be more static. Also, the files are not necessarily unique; the same files (*i.e.,* popular articles) will be available at many locations. Our mechanism guarantees that as users search for articles in the encyclopedia, the search requests are efficiently propagated only to peers with similar interests.

Our mechanism operates under the following assumptions:

- Each node has a large number of connections with other peers and there exists a path between any two nodes in the network. Thus, the network remains highly connected, even if peers disconnect from the network at random.
- Peers have similar interests, therefore, the same files are available from many peers. This is a reasonable assumption, because in our encyclopedia application, many users are reading the same articles of the encyclopedia and therefore download the same files.

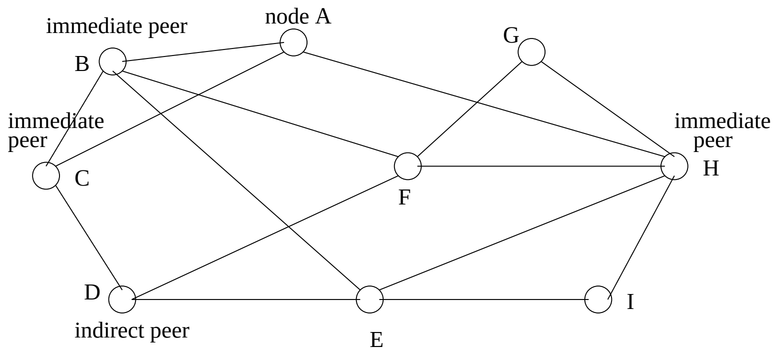

Figure 1: The Peer-to-Peer Network.

The paper is organized as follows: Section 2 presents the System Model and Metrics. In Section 3 we describe our mechanism for finding good peers. In Section 4 we describe our experimental results. Section 5 presents the Related Work and Section 6 concludes the paper.

2 System Model and Metrics

We assume a logical network of nodes (peers) in which each node maintains a connection with a group of other peers. The model is based on common peer-to-peer file searching networks such as gnutella [10]. The number of connections is typically limited by resources at the peer. Two nodes $p(i)$ and $p(j)$ are called *immediate peers* if there is a direct connection between the nodes. Two nodes $p(i)$ and $p(j)$ are called *indirect peers* if there exists a communication path between these nodes. Note here, that, in some cases two nodes may not be connected at all. Also, while connections are typically initiated by one peer, they are symmetric in the sense that either peer can send messages or choose to disconnect at any time.

In Figure 1 we show an example of a peer-to-peer network. In the example, node A has three immediate peers, nodes B, C and H. The rest of the nodes are indirect peers. For example, node D is an indirect peer because there is a communication path between nodes A and D (from A to C to D).

Each peer $p(i)$ is characterized by the capabilities of the processor on which it is located. The processor is characterized by its CPU speed $p_{cpu}(i)$, the size of its local memory $p_{mem}(i)$ and the size of its disk space $p_{disk}(i)$. The node also has a limited amount of bandwidth to the network as noted by $p_{band}(i)$. The communication link between two nodes $p(i)$ and $p(j)$ is characterized by the bandwidth available to it as $p_{band}(i, j)$. These lead to the constraint $\sum_j p_{band}(i, j) \leq p_{band}(i)$. Also, to give high priority to the node's request, we can restrict the bandwidth used for incoming requests (propagated by other peers) to be a small proportion of the total bandwidth on the communication link. This leads to a limit on the number of connections a peer can maintain. We denote the number of connections a peer is maintaining by $p_{conn}(i)$. Each of these resources has a maximum and a current value measured at runtime. More than one peer can be located on the same processor at the same time. In this case, the peers have the same IP address but are connected to different ports.

Each peer maintains a local repository of files. Each file is characterized by meta-data. The meta-data includes the title of the file, the topic, the author, the generation (and possibly expiry) date and keywords. In addition to that, the meta-data also includes a *"reputation"* value which rates the article. The reputation metric takes values between *one* and *five*; *one* corresponds to an article of a low value, while *five* corresponds to an article of high value. Note here that different users have different likes, therefore the ranking mechanism corresponds to the users' personal interests. Typically users give a high reputation value to the articles they like best. The meta-data for the files is stored in a hash table and the keywords of the files are used as the hash key.

Each node i is characterized by its $horizon(i)$ that determines how far the messages sent by node j will be propagated in the network. The node sends messages to search for files among its peers. Its peers will in turn forward the messages to their own peers and so on. In general, a search message will be propagated $(\overline{p_{conn}})^{horizon}$ times where $\overline{p_{conn}}$ is the mean number of connections maintained by the peers. This results in a large number of messages propagated in the network. We therefore desire a method to reduce node's i $horizon(i)$ to reduce both the number of messages and limit the bandwidth on the communication resources.

Each node i associates the following information with each of its peers j:

- $Importance(j)$: a metric that represents the relative importance that the peer j has for the node i and captures the interests of the peers. Node i connects to the peers with the highest importance and furthermore, high degree of interests. The same peer can have different importance for different nodes.

- $percQueryHits(j)$: the percentage of replies (QueryHits) received from the peer j computed as the number of replies (QueryHits) received from the peer over the total number of replies (QueryHits) received from all peers as a response to the node's queries.

- $averNumHops(j)$: a metric that represents the distance (average number of hops) of node i from indirect peers whose replies (QueryHits) are propagated to node i through its immediate peer j.

- $connectionTime(j)$: the amount of time the connection with the immediate peer j is active.

3 Determining Good Peers

The main objective of our algorithm is to establish connections between "good" peers, that is, peers with high degree of interests. Our aim is to (1) minimize the horizon of each peer and therefore minimize the number of messages in the network and, (2) maximize the probability that peers with similar interests are connected together so that the results are obtained faster from the peers.

3.1 Messages in the Peer-to-Peer Network

The peers in the network communicate by exchanging messages. The format of the message header (shown in Figure 2) is:

UUID	descriptor_id	payload_descriptor	group_id	ttl	hops	payload_length

Figure 2: The Message Header.

- *unique message id (UUID):* which ensures that every query message is distinguishable from any others.
- *descriptor_id:* a 16-byte string that uniquely identifies the sender of the message. If the message has been forwarded more than once, the descriptor_id refers to the node that forwarded the message last.
- *payload_descriptor:* the type of message sent which can be one of the following: `Connect`, `Accept_connection`, `Query`, `QueryHit`.
- *group_id:* the name of the group where the message should be propagated.
- *time to live (ttl)*: the number of times the message will be forwarded among the peers before it expires. The initial value given to the TTL parameter is the *horizon* of the peer.
- *hops*: the number of hops the message has already been forwarded in the network.
- *payload length*: the length of the descriptor immediately following the header.

These are the messages exchanged among the peers in the network:

- *Connect:* used to discover active nodes on the network. The node that receives a connect message can reply with an Accept_connection message.
- *Accept_connection:* used as a response to a Connect message and includes the IP address of the sender node.
- *Query:* this is the primary mechanism for searching in the network and includes the constraint that will carry the search operation. If the node that receives a Query message has a reply, it responds with a QueryHit message.
- *QueryHit:* used as a reply to the Query message and includes information so that the recipient can acquire the corresponding data.

3.2 Connecting to the Group of Peers

Each node p maintains a list of peers $PeerList_p$ in the network. These are peers with which the node had previous connections to and is likely to connect to in the future. The peer list is updated dynamically based on the user's interests and ordered based on the importance of the peers. Since the nodes are not always connected to the network, some of these peers may not be active when the node tries to connect to them. If the node cannot find any peer to connect to, it can contact a centralized server *e.g.,* Napster [11] to get a list of currently active peers in the network.

A node connects itself to the network of peers by establishing a connection with at least one peer currently on the network. To connect to the network of peers, the node constructs a `Connect()` message containing his `descriptor_id`, the `Connect` payload descriptor, the number of times (`ttl`) the message should be propagated in the network and the name of the group (`group_id`) where the message should be sent. It then uses the connect message to actively probe the network for peers. In the paper we are not concerned how the node discovers the `group_id`, we assume that there are specific services called *name servers* and the node obtains the group name by invoking them. The connection to a group can be considered as an initial thrust for the node to find its peers; this is a group of peers with similar interests.

When a node receives a `Connect()` message, it decides whether it should accept the connection from its peer. The decision depends on a number of factors, such as the resource capabilities of its peer and the bandwidth on its communication links. For example, the node can choose to establish a connection with a peer with which it had a stable connection in the past over another peer that was frequently disconnecting. The node decides to accept a connection if the number of its peers is less than the maximum number of connections `MAX_CONNECTIONS` it can accept and replies with an `Accept_connection` message.

When the node receives the `Accept_connection` message, it extracts the `IP_ADDRESS` from the message and connects to the sender peer. Also, for future connections, the node updates its peer list by adding the peer's address. This is based on our assumption, that these are peers have similar interests and if a node decides to connect to a "good" peer once, then, it is likely that it will connect to the same peer later in the future.

A node can choose to disconnect from the network at random. In this case, its peers will decide whether they should issue a `Connect` message to find another peer or they should keep the existing number of connections.

3.3 Searching in the Peer-to-Peer Network

A node searches in the network by sending Query messages to its peers. The Query message contains a constraint which will be evaluated locally in each peer to determine what results to return. Typically, the constraint includes a set of keywords, such as the topic of the file.

When the node receives a Query message, it evaluates the constraint against the metadata of the documents in its local repository. If the constraint evaluates successfully, the node generates a QueryHit message that includes the files corresponding to the constraint, and sends it to the immediate peer from which it received the Query message. The Query Hit message follows the query path to reach the node that initiated the search.

In addition, the node decrements the TTL value in the message's header and forwards the message to each of its immediate peers. To provide a termination condition so that messages are not propagated indefinately in the network, a node that receives a message with TTL value zero, stops forwarding the message.

When the QueryHit messages reach the node who initiated the search, they are stored in a buffer until the replies from all the peers are collected. The order with which the replies are displayed is based on the files' reputation values; files with high reputation values are returned first. The node records all the replies it receives (both from the immediate and the indirect peers) and also the peers who provided the results. It uses the results to determine which are "good" peers to connect to.

3.4 Determining the Peer's Horizon

Each node p in the network has information about its immediate peers. For example, it knows the logical (one hop) and physical (IP address) distance of the peers. Also, the node discovers the files maintained by the immediate peers by sending search requests and recording their replies.

Information about indirect peers is obtained through p's immediate peers. When p sends a `Query` message to its immediate peers, they will propagate the message to their own immediate peers. Replies from indirect peers are sent through the same path to node p. Node p records all the replies it receives to its search requests, both from its immediate and indirect peers. Note though that p's view about indirect peers may not be complete, because only some of the indirect peers may have replied to p's requests. Furthermore, the topology of the network changes dynamically, and therefore, it is likely that new peers may have joined the network.

Assume that node p generates a Query message $Query_q(p)$ and sends it to each immediate peer q. The immediate peer q will in turn propagate the message to each of its own immediate peers s. Let $QueryHits_{p,q}(s)$ be the number of QueryHit messages generated as replies from indirect node s and sent to node p through the immediate peer q. Let $numHops_p(s)$ be the logical distance between nodes p and s at the time of the request. The distance of the peers may change (grow or shrink) at the time of later requests. The node p computes the average number of hops of its immediate peer q as:

$$averNumHops_p(q) = \frac{QueryHits_p(q) + \sum_{s,s:indirect}(QueryHits_{p,q}(s) * numHops_p(s))}{QueryHits_p(q) + \sum_{s,s:indirect} QueryHits_p(s)} \quad (1)$$

The $averNumHops_p(q)$ captures the number of hops taken, in average, for replies that come through the immediate peer q to reach node p. The replies include both the replies generated from the immediate peer q and the replies generated from indirect peers and propagated to node p through its peer q. Those replies that originate from indirect peers contribute to the relative importance of the immediate peer q in inverse proportion to the logical distance from the immediate peer. An immediate peer q with a large average number of hops indicates that it receives replies from peers that are located farther from the requesting node.

Note though that, even if an immediate peer does not generate any replies for the search request, this does not necessarily mean that it is not good a "good" peer any more, because of two reasons: (1) the peer may give good results for other subsequent queries and, (2) it propagates the search messages to indirect peers in the network and if the node disconnects from this immediate peer, it may loose connection to those peers. By calculating the importance of the immediate peer (Section 3.4) we can decide whether we need to replace this immediate peer.

Example: The values of the averageNumHops metric are illustrated in the following example. Consider a network of four peers A, B, C and D as shown in Figure 3. Let nodes B and C be immediate peers to node A and let node D be an immediate peer to node B (therefore, an indirect peer to node A). Assume that nodes B, C and D give 80, 100 and 20 QueryHits, respectively, as a reply to queries from node A.

Note here that both peers B and C give a relatively high number of QueryHits. Note also that through peer B, node A receives replies from other peers, such as node D and the total

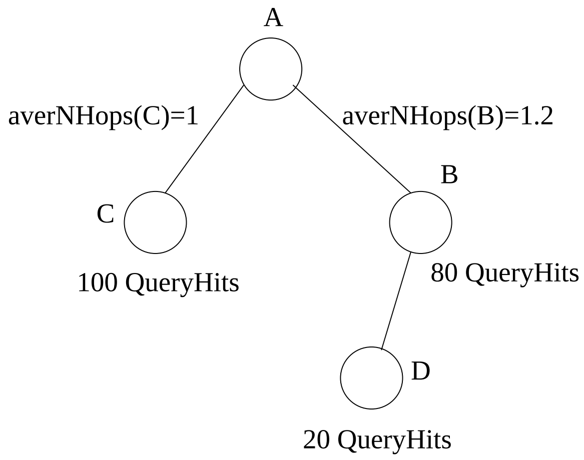

Figure 3: Average Number of Hops for Node's A Immediate Peers B, C.

number of QueryHits that come through node B is equal to the total number of QueryHits that come through node C ($= 100$).

Using equation (1) we calculate the average number of hops for node's A immediate peers, B and C as: $averNumHops_A(B) = 1.2$ and $averNumHops_A(C) = 1$. The numbers indicate that although the total number of QueryHits propagated through both immediate peers B, C is the same, replies from node B have to travel a larger number of hops. Our objective, is to reduce the A's horizon (minimize the number of hops) and at the same time maximize the number of QueryHits that node A.

3.5 Importance of a Peer

We determine "good" peers by evaluating the relative importance of both the immediate and indirect peers.

Node p computes the importance of an indirect peer s through the percentage of QueryHit messages generated by s as:

$$percQueryHits_p(s) = \frac{QueryHits_p(s)}{\sum_{i,i:all\_contributing\_peers} QueryHits_p(i)}$$

To evaluate the importance of an immediate peer q, node p also has to consider how many replies from indirect peers are being propagated through the immediate peer q. This is important because, if node p disconnects from its peer q, node p may loose the communication with some other indirect peer. If an indirect peer is important, then p may choose to connect to that peer directly. The Importance of an immediate peer q at time t is computed as:

$$Importance_p(q,t) = \alpha * \frac{percQueryHits_p(q)}{averNumHops_p(q)} + (1-\alpha) * Importance_p(q,t-1)$$

where we are using exponentially weighted averaging to determine the mean Importance value of peer q. The mean value of $Importance$ is not the average of the previous measured values, but is determined by a sequence of measurements with greater weight being given to more recent measurements. If the number of connections is stable, the above calculation gives a good

$PeerSelection\_Algorithm(peer\ p)$

for all my peers q

find indirect peer s with maximum $percQueryHits_p(s, m)$

find immediate peer q with least $Importance_p(q)$

if $percQueryHits_p(s) \geq percQueryHits_p(q)$

make a direct connection from peer p to peer s

if $numConnection_p \geq MAX\_CONNECTIONS$

remove peer q with least $Importance_p(q)$

Figure 4: Pseudocode for the PeerSelection Algorithm.

approximation to the mean value. If the behavior changes dynamically, the calculation tracks current behavior with a large value of α yielding rapid response to changing conditions, and a small value of α yielding more smoothing and less noise.

Our algorithm determines "good" peers as peers with high importance. These are peers that provide a high percentage of QueryHits and are connected over the longest time period. The advantage is, that, if two immediate peers are connected over a long period of time, this is a strong indication that the peers have similar interests and should remain connected.

3.6 Peer Selection Algorithm

Periodically, each node p evaluates its immediate and indirect peers. The node decides to make an indirect peer s an immediate peer, when the $percQueryHits_p(s, m)$ of peer s becomes greater than the minimum percentage of QueryHit messages of its immediate peers. Then, node p probes peer s to make a direct connection. If node s accepts the connection, a direct connection between nodes p and s is established. If node p has excited the maximum number of connections `MAX_CONNECTIONS`, then it removes the immediate peer with the least Importance. The pseudocode for the PeerSelection algorithm is illustrated in Figure 4.

3.7 Stability

The stability of the system is affected by the rate with which each node executes the peerSelection algorithm. The effectiveness of our PeerSelection algorithm can be best evaluated by considering the frequency with which nodes make connections to new peers and disconnect from old ones which, in turn, is greatly influenced by the user's behavior. For example, if the user is actively searching for articles in the encyclopedia, the peerSelection algorithm should run in a high frequency to find the best peers. On the other hand, if the user is inactive, the PeerSelection algorithm is not running at all.

The effectiveness of our peerSelection algorithm can be further improved if the characteristics of the applications are also considered. For example, for our decentralized online encyclopedia, users usually read articles of the encyclopedia in the morning, so there is high traffic during the early hours and less traffic the rest of the day.

By monitoring the user requests and the replies sent from the peers, our algorithm guarantees that peers with similar interests are connected together and dynamically adjusts the connections between "good" peer as the behavior of the users or the peers change.

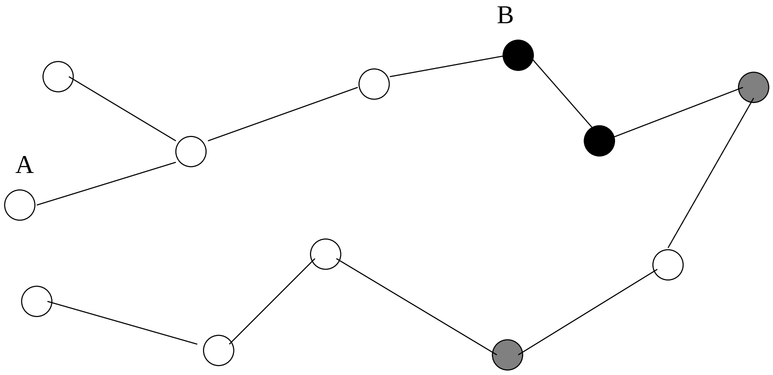

Figure 5: Scenario 1: Initial Topology

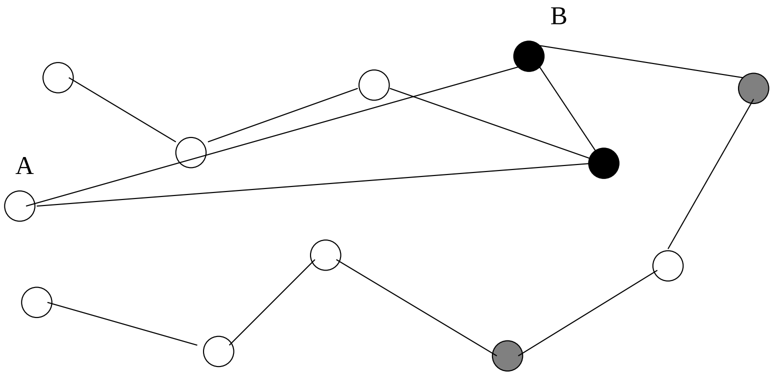

Figure 6: Scenario 1: Final Topology

4 Experimental Results

We used two scenarios to observe the working of our algorithm in the peer-to-peer network. In the first scenario, we determine how accurately and quickly a peer is able to find other peers with similar interests. In the second, we see how the algorithm adapts a peer's connections as the interests are changed.

4.1 Scenario 1

A real time experiment was conducted with 12 peers with each of them generating queries based on assigned interests. The initial topology is shown in figure 5. We monitored the state of peer A and restricted it to a maximum of three connections. We also set the initial TTL of its query messages to three. Peers with filled circles represent those with interests similar to our test peer, A. That is, they are more likely to be able to return results for a given query. The darker the circle the more similar the interests. Each peer also maintained a set of "content" files that other peers may find. Each content file has meta-data associated with it containing a title, topic, keywords and rank. Five hundred queries were generated during the test, and the results were compared to a standard gnutella protocol. A query hit occurs only when the query matches completely either with one of the keywords or the title of the file.

The final state is shown in figure 6. We see that A broke one connection from its initial allocation, and made three new connections to peers with similar interests. We also observed that our test peer made only 6 to 7 "reconnection" decisions before coming to a stable state and these decisions are made near the beginning of the experiment. The large number of reconnection decisions during initialization can be attributed to the near "zero knowledge" of the peer connections. Over time, the peer learns about good connections based on the algorithm and gets connected to more stable and important peers. This lead to decrease in the number of reconnection decisions later. Once the peer got a good set of immediate peers, the reconnections decisions are made rarely and are mostly related to change in the kind of queries being sent. This observation is vital to show the overall stability of a system using our algorithm.

By comparison, we observed in a pure gnutella network without our peer selection algorithm that only query hits from peer B are received because of the TTL limit of 3 even though other peers with relevant information are in the network. This demonstrates the "so near, yet so far" nature of the gnutella protocol. Four peers were contacted only to get the results from one of them. This resulted not only in poor query hits for A, but also wasted resources (bandwidth and query processing time) of the "just" forwarding nodes. In our model A found B to be a good peer and made it an immediate peer. This resulted in the useful nodes just 2 or 3 hops away from A. In course of time, they connected to A directly. This had two advantages:

- More query hits received by A. This is show in Figure 7. Notice how the number of hits grows dramatically and continues to outpace the pure gnutella implementation. This is directly due to our algorithms ability to find and connect to peers with a history of providing results.

- Fewer messages were propagated in the network. Figure 8 shows the mean number of hops a query hit must traverse before returning to the search peer. We see that after a brief instability when the test peer is searching for good peers, the number dramatically decreases. This reduces bandwidth used by query response messages, and could be used as an indication that horizon can be reduced which reduces the propagating of query messages as well.

4.2 Scenario 2

In this scenario we show how the algorithm adapts as the interests of a peer change over time. Six nodes were used for this experiment with 1500 queries being sent from the testing peer (Figure **??**). We assumed 3 peers containing documents related to different topics, A, B, C respectively. The test peer sends out a query stream of the form:

5A's, 50B's, 100C's, 150A's, 150B's, 500C's, 300A's,100B's

where we assume that peer A has the data requested by the A queries, peer B has the results of the B queries and so forth.

We set the initial TTL on the query messages to two, and also restricted the peer to two simultaneous connections. When the 50 B's were sent, B became the immediate peer of the testing peer much as we showed in Scenario 1. However, we observed that disconnection decisions were not made immediately after a change in the type of query message being generated.

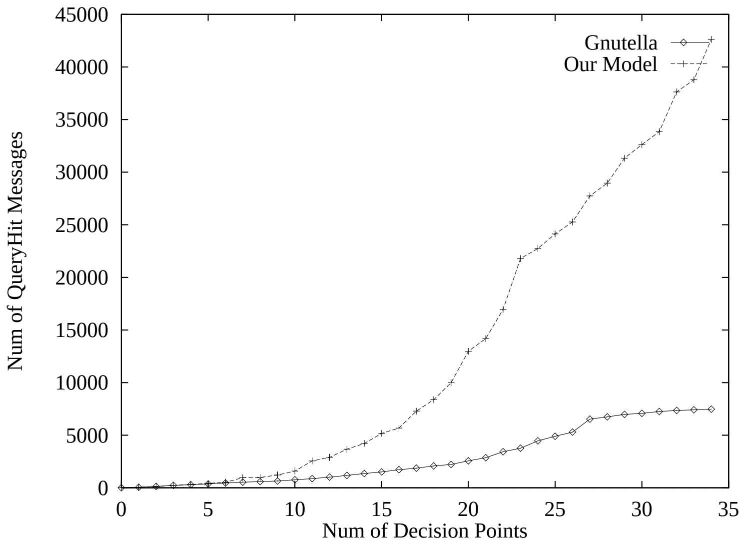

Figure 7: Cumulative number of query hits received

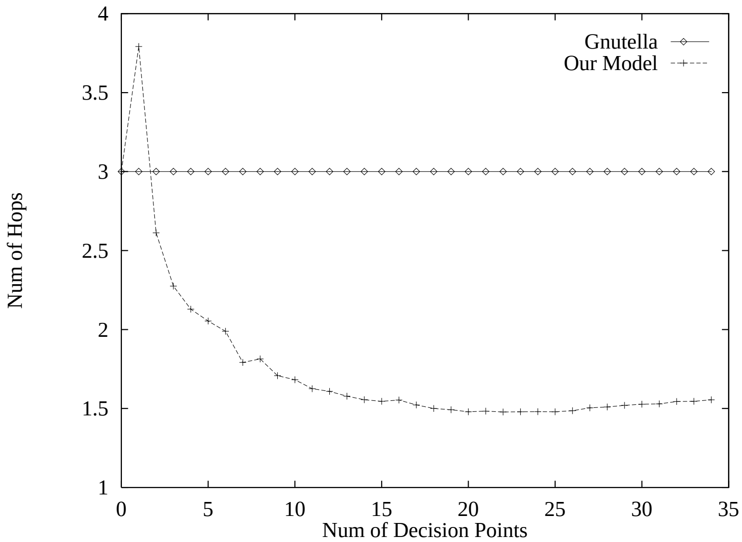

Figure 8: Mean number of hops required for a query-response message to return to the searcher.

Instead, some time, showing consistent new behavior is required before changes are made. Figure 10 depicts the number of message hops required to reach one of the type-C peers from the test peer. We see that the distance changes (both nearer and further) as the test peer's interest changes, but is not in lock-step with the change in query type being generated. The gradual movement of the peers is an important feature of our algorithm, as it tries to reduce the instability factor.

Figure 11 further demonstrates the adaptability of the algorithm. We can clearly see that as interests change, the number of successful queries drops dramatically. But, as the peer creates new connections to replace old one, the success rate increases. Again, there is lag in the time until the rate increases, but this is an intentional result so as to increase stability in the case where interest changes are brief.

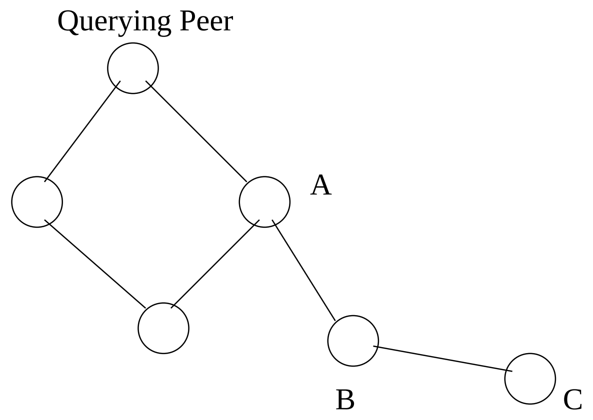

Figure 9: Scenario 2: Initial Topology.

Other Observations:

- Our algorithm can result in a disconnected graph. If a node does not provide information of interest, all of its peers will disconnect from it because it is not contributing to the success of their searches. This may be considered an advantage or disadvantage of the algorithm. Disconnection seems to be a bad situation, but as noted in [1], we are threatened by a "digital tragedy of the commons" in which people are inclined to take from the network but provide nothing. In this case, only by providing useful information will a peer be able to remain an active participant. So, it is unclear whether disconnection is a problem that needs a solution.

- We observe a phenomenon of a "local maximum." That is, after running the algorithm the peer becomes content with its situation, even if better situations may be possible. This is, in part, due to our calculation of Importance favorably weighting long running connections. We make this decision to improve the overall stability of the system, but it is a potential problem. More experiments with the weighting factor may help to find a good compromise.

5 Related Work

The current distributed searching mechanisms in the peer-to-peer network [10] use a brute force algorithm and broadcast the search request to all the peers. The problem with this approach is that there is no mechanism to determine which peers are likely to have results to propagate the requests to these peers only, and therefore send a large number of messages and consume many processing and communication resources. Among these search mechanisms, Limewire [13] defines interest groups, called Channels. This is similar to our approach, however, once the connection between the peers are established, there is no automatic way to change the topology of the network as the interests of the peers change over time.

The behavior and performance of hybrid peer-to-peer networks have also been studied in the literature. In hybrid peer-to-peer networks such as Napster [11] and Morpheus [14] centralized

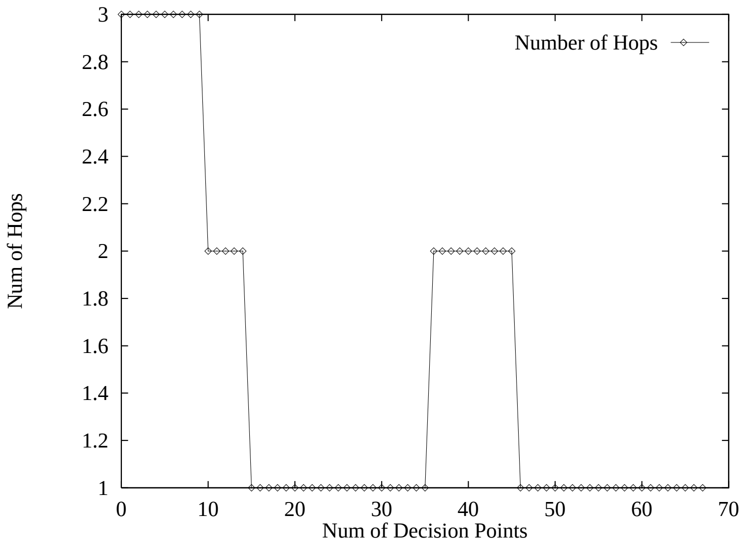

Figure 10: Distance of type-C peer from test peer

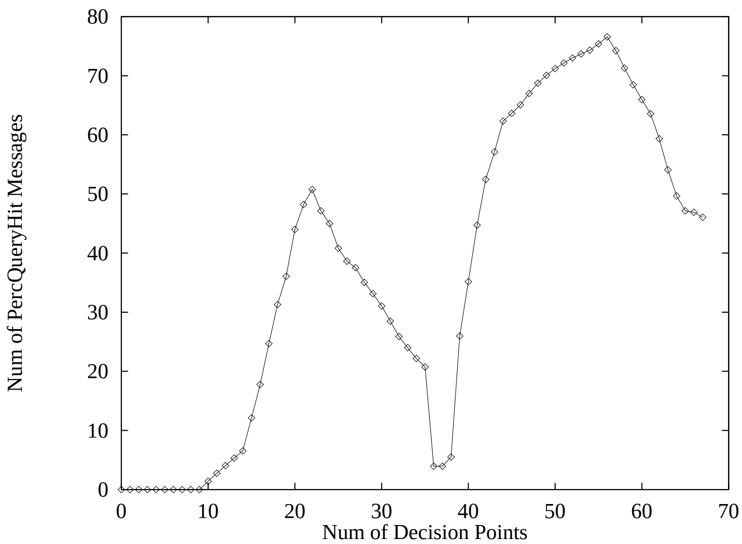

Figure 11: Percentage of queries returning hits

servers maintain a global index of the peers and the files in the network. Nodes search for files by sending search requests to the centralized servers and obtain the list of files and the corresponding peers. The problem of this scheme is that it is a bottleneck in terms of scalability and reliability. Furthermore, there is no mechanism to ensure that different servers have an always updated view of the files of their peers. Yang *et al* [2] have proved that chained architectures are the best strategy for today's file sharing systems, but have poor performance when the user interests are diverse.

Recent work [6] has studied location and routing mechanisms in large-scale peer-to-peer systems. Stoica *et al* [3] use a consistent hashing scheme to distribute the objects in the peer-to-peer network, so that an efficient location algorithm can be implemented. Tewati *et al* [9] propose a scalable, high-performance architecture for finding data. Similar to our approach, their objective is to minimize the number of hops to locate and access data in the network. Zhuang *et al* [5] have built an application level multicast system that incurs minimal delay and bandwith penalties. Their primary focus is on handling faults in links and routing nodes. Rowstron *et al* [4] propose a scalable, distributed object location and routing algorithm for wide-area peer-to-peer applications to minimize the distance that the messages travel. The difference of the above approaches with our work is that they assume that the topology of the network is known in order to distribute the objects and route the messages to their destination.

Other work has exploited power-law link distributions in communication and social networks. Adamic *et al* [1] have shown that they can employ local search strategies that take advantage of the structure of power-law networks to efficiently search in a peer-to-peer model. The algorithm explores nodes with high connectivity. In a similar approach, Adjih *et al* [15] have examined multicast trees and power law networks and discuss the number of nodes that receive any message sent through multicast. Their goal is to measure and potentially reduce the total network traffic required to perform a multicast. Both of these approaches, however, assume a fixed network topology based on the physical layout of the network. Our model has the advantage of an easily reconfigurable network which permits us to change topology to improve performance.

In the area of distributing storage, Akamai [12] is building an intelligent global distributed network of servers located at the edges of the Internet. Their aim is to improve content delivery and provide content personalization based on the user's line speed, geographic location and device type. Rost *et al* [8] propose a new technique for efficiently delivering popular content from information repositories with bounded file caches. Their technique uses fast erasure codes to generate encodings of popular files, of which only a small sliding window is cached at any time instant. Their approach maximizes sharing of state across different request threads and minimizing cache memory utilization. Doyle *et al* [7] explore the effects of caches on the properties of the request traffic. They focus on Web requests that follow a Zipf-like object popularity distribution and examine the impact on load distribution strategies and content cache effectiveness for servers.

6 Conclusions

Decentralized peer-to-peer networks appear to be an attractive means for distributing information on the Internet. The advantages in terms of reliability and scale are compelling. Unfortunately, naive schemes for organizing such a network quickly break down, making these seeming advantages into weaknesses. We have attempted to overcome some of these by introducing the notion of interests into the network, and to organize the network based on interest. This permits us to alter the topology of the network to form clusters with similar interests. This, in turn, permits us to improve the overall performance of the system by limiting the resources required for searching in the network. The algorithm has been defined and tested in the context of a specific application, the decentralized on-line newspaper, which is built on a standard peer-to-peer sharing system. In our limited testing environment, the algorithm has shown itself to be both effective and stable in creating the clusters we desire.

We need to further validate the work through improved testing. Testing of these sorts of systems turns out to be problematic, however. In particular, we are concerned with scale, but running a controlled test to validate scalability is difficult. We see two approaches to solving this. First is simulation. We are beginning to investigate the alternatives for simulating a network of this sort, but must still find relevant parameters to run the simulation. The second is a deployment. Because we have built our system in the context of an existing peer-to-peer network, and have live implementations of the algorithm, we expect that the most valuable results could come from making the system available. In doing so, we intend to instrument the system so that large scale measurements can be made of the effectiveness of the system. The challenge then becomes to make the application compelling enough to encourage a large number of users to participate. We hope that the newspaper-style information sharing application will be.

References

[1] L. A. Adamic, R. M. Lukose, A. R. Puniyani, B. A. Huberman, "Search in power-law networks", *http://www.parc.xerox.com/istl/groups/iea/papers/plsearch/*

[2] B. Yang and H. Garcia-Molina, "Comparing Hybrid Peer-to-Peer Systems", *Proceedings of Very Large Databases*, Rome, Italy (September 2001).

[3] I. Stoica, R. Morris, D. Karger, M. Frans Kaashoek and H. Balakrishnan, "Chord: A scalable peer-to-peer lookup service for Internet applications", *ACM SIGCOMM*, San Diego, CA (August 2001).

[4] A. Rowstron and P. Druschel, "Pastry: Scalable, decentralized object location and routing for large-scale peer-to-peer systems", *Proceedings of the 18th IFIP/ACM International Conference on Distributed Systems Platforms*, Heidelberg, Germany (November 2001).

[5] S. Q. Zhuang, B. Y. Zhao, A. D. Joseph, R. Katz and J. Kubiatowicz "Bayeux: An Architecture for Scalable and Fault-tolerant Wide-area Data Dissemination", *Proceedings of the Eleventh International Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV 2001)*, Port Jefferson (June 2001), pp. 11-20.

[6] Y. D. Chawathe, "Scattercast: An architecture for Internet broadcast distribution as an infrastructure service", Ph.D. Dissertation, *http://yatin.chawathe.com/thesis/*.

[7] Ronald P. Doyle, Jeffrey S. Chase, Syam Gadde and Amin M. Vahdat, "The Trickle-Down Effect: Web caching and server request distribution", *Proceedings of the Sixth International Workshop on Web Caching and Content Distribution*, Boston, MA (June 2001).

[8] S. Rost, J. Byers and A. Bestavros, "The Cyclone server architecture: streamlining delivery of popular content," *Proceedings of the Sixth International Workshop on Web Caching and Content Delivery*, Boston, MA (June 2001).

[9] R. Tewari, M. Dahlin, H. Vin, and J. Kay, "Design Considerations for Distributed Caching on the Internet" *Proceedings of the 19th IEEE International Conference on Distributed Computing Systems*, Austin, TX (May 1999), pp. 273-284.

[10] The Gnutella Homepage, http://www.gnutella.wego.com/

[11] The Napster Homepage, http://www.napster.com/

[12] The Akamai Homepage, http://www.akamai.com

[13] The Limewire Homepage, http://www.limewire.com

[14] The Morpheus Homepage, http://www.musiccity.com

[15] C. Adjih, P. Jacquet, L. Georgiadis, W. Szpankowski "Multicast tree structure and the Power Law", *http://www.cs.purdue.edu/homes/spa/papers/multicast.ps*