Using Locality of Reference to Improve Performance of Peer-to-Peer Applications [*]

Marcelo Werneck Barbosa Melissa Morgado Costa
Jussara M. Almeida Virgílio A. F. Almeida

Department of Computer Science
Federal University of Minas Gerais
Av. Antonio Carlos 6627 - 31270-010
+55 31 3499-5860 - Belo Horizonte MG - Brazil
{mwb, melissa, jussara, virgilio}@dcc.ufmg.br

ABSTRACT

Peer-to-peer, or simply P2P, systems have recently emerged as a popular paradigm for building distributed applications. One key aspect of the P2P system design is the mechanism used for content location. A number of different approaches are currently in use. In particular, the location algorithm used in Gnutella, a popular and extensively analyzed P2P file sharing application, is based on flooding of messages in the network, which results in significant processing overhead on the participant nodes and thus, poor performance.

In this paper, we provide an extensive performance evaluation of alternative algorithms for content location and retrieval in P2P systems, in particular, the Freenet and Gnutella systems. We compare the original Freenet and Gnutella algorithms, a previously proposed interest-based algorithm and two new algorithms which also explore locality of interest among peers to efficiently allow content location. Unlike previous proposals, the new algorithms organize the peers into communities that share interests. Two peers are said to have common interest if they share some of the locally stored files.

In order to evaluate the performance of these algorithms, we use a previously developed Freenet simulator and build a new Gnutella simulator, which includes several realistic system characteristics. We show that the new community-based algorithms improve the original Gnutella content location latency (and thus the system QoS) and system load by up to 31% and 30%, respectively. Our algorithms also reduce the average Freenet request and response path lengths by up to 39% and 31%, respectively. Furthermore, we show that, compared to the previously proposed interest-based algorithm, our new algorithms improve query latency by up to 27% without a significant increase in the load.

[*]Jussara M. Almeida and Virgilio A. F. Almeida are supported by CNPq/Brazil.

Keywords

Peer-to-peer systems, content location algorithms, interest-based communities, performance analysis

1. INTRODUCTION

Services on the Internet are evolving from centralized client -server to fully distributed architectures. A new descentralized approach where end-hosts, called peers [23], interact as equals sharing their computing resources (CPU, memory, disks) is emerging and becoming one of the most popular service architectures in today's Internet. Such systems are called peer-to-peer (or simply, P2P) systems.

P2P systems are attractive for several reasons. First, they present good scalability with the number of simultaneous users. Second, the barriers to starting and growing such systems can be considered low, since they usually do not require any special administrative or financial arrangements. Third, P2P systems offer a way to aggregate and make use of the tremendous computation and storage resources on computers across the Internet. Finally, the descentralized and distributed nature of P2P systems gives them the potential to be robust to faults or intentional attacks, making them ideal for long-term storage as well as for lengthy computations. A number of different applications that benefit from such characteristics has been proposed and deployed in the Internet, including Napster [31], Gnutella [27], an extensively analyzed file sharing application [21, 15], and Freenet [5]. Recently, more sophisticated systems such as Kazaa [29] and Morpheus [30] have also been developed.

One of the main challenges in P2P research is the content location or searching problem. How do you find a data item in a large P2P system in a scalable manner? This problem is at the heart of the design of any P2P system. A solution for it should provide efficient mechanisms for locating peers that have copies of the requested data and selecting one of those peers for retrieving the data. The searching problem is a major issue in the P2P area mainly because the effectiveness of a P2P network strongly depends on the versatility and scalability of its content location mechanism [6].

The best searching technique for a given system depends on the needs of the application. For storage of archival systems focusing on availability, techniques such as Chord [8], Pastry[20], Tapestry[34] and CAN[18] are well suited, because, by tightly controlling the data placement and logical topology within the network, they guarantee location of content, if it exists, within a bounded number of hops. In systems where persistence and availability are neither guaranteed nor necessary, such as Gnutella, Freenet, Napster and Morpheus, searching techniques can afford to have looser guarantees. However, despite the intense research activity in the area [21, 33], current searching techniques in such "loose" systems are mostly inefficient either because they generate too much load by flooding the network with messages or because they might result in poor response time [33].

This paper provides an extensive performance evaluation of three classes of algorithms for content location and retrieval in P2P systems, in particular, Freenet and Gnutella. The first class includes the original algorithms used in these two systems. The second class includes a previously proposed algorithm which explores common interests among Gnutella peers to improve content location [24]. This algorithm, described in detail in Section 3.2, assigns to each peer a list of shortcuts, which are pointers to other peers that have recently responded to previously sent queries and thus, are believed to be more likely to answer future queries. Throughout this paper, we refer to this algorithm as the interest-based algorithm that uses shortcuts or simply, the shortcut-based algorithm. The third class of content location algorithms evaluated in this paper includes two new algorithms, which further explore the impact of using common interest among peers to guide the searching mechanism.

Previous research in different areas, such as biological and social sciences and the World-Wide Web has proved that the identification of interest related communities among entities (e.g., web pages) is possible [32]. Our new content location algorithms, motivated by these studies, group peers into interest-based communities to efficiently allow content location and retrieval. Unlike the shortcut-based algorithm [24], which define two nodes to share interests if they responded to queries previously sent by each other, we consider there is common interest between two nodes when they share locally stored files. Two methods are proposed for creating peer communities. One, inspired from previous web community algorithms [9, 10], maps the problem into a maximum network flow problem [2]. The other, simpler and more efficient, groups together those peers that share more files.

In order to evaluate the performance of these three classes of content location algorithms, we use the publicly available Freenet Aurora simulator [25] and develop our own Gnutella simulator, since we were not aware of any publicly available simulator of this system (the one used in [24] was neither available nor clearly described). Unlike Aurora, our Gnutella simulator includes several realistic system characteristics which have been reported in previous analyses of the system [21, 17]. We evaluate the performance of the content location algorithms using a number of different metrics such as query latency, load on the system and query scope. We found that our algorithms decrease the average query latency time (and thus, improve the QoS) and the load in the original Gnutella by up to 31% and 30%, respectively. The average request and response path lengths in Freenet are reduced by up to 39% and 31%, respectively. Our new community-based algorithm improves the average Gnutella query latency achieved with the shortcut-based algorithm [24] by up to 27%, without significantly increasing the load on the system.

The key contributions of this paper are:

- An extensive performance evaluation of alternative content location and retrieval algorithms for P2P systems, in particular for Freenet and Gnutella, using a variety of meaningful performance metrics. We evalute the original Gnutella and Freenet algorithms, an interest-based algorithm that uses shortcuts previously proposed for Gnutella [24] and two new algorithms.

- Two new interest-based algorithms which group peers into communities to improve the performance of the content location mechanisms in Gnutella and Freenet.

- The design and implementation of a Gnutella simulator, which models different aspects of the system behavior. The selection of the models and distributions used in our simulator are guided by the results reported in previous characterizations of P2P systems [21, 17] Thus, our simulator provides a more realistic environment for experimenting with P2P systems (in particular, the Gnutella architecture) than other simulators available in the literature [25].

This paper provides a performance comparison between two well-known search techniques that have already been applied in other research areas, but not in P2P networks at the same time (search through flooding and shared interests). As our results will show, the search based on interests in peer-to-peer networks yielded a considerable better performance than flooding. We consider that the development of techniques based in interests might be useful to design high performance software in other computer-related fields.

The remaining of this paper is organized as follows. Section 2 presents the related work. Section 3 provides a detailed description of the content location algorithms explored in this paper. The main characteristics of the simulators, in particular of our new Gnutella simulator, as well as the performance metrics used to evaluate the algorithms are presented in Section 4. Section 5 presents the main performance results for the algorithms in Freenet and Gnutella. Our conclusions and future work are offered in Section 6.

2. RELATED WORK

One can categorize the content location algorithms used in popular P2P systems into three groups. The earliest design, used in Napster [31], is based on a central server (or cluster of servers) which maintains an index with the location of all shared files available in the system. The two other approaches are based on the distribution of the indices to the shared files among all participant peers. These approaches differ as to whether the content location mechanism relies on any special structuring of the peers.

Unstructured content location is used in systems such as Gnutella [27]. Gnutella relies on flooding of messages. Peers organize themselves into an overlay. In order to find content, a peer sends a query to all its neighbors on the overlay, which, in turn, forward the query to all of their neighbors and so on, until the query time-to-live has expired. While this solution is simple and robust, it does not scale well. A more detailed description of the Gnutella content location algorithm is provided in Section 3.1.2.

Freenet [26] permits the publication, replication and retrieval of files while protecting the anonymity of both authors and readers. To do so, files are named by location-independent keys. Each Freenet node maintains its own local datastore and a dynamic routing table containing the addresses of other nodes and keys to the files they are thought to hold. These two data structures are used by the Freenet content location algorithm, as will be described in detail in Section 3.1.1. Freenet is considered a system with "loose structure" because the placement of files is based on hints [14]. In systems with "tight structure", the structure of the P2P network and the placement of files are extremely precise and so subsequent queries can be satisfied very efficiently [14]. Examples of such systems are the so-called *lookup systems*.

Lookup protocols based on the Distributed Hash Table (DHT) abstraction [18, 8, 34, 20] have been proposed to address the scalability problem inherent to flooding. In these protocols, peers organize themselves into a well-defined structure that is used for routing queries and storing or indexing pre-defined data items. DHTs require that peers store data for the "common good", incur larger overhead than unstructured architectures in the presence of peer failure or disconnection and inherently, cannot efficiently support partial-match queries [6].

The work described in [11] proposes a simple analytical model for exploring the performance tradeoffs of these three design approaches. Key conclusions are the limited scalability of flooding-based algorithms and the relatively small impact of freeloaders on the performance of other peers.

Motivated by the scalability problems inherent to Gnutella networks, the authors of [14] proposed a content location algorithm for such networks based on multiple random walks that resolves queries almost as quickly as flooding while considerably reducing the network traffic. Instead of resorting to flooding, in order to search for a file, a node chooses a neighbor randomly and sends the query to it. The neighbor in turn chooses one of its neighbors randomly and forwards the query. This process goes on until the file is found or the query terminates according to a pre-defined TTL. The simulation results reported demonstrate a considerable improvement in the network load, however, at the expense of an increase in the user-perceived delay of successful searches. In spite of having the same goals, our algorithms differ from this one in the sense that ours explore the existence of communities of interests to improve the performance of the network while the one described in [14] are not based on interest-based relationships. A performance comparison between these algorithms is subject for future work.

Recently, Sripanidkulchai *et al* [24] proposed a new content location algorithm that explores common interests among peers. Their design philosophy departs from existing work in that they seek to retain the simple, robust and fully decentralized nature of Gnutella, while improving its scalability. The algorithm is based on the principle that peers exhibit interest-based locality. Peers that share similar interests create "shortcuts" to one another and use them to locate content. When shortcuts fail, peers resort to using the underlying Gnutella protocol [24]. This algorithm, which is referred to, in this text, as the shortcut-based algorithm is further described in Section 3.2.

The identification of interest-based communities has been applied to a number of different areas, including the web. The authors of [9, 10] define a community on the web as a set of sites that have more hyperlinks (in either direction) to members of the community than to non-members. The problem of identifying a web community is NP-complete [10]. However, by assuming the existence of one or more seed web sites and exploring systematic regularities of the web graph, the authors in [9, 10] showed that the problem can be mapped into the s-t maximum flow network problem [2], which has many efficient polynomial time solutions. One of the searching algorithms proposed in this paper is inspired from this approach.

Like the shortcut-based algorithm described in [24], our new algorithms also explore the common interests among peers to improve the scalability of the content location mechanisms in Gnutella and Freenet systems. However, they differ from the previous approach in how common-interest among peers is assessed, as will be clear in Section 3.3.1. By exploring locality of interest in a different way, we expect to reach more significant improvements in the Gnutella and Freenet performance.

3. CONTENT LOCATION ALGORITHMS

This section describes the original content location algorithms used in Freenet (Section 3.1.1) and Gnutella (Section 3.1.2). These algorithms serve as a baseline of comparison for two different classes of content location algorithms that explore common interests among peers: the algorithm that uses shortcuts, proposed in [24] and described in Section 3.2 and our new algorithms that are based on the identification of communities of interests and described in Section 3.3.

3.1 Traditional Algorithms

3.1.1 Content Location in Freenet

The basic approach for locating content in Freenet is that queries are passed along from node to node in a chain of proxy requests, with each node making a local routing decision about where to send the query next. Each query is given a hops-to-live count, which is decremented at each node. The process continues until the query is either satisfied or exceeds its hops to live limit [26].

In order to retrieve a file, a user first obtains or calculates its binary file key. She then sends a request to her own node specifying the key and a hops-to-live value. The node checks whether the requested data is stored locally, in which case, the data is returned to the user with a note saying the local node is the source of the data. Otherwise, it looks up in its routing table for the nearest key (the one with the closest identifier) to the requested key and forwards the request to the corresponding node. This process is repeated in a chain of forwarding requests until the file is found or the hops-to-live, decremented at each node, expires. In either case, a chain of responses is sent back in the reverse path, starting from the node where the file was found or the hops-to-live expired. In the first case, each node visited by the response stores a local copy of the file, creates a new entry in its routing table associating the data source with the requested key and forwards the response back in the chain, changing first the data source to be itself. If a node cannot forward a request to its preferred downstream node, it tries the node having the next nearest key. If it runs out of candidates to try, it reports failure back to its upstream neighbor, which will then try its next best choice [5, 4, 13].

Note that every time a node forwards a successful response, a "copy" of the requested file is stored locally. As time goes by, nodes should become specialized in locating sets of similar keys and storing clusters of files having similar keys. If a node is listed in routing tables under a particular key, it will tend to receive mostly requests for similar keys [5, 4]. Considering this, the quality of the routing mechanism should improve over time, specially if we consider a stable topology.

3.1.2 Content Location in Gnutella

Gnutella peers form an application-level overlay on top of the physical network and interact with each other exchanging messages. There are 3 different types of messages: (a) *ping* (and corresponding *pong*) messages for checking whether a node is still participating in the overlay, (b) *query* (and *query response* messages for querying for specific files and (c) *get/push* messages for retrieving content from a node. The *query* messages, exchanged as part of the content location algorithm, are of special interest.

In order to locate a file, a peer sends a query to all of its neighbors in the overlay network. Upon receiving a query, a peer checks if any of the locally stored files match the query. If so, the peer sends a query response back to the query originator. Whether or not a file match is found, the peer continues to flood the query over the overlay network as long as a pre-defined TTL (time-to-live) value, assigned to every query and decremented at each visited node, is greater than zero [21]. This broadcast method will find the target file very quickly, given that it is located within a radius equal to the TTL value. However, this searching strategy does not scale well [1] because of the bandwidth consumed by broadcast messages and the computing cycles consumed by the nodes that must handle these messages [22].

In order to scale Gnutella to more than hundreds of users, it is imperative to stop the flooding of query and reply messages [22]. A number of previous studies has explored new mechanisms to replace the flooding [24, 22, 33]. The searching algorithms based on communities of interests proposed in this paper represent a new attempt towards this goal.

3.2 Shortcut-Based Algorithm

The interest-based algorithm using shortcuts, proposed in [24] for the Gnutella network, is based on the following principle: if a peer was able to successfully respond to a query sent by another peer, it is likely that it will also be able to successfully respond to future queries sent by the same peer. The authors propose a self-organizing protocol that efficiently explores this property for content location. Peers that share similar interests create *shortcuts* to one another and use them to locate content [24].

A peer's first attempt to locate any content is executed through the Gnutella original protocol, i.e. flooding, since it does not have any information about other peers' interests. This process returns a set of peers where the requested content is stored. A number of these peers is randomly selected and added to the "shortcut" list of the requesting node. Subsequent queries for content go first through this list, contacting each peer in the list, one at a time, based on a pre-defined ranking value. Peers in the shortcut list are ranked based on their perceived "utility", which can be estimated by the probability of providing content, the path latency, load, amount of stored content or any combination of these indices. In [24], the perceived utility of a shortcut is defined as the number of times it was useful in the past. If a peer cannot find content through the shortcut list, it issues a lookup through Gnutella, and repeats the process for adding new shortcuts.

We have implemented this shortcut-based algorithm in order to evaluate whether a different mechanism for exploring interest based relations among peers is more effective in improving the performance of Gnutella. We implemented the basic algorithm proposed in [24], where only one shortcut peer is added at a time. We did not vary this parameter because it was shown to have little impact on the system performance [24]. Like in [24], we assume each shortcut list has a maximum size of 10 elements and is managed by a maximum perceived utility policy, i.e., whenever its maximum size is reached, the shortcut with the lower perceived utility is removed to free space for a new shortcut.

3.3 New Algorithms Using Communities of Interests

We propose two distributed algorithms for peers to self-organize into clusters, or peer communities, based on interests. Conceptually, our algorithms can be applied to any P2P network. However, in this paper, we consider only Gnutella and Freenet, given the severe performance problems of Gnutella [22, 11], the availability of a previously developed Freenet simulator [25], and the essential differences between the content location protocols of these two systems.

Our two algorithms differ only on how peer communities are created. Section 3.3.1 describes the common aspects of the new algorithms, i.e., the searching protocol, and explores the similarities and differences between our algorithms and the shortcut-based one that uses shortcuts [24], described in the previous section. Section 3.3.2 introduces the two algorithms used to create the interest-based community for a given peer.

3.3.1 Content Location Protocol

This section describes the overall behavior of our new content location mechanism. We start by contrasting it against the previously proposed algorithm that uses shortcuts, emphasizing their differences. We next discuss the searching protocol in detail.

Our new algorithms and the algorithm that uses shortcuts [24] are based on the same principle of locality of interest, or, in other words, that there is a greater chance of finding a file one peer is looking for in another peer that shares interests with it. However, the key difference between them is how common interest is evaluated. In the shortcut-based algorithm, the assessment of whether a peer X has common interests with a peer Y is based on historical data, i.e., on whether peer X was able to respond to previous queries sent by peer Y. The principle behind our algorithms is that the "current" content stored at a peer reflects its current interest. Thus, common interest between two peers is assessed directly from the current content locally stored at both peers and is periodically re-evaluated.

Each peer maintains a community of peers which share similar interests, i.e., a number of files. Like in [24], peers in the community can be ranked based on different metrics such as number of shared files, available bandwidth or amount of stored content. In this paper, we use the number of shared files as the ranking metric. The searching algorithm works by first contacting the peers in the community, k peers at a time, where k may vary from 1 up to the size of the community, starting with the top ranked peers. (We evaluate the impact of k on the system performance in Section 5.2.1.) Like in [24], in case of no successful response, our algorithms resort to the underlying protocol of the network where it is implemented (in this paper, Freenet or Gnutella). Also like in [24], we assign a maximum size to each community (10, in our experiments) and manage each community list with a maximum interest policy, i.e., whenever space is needed, the peer that shares the smallest number of files is removed from the list.

Determining common interest by comparing the content of each pair of peers in the network is potentially very expensive and impractical. Thus, in order to limit the overhead of our algorithms, we have constrained the number of peers contacted and files compared by each peer to small values and we have set the frequency at which the community creation algorithm in use is run by each peer as a function of the amount of new local content. In other words, the algorithm is run only after a certain fraction (20%, in our experiments) of the content has changed.

```
1: Start the community graph with the seed node at depth 0
2: For i = {0, 1} :
       For each node p at depth i :
           Randomly select a subset of p´s local files and a
             number of other peers known as participants
           Send queries to each selected node with the list
             of selected files
           For each peer q that responds with at least one
             shared file :
               Add q to the community graph at depth i + 1
               Add an edge between p e q with weight equal to
                 the number of shared files returned by q
3: Create an artificial sink and connect each node at depth
   two directly to the sink with weight equal to 1.
4: Calculate the maximum flow of this graph: the seed node
   is the origin and the artificial node the sink.
5: Insert the nodes found as solution of step 4 in the
   community list of the seed node.
```

Figure 1: Creating an Interest-Based Community of Peers: Basic Algorithm

3.3.2 *Creating an Interest-based Community of Peers*

We propose two algorithms for creating communities of peers based on interests. The first algorithm is inspired from the previous work on web communities [9, 10] and maps the problem into a maximum flow network problem [2]. The second is a result of a simplication in the first algorithm that led to a significantly different strategy.

The community of a peer can be viewed as a graph where the vertices are nodes and an edge exists between two nodes if they share interests, i.e., store files in common. Each edge is weighted by the number of files shared by the two nodes. The process of creating a community is very similar to the algorithm proposed in [9, 10]. The community starts with a single node, the seed, whose community will be identified. The first step is to find out nodes in the network that store files in common with the seed node. To do so, the seed randomly selects a subset of its locally stored files and sends queries to a number of nodes in the network as to whether they store any of those files. These nodes are chosen from a list of nodes which are known to be participating in the system. It comprises nodes that have sent pong messages in the past. In our experiments, we limit the number of selected files and number of contacted nodes to only 4 and 10, respectively. The nodes that share at least one of the files are added to the community graph, at depth one, i.e., directly connected to the seed node. The same process is repeated once again for each of the newly inserted nodes and a new group of nodes is added to the network, at depth two. Next, an artificial sink is created and connected to these nodes. A community is then identified in the graph by solving the maximum flow problem [2]. A summarized description of this algorithm, which will be referred to as the basic community algorithm, is shown in Figure 1.

Note that, in the second iteration of step 2, the algorithm adds to the community graph nodes that may not currently have files in common with the seed node but do have files in common with the nodes at depth one (those that share files with the seed node). The motivation for this step is that, despite not sharing currently any content, the seed node and a node at depth two may share files in the near future given their common relationship with at least one other node (at depth one). Thus, adding those nodes may improve the efficiency of the communities.

We also propose a different algorithm where only the nodes that are known to share files with the seed node (i.e., the nodes at depth one in the above algorithm) are added to the graph. In this algorithm, we do not need to solve the maximum flow network problem, but rather add to the community the nodes that share the largest number of files. Each time the algorithm is executed, a pre-defined number of nodes, given by the parameter n, are added to the community. In the next sections, we refer to the content location algorithm that uses this modified and simpler algorithm to create peer communities as the extended community algorithm.

In order to make the use of the basic community algorithm possible, some characteristics of P2P search protocols may need to be changed. Currently, there are indeed some different implementations of the Gnutella architecture that are considered small variations of the its specification. According to [3], for instance, some Gnutella clients allow their users' file lists to be retrieved, which is not a feature considered in the specification of the architecture. In a real implementation, the extra overhead of the algorithm must be taken into account. Recall that, as mentioned at the end of section 3.3.1, the algorithm is run only for those peers whose content has changed by some fraction. Thus, there is a trade-off between how often the algorithm is run and the extra overhead. If the algorithm is run very often, its overhead might end up degrading the overall system performance. On the other hand, if the algorithm is seldom run and a considerable amount of content of a peer has changed, its community of interest is not accurately representative of its interest. Either way, we expect the extra overhead not to be too significant because the number of nodes contacted and the number of files evaluated are small and thus both lists are composed of few elements.

4. EVALUATION METHODOLOGY

In this paper, we evaluate the performance of our new searching algorithms and the previously developed algorithms described in Section 3 by means of simulation. The performance improvement over the traditional Freenet searching algorithm introduced by the new protocols that take advantage of the interest-based peer communities is evaluated using the previously developed, open source Aurora simulator [25]. In order to evaluate the searching algorithms over the Gnutella network, we have built a new simulator, which models many realistic aspects of the system behavior. The main characteristics of both P2P system simulators are summarized in Section 4.1 and the performance metrics used to analyze the alternative algorithms are given in Section 4.2.

4.1 P2P System Simulators

4.1.1 The Freenet Simulator

Aurora simulates a very homogeneous Freenet network, in which every node has the same storage capacity and the selection of which host will perform the next operation on the network, which operation (i.e., query, insertion) will be performed and, in case of a query, which file will be requested or inserted are uniformly distributed over pre-defined parameter spaces. Heterogeneous, and thus more realistic, workloads as well as peer storage capacities are not taken into account. Furthermore, the participation of hosts is static; they are online throughout the simulation. Despite the simplicity of the model, Aurora has a quite complex implementation and little available documentation [25].

4.1.2 The Gnutella Simulator

Since we are not aware of any previously developed publicly available Gnutella simulator, we are faced with the task of building one of our own. The main goal in designing the new Gnutella simulator is to realistically reproduce as closely as possible several characteristics of the real system. In particular, we want to simulate a much more heterogeneous network than previous simulators (i.e., Aurora). In our simulator, the network is composed of a number of nodes which communicate with each other by exchanging messages as specified in the original Gnutella protocol [28] and/or the specific searching algorithm being implemented.

Proper evaluation of a P2P system must also take into account the peer characteristics. However, only a few papers have performed a detailed measurement study of P2P systems [21, 17, 12]. Furthermore, although some of these papers provide analyses of real data obtained from Gnutella users [21, 17], they do not clearly summarize the analyzed data into statistical distributions. Thus, in our simulator, we have modeled the system and peer characteristics for which data was available in [21, 17, 12] using distributions that closely approximate the reported data. Furthermore, we have also made some reasonable assumptions to model other Gnutella user characteristics. The key system aspects captured in our simulator are described next.

In our simulator, the Gnutella network is composed of a fixed number of nodes (1000 in our experiments) which are connected to each other in a way that attempts to model a small-world network, as proposed in [12]. Small-world graphs are characterized by a small average path length and a large clustering coefficient. In order to create a small-world network, we have first connected the nodes in a ring fashion and then added shortcuts between pairs of nodes, with a small probability p (0.05 in our experiments), as proposed in [12].

Apart from the average sizes presented in previous studies [21, 17], the distribution of file sizes in P2P systems has not been previously analyzed. We choose to model three classes of files corresponding to music, TV shows and 2-hour movies [17], popular content in P2P systems. The file sizes in each of these three classes are normally distributed with average (standard deviation) equal to 4.5(0.5) MB, 70(15) MB and 700(150) MB, respectively. For our experiments, the files are distributed across these classes with probabilities 0.7, for music, 0.15, for TV shows and 0.15 for movies.

The characteristics of each node, namely the storage capacity and available bandwidth, are also heterogeneous. Because we are aware of no previous study that thoroughly analyzed storage capacity of Gnutella peers, we have made the assumption that peers may have 1GB, 5GB and 10GB of available disk space with probabilities 0.2, 0.4 and 0.4, respectively. The distribution of available bandwidth is based on the results in [21]. We assume peers may have available bandwidth equal to 64 Kbps, 256 Kbps or 512 kbps with probabilities 0.1, 0.6 and 0.3, respectively.

Finally, the selection of which node will issue an operation at each step follows a Power law (i.e., Pareto distribution) recognizing that some nodes are more active than others. The selection of the file requested or inserted by a given node also follows a Pareto distribution, since it is common knowledge that the distribution of accesses in many Web services is skewed towards a few files.

Given the lack of a thorough characterization of Gnutella systems, we evaluated the correctness of our simulator by analyzing a few metrics for which we expect a certain behavior given the distribution models used. As examples, Figure 2 shows the distributions of the number of copies of files in the network, the popularity of the queried files and the number of files stored in each node. All three distributions exhibit the Power-law characteristic (i.e., a straight line in a log-log plot) reflecting the Pareto distributions used for node and file selection.

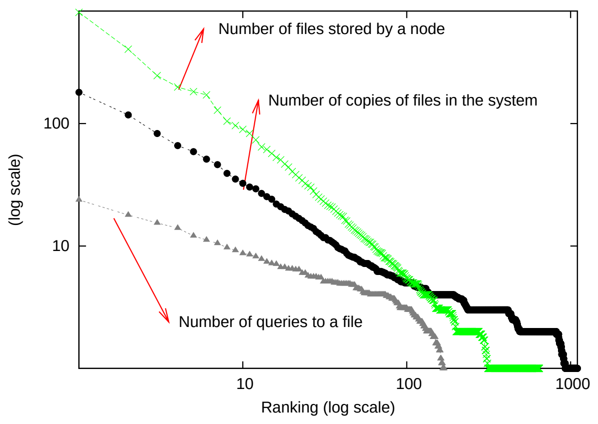

Figure 2: Distributions of the average number of file copies and file popularity, as a function of the file rank and distribution of the average number of files per node as a function of the node rank

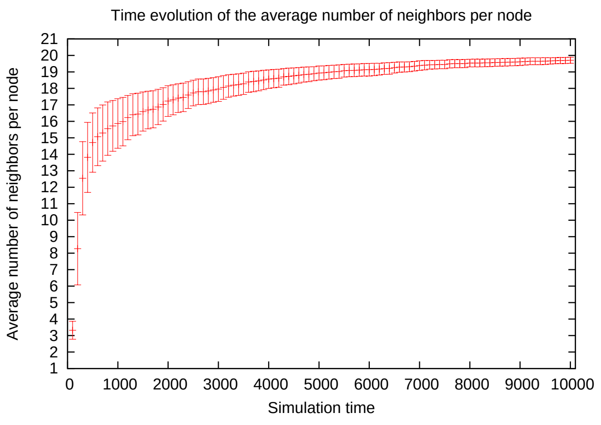

Figure 3: Average number of neighbors per node, as a function of the simulation time (maximum number of neighbors = 20)

We also evaluated the average number of neighbors per node as a function of time. Recall that a node periodically floods the network with *ping* messages and updates its neighbor list with the nodes that respond to those pings. We expect the average number of neighbors to increase rapidly in the beginning of the simulation as nodes become aware of a larger number of other peers as a result of the flooding. Figure 3 shows the time evolution of the average number of neighbors per node and its 90% confidence interval. Note that as the simulation progresses, the average number of neighbors increases rapidly, as expected, approaching the maximum, defined by a parameter, which is set as 20 in all experiments.

There are a number of enhancements that we plan to add to our simulator in the future. One key aspect that we intend to explore next is the dynamic behavior of peers, which can join and leave the network at unpredictable times. We will then be able to explore the impact of network growth and resilience to attacks.

4.2 Performance Metrics

We evaluate the performance of alternative algorithms for content location and retrieval using a set of different metrics. We distinguish between the metrics used to evaluate the algorithms designed for each system, Freenet and Gnutella, as they explore specific and relevant aspects of their behavior. The criteria used to analyze the algorithms for Freenet are:

- Request Path Length: number of hops traversed by a query from the requesting node until the node where the content is found.
- Response Path Length: number of hops traversed by a response between the node where a file is found and the node that requested it.

We choose these two metrics to evalute the performance of the content location algorithms for Freenet since, in this system, the searching procedure stops as soon as a copy of the file is found. Note that the query latency is a function of the sum of the request path length and the response path length. Furthermore note that the request path length tends to be considerably greater than the response path length since that metric includes failed attempts to find a file.

The performance of the Gnutella algorithms are evaluated mainly according to:

- System Load: number of queries processed by each node in the network.
- Query scope: the number of unique nodes that are visited by a query. Note that a query visits the same node multiples times. Thus query scope and system load are not necessarily complementary metrics.
- Query latency: the time a node waits for the download of a file to start.
- Success Rate: percentage of queries that are successfully responded.

We expect our searching algorithms to be very benefitial to Gnutella networks as they avoid the flooding of queries in the network and thus should decrease the average load in each node and the average query scope. Furthermore, we also expect a shorter average query latency, i.e., the average time taken to find a file, since it is more likely to find it in the community. We are particularly interested in determining how our algorithms perform compared to the interest-based algorithm that uses shortcuts. As will be shown in Section 5 the new method for evaluating common interests among peers results in significant improvement in the query latency and thus the QoS perceived by the user. Regarding Freenet performance, we expect our algorithms to improve both the average request path length and the average response path length, as we expect many queries to be responded through the communities.

5. EXPERIMENTAL RESULTS

This section presents the main performance results obtained for a number of alternative content location algorithms in Freenet (Section 5.1) and Gnutella (Section 5.2). All results reported in the following sections are averages of 10 experiments, in order to guarantee a standard deviation under 10% of the mean. The results are shown since the beginning of each simulation until the data converges and a more stable behavior of the system can be noticed. Hence, the results are reported in the form of snapshots (a certain moment during the simulation), average results of the whole simulation and data obtained at the end of the experiments.

We run the Freenet and Gnutella experiments for a network with 1000 nodes initially inter-connected in a ring fashion. As discussed in Section 4.1.2, the Gnutella Simulator attempts to create a small world network by adding connections among pairs of nodes, with a probability of 0.05. In addition to the workload and system characteristics modeled in Aurora and in our Gnutella simulator, described in Sections 4.1.1 and 4.1.2, a number of other parameters are used. For the experiments with Aurora, the simulation is run through 5000 units of time, the total number of files in the system is set to 5000, the maximum number of files a node can store is set to 50, the maximum size of each routing table is set to 200 entries and the maximum community size (used in our algorithms) is set to 20. For the experiments with Gnutella, the simulation is run through 10000 time units. We assume a total of 2000 unique files, a TTL for all messages equal to 7 hops (as commonly used [19]), a maximum number of neighbors per node equal to

20 and the maximum community size is set to 10. Previous works have measured the average number of neighbors in a Gnutella network as varying from 3 to 4 [32] and from 2 to 12 [7]. We also experimented with a maximum number of neighbors equal to 4. In this section, we show the results obtained with a maximum number of 20 neighbors allowed, which provide the most conservative performance improvement observed for the algorithms. However, we point out that even more significant improvements (roughly twice the percentual reduction in the load and query latency) can be obtained when the maximum number of neighbors allowed is set to 4.

We intend to perform the same experiments with different workloads in the future.

5.1 Results for Freenet

In this section, we present a performance evaluation of two content location algorithms for Freenet: the original and the basic community algorithms. We show that our new algorithm, which explores common interests among peers, reduces the average Freenet request and response path lengths by up to 39% and 31%, respectively, contributing to significantly improve the QoS observed by the user.

Figure 4 shows the cumulative distributions of the request path length, measured at two different timesteps in the simulation, for both algorithms. The distributions are calculated taking into account all the requests issued from the beginning of the simulation until the time when each measurement is made. Note that, as time progresses, requests must traverse fewer hops before a copy of the file is found. Furthermore, at both timesteps, the distribution is skewed towards fewer hops for the basic community algorithm, compared to the original Freenet algorithm. In other words, the use of the communities of interests helps finding the content faster (in fewer hops), as it was expected. As the simulation progresses, this gap increases. For instance, at timestep 1000, the average request path lengths are 152 and 141 for the original and community-based algorithms, respectively, i.e., a 7% decrease. At the end of the simulation, the average request path lengths are 67 and 40, a 39% reduction.

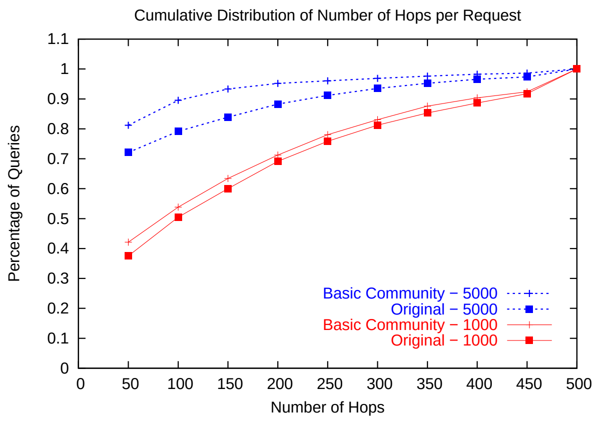

Figure 4: Cumulative Distribution of the Request Path Length at Timesteps 1000 and 5000 (Freenet)

The average request path length decreases faster for the community-based algorithm because the community lists increase, becoming more useful, as the simulation progresses. This is clear in Figure 5, which shows the distribution of the community sizes at the two timesteps. Soon after the beginning of the simulation (timestep 1000), most communities are composed of just one or two nodes and few communities are composed of more than four nodes. On the other hand, at the end of the simulation, the majority of the communities are composed of five or more nodes. Thus, as the simulation progresses, communities of interests become more representative of a group's interests and provide more benefits in the reduction of the average request path length.

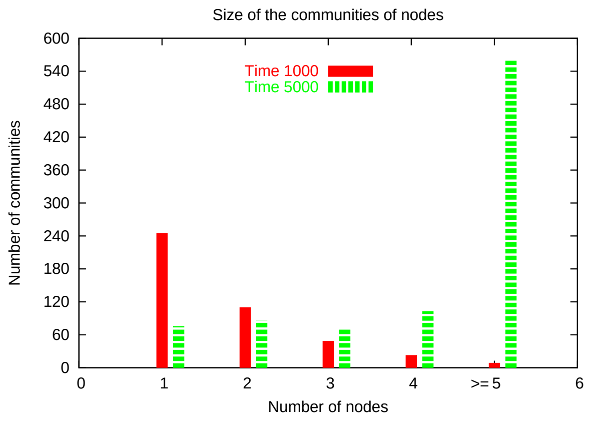

Figure 5: Size of the Communities at Timesteps 1000 and 5000 (Freenet)

We next analyze the response path lengths for the original and community-based content location algorithms. Figure 6 shows that, for both algorithms, the average response path length decreases with time, with a significant drop at the beginning. As discussed in Section 3.1.1, as time progresses, Freenet nodes tend to be clustered according to the key identifier of the files they store, becoming specialized in locating sets of similar keys [5]. Thus, not only the average request path length (as shown in Figure 4) but also the average response path length decreases with time. Furthermore, Figure 6 also shows that, like for the average request path length, the gap between the average response path length for the original and new algorithms increases as time progresses. By the end of the simulation, when the systems reach a stable state, the use of interest-based communities to guide content location improves the average response path length in 31%.

The data presented in this section provided a picture of the benefits of the use of the content location algorithm that relies on the identification of communities of interests in a Freenet network. The next section describes the use of such algorithm in Gnutella networks.

Experimenting with the extended community algorithm is subject for future work. However, given the performance improvements observed for Gnutella, as described in section 5.2.2, we expect an even more significant improvement in the results for Freenet networks shown in this section.

5.2 Results for Gnutella

This section presents the performance evaluation of alternative algorithms that explore interest-based locality for content location and retrieval in Gnutella. Section 5.2.1

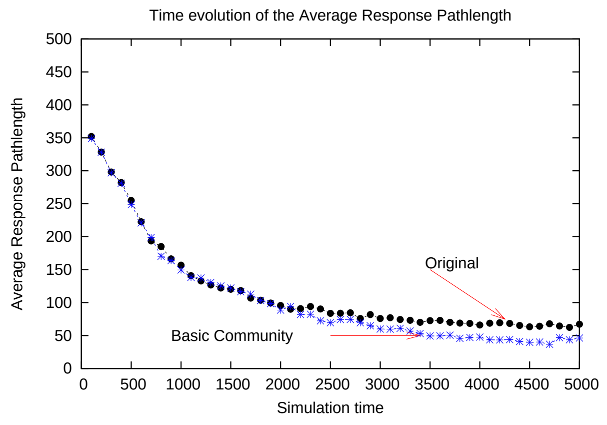

Figure 6: Time Evolution of the Average Response Path Length (Freenet)

compares the original, the previously proposed interest-based algorithm that uses shortcuts [24] and our new basic community algorithm which maps the problem of identifying a peer's community into the maximum flow network problem. Section 5.2.2 presents the results obtained with our extended community-based algorithm, which simply adds to a peer's community those nodes that share the largest number of files.

5.2.1 Basic Algorithms

In this section, we compare the performance of three alternative algorithms for content location in Gnutella: the original algorithm, the shortcut-based algorithm [24] and our new basic community algorithm, which establishes interest related communities among peers. We show that by sending queries first to the peers in a node's community, the basic algorithm contributes to a reduction on the original Gnutella average query latency and system load of up to 21% and 13%, respectively. Furthermore, compared to the shortcut-based algorithm, it provides a 31% shorter average latency with no significant load increase. We also point out that the results obtained with the shortcut-based algorithm are qualitatively comparable to the ones reported in [24], for the performance metrics used in both studies.

Recall that, in the original Gnutella protocol, each node that issues a query waits for responses for a pre-defined time interval. Only after this interval expires, can the download start. In our implementation, this waiting time is defined as twice the TTL, so that every node that receives the query is able to answer it in time if it stores the requested file. In the current implementation of our simulator, we assume that a message takes one timestep to traverse each network hop. Thus, the average query latency for the original Gnutella algorithm is fixed, equal to two times the TTL (i.e., 14 timesteps). The average query latency for the other two algorithms are shown in Table 1. The new basic algorithm that uses interest-related communities reduces the original Gnutella latency time in 21%. Compared to the shortcut-based algorithm, the improvement increases to 31%. We believe that the longer average query latency observed for the shortcut-based algorithm is due to the fact that some shortcut lists might include a relatively large number of peers, especially compared to the community lists, and many of those peers are not able to provide answers to future queries. We found that, at the end of the simulation, the average shortcut list was twice the size of the average community list. Recall that during the searching procedure, each shortcut in a peer's list is contacted, one at a time. Thus, if many peers do not respond successfully, latency increases. For the basic community algorithm, recall that the number of peers that are contacted at a time is a parameter of the algorithm. As will be clear at the end of this section, the average query latency does not vary significantly with k, the number of peers contacted at a time. Thus, we speculate that the additional nodes the shortcut algorithm is able to identify do not have a significant contribution as the query success rate is the same for both algorithms, as shown further. We point out that query latency was not evaluated in [24].

Time (timesteps)	Content Location Algorithms		
	Original	Shortcut	Basic Community
Latency	14	16	11
Download	416	418	438

Table 1: Average Latency and Download Times (Gnutella)

Table 1 also shows the average download time obtained for each algorithm. As it can be seen, the differences are not significant, as one might expect, since download time depends only on the size of the file and the available bandwidth of the nodes involved in the download. The fluctuations shown in Table 1 are probably due to the randomness of the probability distributions used in the simulator.

The load on the system, measured as the average number of query messages processed by each peer, is also an important performance metric for evaluating alternative content location algorithms, especially for Gnutella, given that its original flooding-based mechanism is a challenging problem that severely limits the system scalability. Figure 7 shows how the average load increases with time. We measure the average load at any given time t considering all queries issued from the beginning of the simulation until t. As time progresses, the community and shortcut lists help reducing the total load on the system as more queries are answered through these lists, without resorting to the flooding. At the end of the simulation, compared to the original algorithm, the basic community algorithm reduces the load by 12%, whereas the shortcut-based algorithm, compared to the new algorithm, provides only a slightly 6% improvement. This is a small improvement, especially given the significantly higher query latency observed for the shortcut-based algorithm.

A different metric that can also be used to assess how effective the interest-based algorithms were in avoiding the original flooding of messages is the query scope. The scope of a query is defined as the number of unique peers in the system that process a particular query. For example, flooding may have a scope of approximately 100% because all peers (except those beyond the TTL limit) process the query [24]. Figure 8 shows how the average query scope increases with time for the original, and basic community location algorithms. The query scope for the shorcut-based algorithm is similar but presents a slight negative slope as we reach the end of the simulation. The relative performance among the algorithms is the same as for the system load, shown in Figure 7. The new community based algorithm improves query scope by 13%, compared to the original algorithm.

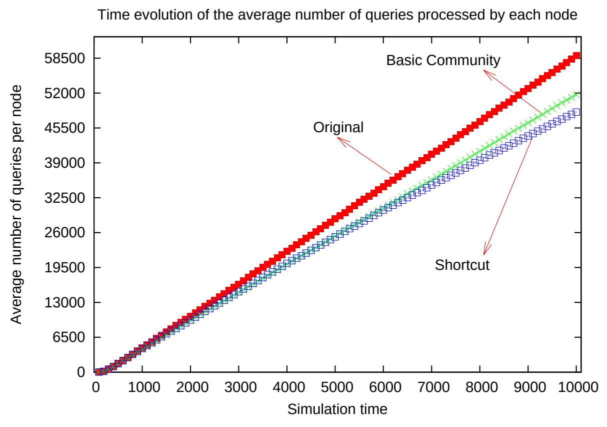

Figure 7: Average Load on the Nodes (Gnutella)

The shortcut-based algorithm provides only a marginal improvement (7% extra). Figure 8 also shows that the average scope increases rapidly. As times progresses and nodes become more connected, queries are sent to a larger number of nodes, increasing their scope.

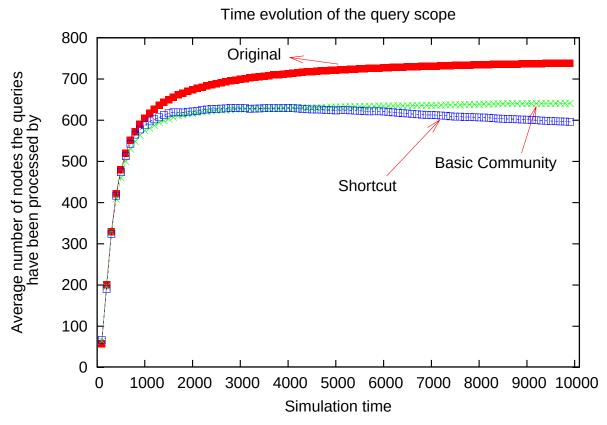

Figure 8: Average Query Scope (Gnutella)

Recall that a query is successful if the requested file is found before the query TTL expires. The percentage of successful queries, also referred to as the success rate, is also considered in our evaluation of Gnutella content location algorithms. Figure 9 shows how the cumulative success rate changes with time for the three algorithms. Note that the curves for the shortcut-based and the basic community algorithms overlap and that, compared to the original algorithm, they provide only a slightly higher query success rate. More interesting is how the query success rate increases with time, in Gnutella. At the beginning of the simulation only a few queries are successfully answered because most nodes have only a few files in local storage and they are poorly connected. As time evolves, more queries are issued, more file insertions are performed and nodes become more aware of the network participants, increasing their neighbor and community lists. As a consequence, more queries can be successfully responded.

In the new community-based algorithms, a node performs a request by first sending a query message to the nodes in

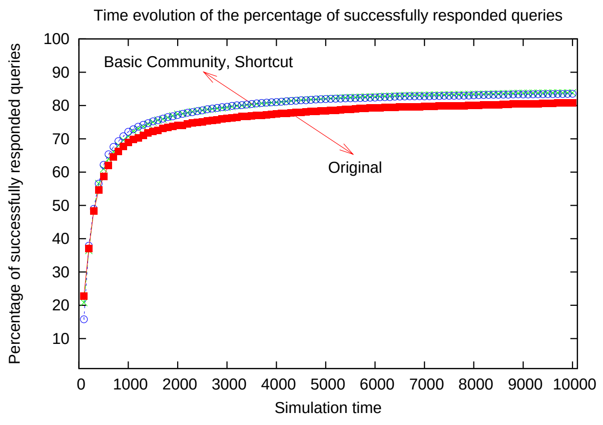

Figure 9: Query Success Rate (Gnutella)

its community list, k nodes at a time. The results shown so far were obtained with k=10, or, in practice, to all members of the community. Furthermore, also recall that, in the shortcut-based algorithm, the nodes in the shortcut list are contacted one at a time. In order to evaluate whether the value assigned to k was determinant to the relative performance of both algorithms, we ran a set of new experiments varying k from 1 to 10. We found little variation in the performance measures. Compared to the shortcut-based algorithm, the basic community algorithm with k=1 reduced the average query latency in up to 29% with a very small increase in the system load (6%).

5.2.2 *Extended Community Algorithm*

In this section, we extend the performance evaluation of community-based content location algorithms by comparing the basic and extended algorithms, described in Section 3.3. The extended community algorithm periodically adds to a peer's community list the n nodes that currently share the largest number of files with that peer. Thus, in order to evaluate the impact of n on the algorithm's performance, we experiment with values varying from 1 to 10, the maximum community size. Recall that the authors in [24] also experiment with a similar parameter used to specify the number of shortcuts that are added to a shortcut list at a time. Since the results in [24] show no significant impact of this parameter on the algorithm's performance, our experiments with the shortcut-based algorithm assume that one shortcut is added at a time. When we discuss the results of this algorithm in this section, we compare them with the results obtained for the extended community algorithm with n=1.

Our results show that the extended community algorithm is not only simpler than the basic one but also results in significant performance improvements in terms of average query latency, query scope and load, especially for values of n larger than 1. Thus, given the results shown in the previous section, the extended community algorithm reduces even further the query latency, scope and system load of the original Gnutella algorithm. Compared to the shortcut-based algorithm, the extended algorithm with n=1 provides an improvement of 27% in average query latency while keeping the same average load and query scope.

Table 2 shows the average latency and download times for the basic and extended community algorithms, with n=1, 5 and 10. Compared to the basic algorithm, the extended algorithm with n=1 results in an average latency that is only slightly longer. However, as n increases, more peers are added to a community list every time the algorithm is executed, more effective the community becomes in reflecting the peer's interests, improving the probability that future queries will be successfully responded through it. Note that, compared to n=5, adding 10 peers at a time (n=10) provides only a minor reduction in the average latency. Thus, most of the gains due to adding multiple peers to the community are reached with n=5. In this case, the extended algorithm improves the latency of the basic algorithm in 12%, which corresponds to a 31% improvement over the original Gnutella algorithm. With respect to the shortcut-based algorithm, the extended community algorithm with n=1 results in a reduction of 27% in the average query latency. As discussed in the previous section, this is due to the fact that some of the shortcut lists created are long and contain many shortcuts that are not useful for responding to future queries. As before, there are no significant variations in the average download times for the algorithms.

Time (timesteps)	Community-based Algorithms			
	Basic	Extended		
		n=1	n=5	n=10
Latency	11	11.5	9.7	9.6
Download	438	420	428	420

Table 2: Average Latency and Download Times (Community-based Algorithms, Gnutella

Figures 10 and 11 show the average load and query scope, respectively, for the basic community algorithm and for the extended community algorithm with n=1, 5 and 10. Both the load on the nodes and the scope of the queries decrease, significantly, as n increases. In particular, for n=10, the average load and query scope at the end of the simulation are 20% and 19% better, compared to the basic community algorithm. These numbers correspond to an improvement of up to 30% in both the average load and query scope observed for the original Gnutella protocol. Furthermore, even for n=5, the improvements over the original Gnutella protocol are still significant (25% for both metrics). Thus, these results show that, compared to the basic community algorithm, the simpler extended algorithm is more effective in reducing the overall system load and thus improving the scalability of Gnutella. Comparing the results in Figures 10 and 11 to those shown in Figures 7 and 8, one can see that the average load and query scope at the end of the simulation is practically the same for the extended community algorithm with n=1 and the shortcut-based algorithm.

We do not show the results for success rate because, as was the case for the comparison provided in the previous section, both the basic and the extended community algorithms (with n=1, 5 and 10) result in overlapping curves. Thus, we refer to Figure 9 which shows that the success rate is approximately 83% at the end of the simulation for the community algorithms as well as for the shortcut-based algorithm.

6. CONCLUSIONS AND FUTURE WORK

Peer-to-peer systems have become a very popular design approach for scalable distributed applications, especially file sharing applications. One key aspect of the P2P system design is the algorithm used for content location and retrieval.

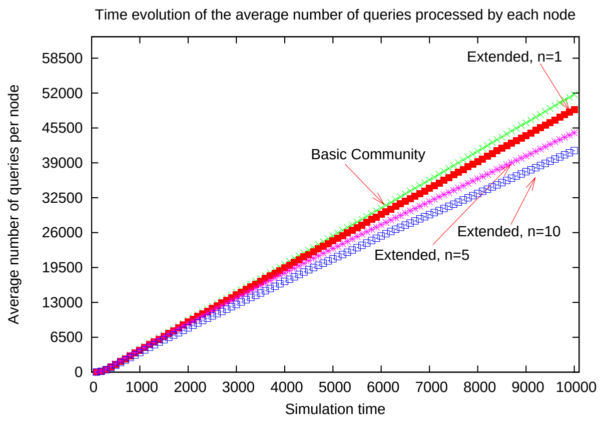

Figure 10: Average load per node. (Community-based Algorithms, Gnutella)

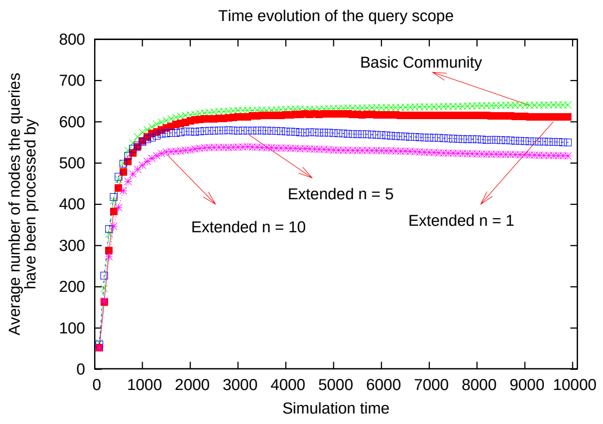

Figure 11: Average Query Scope. (Community-based Algorithms, Gnutella

A number of different approaches have been proposed [31, 27, 26, 18, 8, 34, 20, 24]. However, current search techniques used in popular P2P systems, such as Gnutella [27], for instance, tend to be very inefficient because they are based on flooding of messages and thus increase significantly the load on the system, compromising its scalability.

This paper provided an extensive performance evaluation of alternative algorithms for content location and retrieval in P2P systems, in particular, the Freenet and Gnutella systems. We compare three classes of algorithms: (1) the original Freenet and Gnutella algorithms, (2) a previously proposed interest-based algorithm that creates shortcuts among peers that were able to succesfully respond to each other previous queries [24] and (3) two new algorithms which also explore common interests among peers but unlike [24] assess common interest based on the content currently stored at each peer. In other words, the two new algorithms organize peers into communities that share interests, i.e., files and use the communities to guide the content location mechanism.

Our performance study shows that the community-based algorithms provide significant improvements over the original location mechanisms, in both Freenet and Gnutella. In particular, the extended community algorithm, the simplest one, provides the best improvements: up to 39% shorter average request path length in Freenet and up to 31% shorter query latency in Gnutella. Furthermore, compared to the shortcut-based algorithm, the extended algorithm provides a 27% improvement with respect to the average Gnutella query latency and generates approximately the same load on that system.

Possible directions to future work include: (a) extend our performance evaluation to other content location algorithms, such as the one described in [14], and possibly, other P2P systems, (b) explore other synthetic as well as trace-based workloads, such as the Boeing corporate proxy traces [16] and (c) evaluate the algorithms under dynamic conditions (i.e., peers joining and leaving the system at unpredictable times during the simulation)

7. REFERENCES

[1] L. A. Adamic, R. M. Lukose, A. R. Puniyani, and B. A. Huberman. Search in power-law networks. *Physical Review E*, 2001.

[2] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin. *Network Flows: Theory, Algorithms and Applications.* Prentice Hall, Englewood Cliffs, NJ, 1993.

[3] J. Chu, K. Labonte, and B. Levine. Availability and locality measurements of peer-to-peer file systems, 2002.

[4] I. Clarke, T. W. Hong, S. G. Miller, O. Sandberg, and B. Wiley. Protecting freedom of information online with freenet. for review purposes only.

[5] I. Clarke, O. Sandberg, B. Wiley, and T. W.Hong. Freenet: A distributed anonymous information storage and retrieval system. In *ICSI Workshop on Design Issues in Anonymity and Unobservability*, July 2000.

[6] E. Cohen, A. Fiat, and H. Kaplan. Associative search in peer to peer networks: Harnessing latent semantics. *Infocom*, 2003.

[7] F. Cornelli, E. Damiani, and S. D. Capitani. Choosing reputable servents in a p2p network, 2002.

[8] F. Dabek, E. Brunskill, M. F. Kaashoek, DavidKarger, R. Morris, I. Stoica, and H. Balakrishnan. Building peer-to-peer systems with chord, a distributed lookup service. In *8th IEEE Workshop on Hot Topics in Operating Systems*, pages 71–76, Elmau/Oberbayern, May 2001. MIT Laboratory for Computer Science.

[9] G. W. Flakes, S. Lawrence, and C. L. Giles. Efficient identification of web communities. *ACM KDD*, 2000.

[10] G. W. Flakes, S. Lawrence, C. L. Giles, and F. M. Coetzee. Self-organization of the web and identification of communities. *IEEE Computer*, 2002.

[11] Z. Ge, D. R. Figueiredo, S. Jaiswal, J. Kurose, and D. Towsley. Modeling peer-to-peer file sharing systens. In *Proceedings of Infocom*, 2003.

[12] B. Hayes. Graph theory in practice: Part ii. *American Scientist*, 88(1), January 2000.

[13] A. Langley. *Freenet*, chapter 1. O'Reilly, 2001.

[14] Q. Lv, P. Cao, E. Cohen, K. Li, and S. Shenker. Search and replication in unstructured peer-to-peer networks. In *Proceedings of the 16th international conference on Supercomputing, extended version in http://crypto.stanford.edu/ cao/p2p-search.ps*, pages 84–95. ACM Press, 2002.

[15] E. P. Markatos. Tracing a large scale peer to peer system: an hour in life of gnutella. *Second IEEE/ACM International Symposium on Cluster Computing and the Grid*, 2002.

[16] J. Meadows, March 1999. ftp://researchsmp2.cc.vt.edu/pub/boeing.

[17] D. Nogueira, L. Rocha, J. Santos, P. Araujo, V. Almeida, and W. M. Jr. A methodology for workload characterization of file-sharing peer-to-peer systems. *IEEE Workshop on Workload Characterization*, 2002.

[18] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A scalable content-addressable network. In *ACM Sigcomm 2001 Technical Conference*, San Diego, CA, August 2001.

[19] M. Ripeanu and I. Foster. Mapping the gnutella network: Macroscopic properties of large-scale peer-to-peer systems. *IPTPS*, 2002.

[20] A. Rowstron and P. Druschel. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. In *IFIP/ACM International Conference on Distributed Systems Platforms (Middleware)*, pages 329–350, Nov. 2001.

[21] S. Saroiu, P. K. Gummadi, and S. D. Gribble. A measurement study of peer-to-peer file sharing systems. Technical report, University of Washington, 2001.

[22] K. Sripanidkulchai. The popularity of gnutella queries and its implications on scalability. Technical report, Carnegie Mellon University, 2001.

[23] K. Sripanidkulchai, B. Maggs, and H. Zhang. Enabling efficient content location and retrieval in peer-to-peer systems. Technical report, Carnegie Mellon University, 2001.

[24] K. Sripanidkulchai, B. Maggs, and H. Zhang. Efficient content location using interest-based locality in peer-to-peer systems. *Infocom*, 2003.

[25] The aurora home page. http://www.doc.ic.ac.uk/ twh1/longitude/.

[26] http://freenet.sourceforge.net.

[27] The gnutella home page. http://www.gnutella.wego.com.

[28] The gnutella protocol specification. http://www.clip2.com.

[29] The kazaa home page. http://www.kazaa.com.

[30] The morpheus home page. http://www.morpheus.com.

[31] The napster home page. http://www.napster.com.

[32] J. Vaucher, G. Babin, P. Kropf, and T. Jouve. Experimenting with gnutella communities. *Distributed Communities on the Web International Workshop*, 2002.

[33] B. Yang and H. Garcia-Molina. Efficient search in peer-to-peer networks. Technical report, Stanford University, October 2001.

[34] B. Y. Zhao, J. D. Kubiatowicz, and A. D. Joseph. Tapestry: An infrastructure for fault-tolerant wide-area location and routing. Technical report, University of Berkeley, April 2000.