DIVIDE ET IMPERA: PARTITIONING UNSTRUCTURED PEER-TO-PEER SYSTEMS TO IMPROVE RESOURCE LOCATION

Harris Papadakis, Paraskevi Fragopoulou, Evangelos P. Markatos
Institute of Computer Science
Foundation for Research and Technology-Hellas
P.O. Box 1385, 71 110 Heraklion-Crete, Greece
(adanar | fragopou | markatos)@ics.forth.gr

Marios Dikaiakos
Department of Computer Science
University of Cyprus
P.O. Box 537, CY-1678 Nicosia, Cyprus
mdd@ucy.ac.cy

Alexandros Labrinidis
Department of Computer Science
University of Pittsburgh
Pittsburgh, PA 15260, USA
labrinid@cs.pitt.edu

Abstract Unstructured P2P systems exhibit a great deal of robustness and self-healing at the cost of reduced scalability. Resource location is performed using a broadcast-like process called flooding. The work presented in this paper comprises an effort to reduce the overwhelming volume of traffic generated by flooding, thus increasing the scalability of unstructured P2P systems. Using a simple hash-based content categorization method the Ultrapeer overlay network is partitioned into a relatively small number of distinct subnetworks. By employing a novel index splitting technique each leaf peer is effectively connected to each different subnetwork. The search space of each individual flooding is restricted to a single partition, and is thus considerably limited. This reduces significantly the volume of traffic produced by flooding without affecting at all the accuracy of the search method. Experimental results demonstrate the efficiency of the proposed method.

Keywords: Peer-to-peer, resource location, flooding, overlay network, network partition.

1. Introduction

Peer-to-peer (P2P) systems have recently gained much popularity in the research community as well as among the general public. Researchers show an increasing interest in this paradigm because of its inherent scalability and robustness, which promises to enable the development of global-scale, cooperative, distributed applications. Different entities, under different authoritative control, interconnect and cooperate to offer services to each other, each of them acting both as a server and a as a client, thus the term *peers* for the participating entities.

Existing P2P systems fall into two main categories. *Structured* P2P systems impose a certain order on the connectivity of the participating peers which is reflected in the structure of the overall network. All files stored in the system are indexed in a distributed manner by employing a Distributed Hash Table (DHT), thus enabling efficient resource location in time usually logarithmic to the number of peers. The drawback of this method however is that the maintenance of such a rigid structure limits the ability of structured P2P systems to heal themselves efficiently in the face of failures and thus render them less robust, albeit more scalable.

On the other hand, *unstructured* P2P systems do not impose a certain structure to the network. Those systems are aptly named unstructured since each peer is directly connected to a small set of other peers, called *neighbours*, making the network more ad-hoc in nature. The absence of a structure makes such systems much more robust and highly self-healing compared to structured systems, however, at the cost of reduced scalability. To exploit peer heterogeneity to the system's benefit, in [13, 4] a distinction between peers was introduced and a two level hierarchy of peers was constructed. High bandwidth peers, the *Ultrapeers* (also known as *Superpeers*), form an unstructured overlay network, while peers with low bandwidth, the *Leaves*, are connected only to Ultrapeers. Each Ultrapeer has an index of all the files contained in its Leaves. This modification allows the system to retain its simplicity while offering improved scalability.

Due to the lack of a particular file indexing method, today's unstructured P2P systems employ a broadcast-like process called *flooding* for resource location. A peer looking for a file issues a query which is broadcast in the network, until all peers have received the request or until the query propagates a predefined, maximum number of hops away from its source (Time-To-Live hops or TTL). Flooding generates a large number of messages, reducing the scalability of the method. Due to the completely decentralized nature of flooding, each peer may receive the same request through a number of different neighbours. Those duplicate messages often exceed in number the non-duplicate ones. On a flood aimed to reach the entire network, the number of duplicate messages is $d-2$ times the number of non-duplicate messages, where d is the degree of the overlay network (average number of peers' neighbours). Recent work carried out in P2P systems with the aim of reducing the number of duplicates generated [8]. However, even if all duplicate messages are eliminated, flooding would still not scale well, since the cost of flooding a request to the entire network is relative to the total number of peers. On the other hand, limiting the number of hops a query propagates, achieves improved scalability at the cost of reduced *network coverage* (defined as the percentage of peers that receive a request). When a two level hierarchy of peers is involved, any request originating at a Leaf peer is forwarded through the Ultrapeers it is connected to, while flooding is performed only at the Ultrapeer overlay network.

The aim of the work presented in this paper is to improve the scalability of flooding by reducing the number of peers that need to be contacted on each request, without decreasing the probability of query success (accuracy of the search method). The proposed method partitions the Ultrapeer overlay network into distinct subnetworks. Using a simple hash-based categorization of keywords the Ultrapeer overlay network is partitioned into a relatively small number of distinct subnetworks. In general unstructured P2P networks are indirectly supplied with some information about the possible location of each resource. By employing a novel index splitting technique each Leaf peer is effectively connected to each different subnetwork. The search space of each individual flooding is restricted to a single partition, thus the search space is considerably limited. This reduces the overwhelming volume of traffic produced by flooding without affecting at all the accuracy of the search method (network coverage). Experimental results demonstrate the efficiency of the proposed method.

The remainder of this paper is organized as follows: Following the related work section, the method used to partition the overlay network is presented in Section 3. In Section 4 the simulation details along with the experimental results are presented. We conclude in section 5.

2. Related Work

In an effort to alleviate the large volumes of unnecessary traffic produced during flooding several variations have been proposed. Schemes like Directed Breadth First Search (DBFS) [12] forward requests only to those peers that have often provided results to past requests, under the assumption that they will continue to do so. Interest-based schemes, like [10] and [5] aim to cluster together (make neighbours of) peers with similar content, under the assumption that those peers are better suited to provide each other's needs. Both those systems try to contact peers that have a higher probability of containing the requested information. Such schemes usually exhibit small gains over traditional flooding.

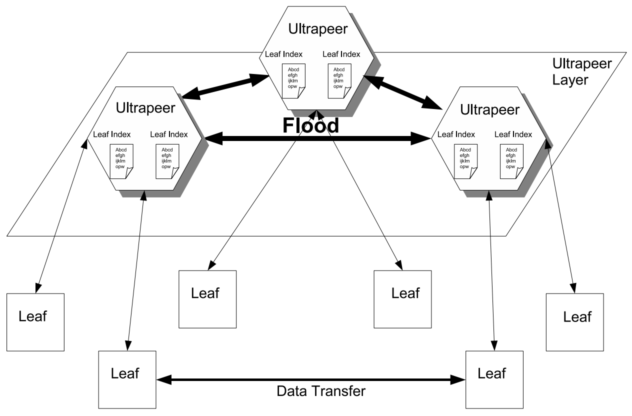

Figure 1. The Gnutella 2-tier architecture

Another technique widely used in unstructured P2P systems today, is 1-hop replication. One-hop replication dictates that each peer should inform all of its immediate neighbours of the files it contains. Using this information during the last hop propagation of a request at the Ultrapeer level, the request is forwarded exclusively to those last hop Ultrapeers that contain the requested file. One-hop replication reduces number of messages generated during the last hop of flooding [7]. However, the traffic generated during that last hop constitutes the overwhelming majority of the traffic generated during the entire flooding. Simple calculations show that 1-hop replication requires *d* times fewer messages to spread to the whole network compared to naive flooding, where *d* is the average degree of the network (average number of connections for each Ultrapeer). It is easy to prove that in order to flood an entire, randomly constructed, network that employs 1-hop replication, one need only reach $3/d$ of the peers during all hops but the last. In today's Gnutella, where the average degree is 30, one would need to reach 10% of the peers and then use 1-hop replication to forward the query to the appropriate last hop peers, in order to reach the entire network.

Most of today's unstructured P2P systems implement 1-hop replication by having peers exchange bloom filters of their indices. A Bloom filter [3] is a space efficient way to represent a set of objects (keys). They employ one or more uniform hash functions to map each key to a position in an N-sized binary array, whose bits are initially set to 0. Each key is mapped through each hash function to an array position which is set to 1. To check for the participation of some key in the set, the key is hashed to get its array position. If that array position is set to 1, the bloom filter indicates key membership. Bloom filters require much less space than the actual set, there is thus some loss of precision translated in the possibility of *false positives*. This means that a bloom filter may indicate membership for some key that does not belong to the set (more than one keys mapped to the same position). It cannot however indicate absence of a key which is in the set (false negative).

In Gnutella 2 [1] which uses a 2-tier architecture, each Leaf node sends its (list of keywords) in the form of a bloom filter to all Ultrapeers it is connected to. Each Ultrapeer produced the XOR of all the bloom filters it receives from its Laves (approximately 30 Leaf nodes per Ultrapeer) and transmits this collective bloom filter to all its neighboring Ultrapeers to implement the 1-hop replication.

Another approach that has been used in the literature to make resource location in unstructured P2P systems more efficient is the partitioning of the overlay network into subnetworks using content categorization methods. A different subnetwork is formed for each content category. Each subnetwork connects all peers that posses files belonging to the corresponding category. Subnetworks are not necessarily distinct. A system that exploits this approach is the Semantic Overlay Networks (SONs) [6]. SONs use a semantic categorization of music files based on the music genre they belong to. The main drawback of this method is the semantic categorization of the content. In file-sharing systems for instance, music files rarely contain information about the genre they belong to and when they do so, each of them probably uses a different categorization of music. In SONs, an already existing, online, music categorization database is used. This database adds a centralized component in the operation of the network. Notice that 1-hop replication can be employed in conjunction with this scheme, inside each subnetwork. However, the fact that each peer may belong to more than one subnetwork, reduces the average degree of each subnetwork and thus, the efficiency of the 1-hop replication.

3. The Partitions Design

The system we propose in this paper allows for the partitioning of any type of content. More specifically, we propose the formation of categories based on easily applicable rules. Such a simple rule is to apply a uniform hash function on each keyword describing the files. This hash function maps each keyword to an integer, from a small set of integers. Each integer defines a different category. We thus categorize the keywords instead of the content (files) itself. Given a small set of integers, it is very likely that each peer will contain at least one keyword from each possible category.

Unstructured P2P systems like Gnutella 2 [1] employ a 2-tier structure. In those systems Ultrapeers form a random overlay network, while Leaf nodes are connected to Ultrapeers only. Each Leaf sends to the Ultrapeers it is connected to its index in the form of a (compressed) bloom filter. Ultrapeers flood queries

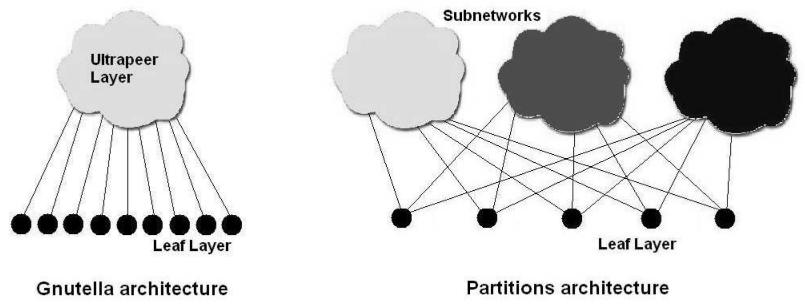

Figure 2. Illustration of the Gnutella network and the Partitions design

to the overlay network on the Leave's behalf. Flooding is only performed at the Ultrapeer level where 1-hop replication is implemented. Whenever an Ultrapeer receives a request this is targetedly forwarded only down to those Leaves that contain the desired information (except in the case of false positives). Fig. 1 shows a schematic representation of the 2-tier architecture.

The keyword categorization method is used in 2-tier unstructured systems. In the Partitions design, each Ultrapeer in the system is randomly and uniformly assigned responsibility for a single keyword category, by randomly selecting an integer from the range set of the hash function used to categorize the keywords. Ultrapeers responsible for the same category form a distinct subnetwork. Leaves connect to one Ultrapeer per subnetwork and send to it all the keywords belonging to that category. Thus, an innovative index splitting technique is used. Instead of each Leaf sending its entire index (in the form of a bloom filter) to an Ultrapeer, each Leaf splits its index (keywords) based on the defined categories and distributes it to one Ultrapeer per category. Notice that peers operating as Ultrapeers also operate as Leaves at the same time (have a dual role). Even though in this design each Leaf connects to more than one Ultrapeers, the volume of information it transmits is roughly the same since each part of its index is send to a single Ultrapeer. Each Leaf node sends to the Ultrapeer of a certain category all keywords that belong to the same category (in the form of a bloom filter). Each Ultrapeer sends to its neighboring Ultrapeers all the aggregate indices of its Leaf nodes to implement 1-hop replication. In Fig. 2 we can see a schematic representation of the Partitions design.

This separation of Ultrapeers from content has the benefit of allowing them to be responsible for a single keyword category. The benefit of this is two-fold. First, it reduces the size of the subnetworks since they are completely discrete (at least on the overlay level). Secondly, it allows each Ultrapeer to use all its

Ultrapeer connections to connect to other Ultrapeers of the same subnetwork, increasing the efficiency of 1-hop replication at the Ultrapeer level.

There are, however, two obvious drawbacks to this design. The first one is due to the fact that each Leaf connects to more than one Ultrapeers, one per content category. Even though each Leaf sends the same amount of index data to the Ultrapeers upon connection as before, albeit distributed, however it requires more keepalive messages to ensure that its Ultrapeers are still operating. Keepalive messages however are very small compared to the average Gnutella protocol message. In addition, query traffic is used to indicate liveliness most of the time, thus avoiding sending keepalive messages. The second drawback arises from the fact that each subnetwork contains information for a specific keyword category. Requests however may contain more than one keywords and each result should match all of them. Since each Ultrapeer is aware of all keywords of its Leaves that belong to a specific category, it may forward a request to some Leaf that contains one of the keywords but not all of them. This fact reduces the efficiency of the 1-hop replication at the Ultrapeer level and at the Ultrapeer to Leaf query propagation. This drawback is balanced in two ways. The first is that even though the filtering is performed using one keyword only, Leaves' bloom filters also contain one type of keywords only, making them more sparse and thus reducing the probability of a false positive. Furthermore, the most rare keyword can be used to direct the search, thus further increasing the effectiveness of the search method. Finally, we also experimented with sending the bloom filters with all keyword types to every Ultrapeer, regardless of category, although Ultrapeers still extract and use only keywords of the same category as their own to form their aggregate bloom filter in order to implement 1-hop replication.

All these schemes have varying degrees of maintenance costs which we explore in the next section using simulations.

4. Experimental Results

In this section, we shall present the results from the simulations we conducted, in order to measure both the efficiency of the Partitions scheme in terms of cost of flooding (in messages) and maintenance costs.

We assumed a peer population of 2 million, a number reported by LimeWire Inc [2]. Each Ultrapeer in the Gnutella network serves 30 Leaves, a number obtained from real-world measurements [11]. In addition, each peer contains a number of files (and hence keywords) derived from a distribution also obtained from real-world measurements in [9].

Each Ultrapeer in the Partitions design serves 300 Leaves since we assume a number of 10 content categories and thus subnetworks. We perform a large number of floods, each designed to return at least a thousand query results before terminating. Table 1 shows the ratio of the average number of messages per flood for the Partitions design over the average number of messages per flood in Gnutella. Replication means that each Leaf sends all its keywords to all Ultrapeers it is connected to, regardless of category. For example, in the case of replication, flooding in the Partitions design generates 5.5 times less messages than flooding in Gnutella, in order to return the same number of results per query. We can see that the drawback of filtering using only one keyword is balanced by the fact that the sparser Leaf indices (since they contain only one keyword category) produce less false positives, but mainly outweighed by the message reduction due to the partitioning of the network and therefore the reduction of the search space. We would like to emphasize that each Partitions bloom filter (i.e. containing keywords of a certain category) has the length of a Gnutella bloom filter. Thus, one can roughly think of all the bloom filters of a single Partitions leaf as a (distributed) Gnutella bloom filter of 10 times the length (due to the 10 category types). However the bandwidth needed to transfer such a bloom filter is not 10 times that of a Gnutella bloom filter, mainly because sparser bloom filters are compressed more efficiently.

Table 1. Flooding efficiencies.

	Ratio
No replication	4.2
Replication	5.5

In order to measure the maintenance cost of Gnutella and Partitions, we focus on the operation of a single Ultrapeer, because the load of Leaves is negligible in both systems compared to a Ultrapeers load since flooding is performed at the Ultrapeer overlay. In both cases we simulated three hours in the life of a single Ultrapeer, with Leaves coming and going. Each time a Leaf is connecting to the Ultrapeer, it sends its index information, which is propagated by the Ultrapeer to its thirty Ultrapeer neighbors. In addition, we assumed that, periodically , each Ultrapeer receives a small keep-alive message from each Leaf and replies with a similar message to each one of them, unless a query and a reply were exchange during the specified period. For each communication taking place, we measured the incoming or outgoing traffic in bytes, in order to estimate the bandwidth requirements.

There are two modifications in this scenario, between Gnutella and Partitions. In Partitions, the number of Leaves is 300. In addition, the process of computing the size of the index information sent to the Ultrapeer differs greatly. In the case of Gnutella, we have used the code by LimeWire [2], the most popular Gnutella client, to construct the bloom filter of each Leaf. We first randomly decided on

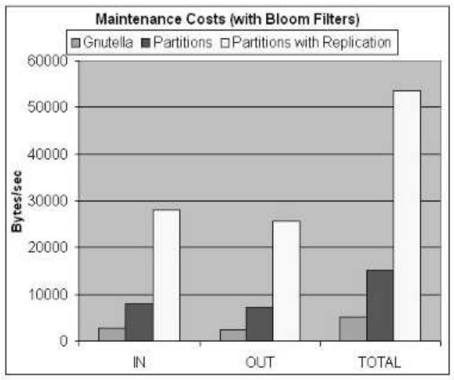

Figure 3. Maintenance traffic load for Gnutella and Partitions using Bloom Filters. Incoming, Outgoing and Total traffic.

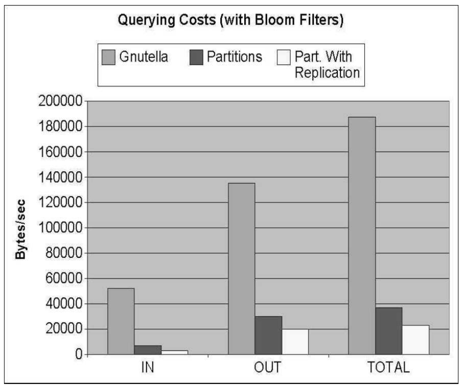

Figure 4. Query traffic load for Gnutella and Partitions using Bloom Filters. Incoming, Outgoing and Total traffic.

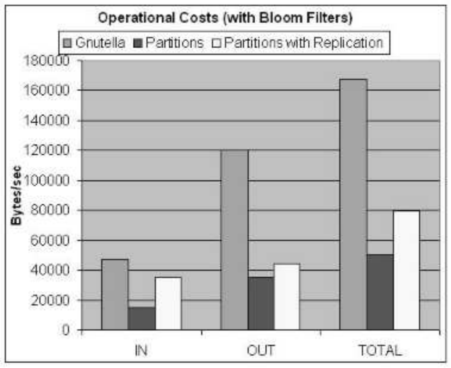

Figure 5. Operational traffic load for Gnutella and Partitions using Bloom Filters. Incoming, Outgoing and Total traffic.

the number of files shared by each Leaf, based on the file sharing distribution per peer presented in [9]. We then extracted this number of files from a list of filenames obtained from the network by a Gnutella crawler developed in out lab. Those filenames were fed to the LimeWire bloom filter generation code, which produced the corresponding bloom filter in compressed form, i.e., the way it is sent over the network by LimeWire servents. Thus we constructed the actual bloom filter, although what we really need in this case is just its size. In the case of Partitions, we likewise computed the number of files to be shared by each Leaf. We extracted again the same number of filenames from the list of available filenames.

We subdivided the Partitions scheme depending on the form of the index information sent by Leaves to Ultrapeers. Two experiments were run with the Partitions scheme using bloom filters. In the first, each bloom filter sent to an Ultrapeer only contained appropriate keywords (of the same category as the corresponding Ultrapeer). In the second experiment, we used replication, i.e. each bloom filter contained all the keywords of the Leaf, regardless of category. In addition, positions of keywords of the corresponding category as the Ultrapeer were set in the bloom filter to the value of two instead of one. (This bloom filter essentially distinguishes between keywords of the appropriate category and the rest of the categories).

Fig. 3 shows the results of the simulation for the cost of maintaining the structures of Gnutella and Partitions, without any query (flood) traffic. From this figure it is obvious that, as expected, the maintenance cost of partitions is higher than that of Gnutella, but not that much. As we will see in the next paragraph the gains incurred during the operational phase of the two systems outweighs the increased maintenance costs.

We then focused our attention to the query traffic load. Measurements conducted in our lab showed that, on the average, each Ultrapeer generates 36 queries per hour (i.e., queries initiated by itself or its Leaves). This adds up to approximately 2000 queries per second generated anywhere in the Gnutella network. In addition, we observed a large number of Gnutella queries in order to find the distribution of the number of keywords in each query. Thus, according to those observations, during the simulations we assumed that 20% of the queries contain 1 keyword, 30% contain two, another 20% contain three and finally a 30% contain 4 keywords.

In our simulation, we assumed that the aim of each flood (both in Gnutella and Partitions) is to reach the entire network, or produce a fixed number of results, whichever comes first. As we mentioned before, such a flood that aims to reach the entire network would need to reach $\frac{1}{10}$th of the Gnutella's network (or a Partitions' subnetwork) during all hops of flooding except the last. This means that the Ultrapeer in our simulations has a probability of 0.1 to receiving each query. In addition, every time this does not occur, it has another opportunity to receive the query during the last hop, depending on its bloom filter (in case the searched keywords match in the bloom filter). Should the Ultrapeer receive a query, it is assumed to propagate it to its Leaves, again depending on their bloom filters or index (again depending on a possible keyword match by the bloom filter). Fig. 5 shows the comparison in the traffic load of Gnutella and Partitions, including maintenance and query traffic. We used a size of 40 bytes for each query. In reality, the size of a query can be up to a few hundred bytes, if XML extensions are used. This means that the performance gains described here are smaller compared to the ones we expect to see in the real world. In addition, for every 1400 bytes for each message sent, we added 40 bytes for the TCP and IP header. From these figures it is evident that Partitions outperform Gnutella in operational costs, in every case. Finally in Fig. 4 one can see the query traffic load alone (without the maintenance traffic) for both the Gnutella and the Partitions Ultrapeer.

5. Conclusions

In this paper, we have described a novel approach to reducing the message costs of querying an unstructured network. A simple model has been described to illustrate that the benefits obtained from our scheme can be as high as an order of magnitude. Work is being carried out to measure the performance of our scheme, while varying the number of partitions. Furthermore, the benefit of Leaves communicating their full index (actual keywords) to Ultrapeers instead of bloom filter is currently exploited.

Acknowledgments

This research work was carried out under the FP6 NoE CoreGRID funded by the EC (IST-2002-004265) and was supported by project SecSPeer (GGET USA-031) funded by the Greek Secreteriat for Research and Technology.

References

[1] Gnutella 0.6 protocol specification. http://rfc-gnutella.sourceforge.net/developer/stable/index.html

[2] Limewire Inc. http://www.limewire.com

[3] B.H. Bloom. Space/time trade-offs in hash coding with allowable errors. *Communications of the ACM*, 13(7):422–426, 1970.

[4] Y. Chawathe, S. Ratnasamy, L. Breslau, N. Lanham, and S. Shenker. Making Gnutella-like P2P Systems Scalable. *Proc. ACM SIGCOMM 2003 Conf. on Applications, Technologies, Architectures, and Protocols for Computer Communication*, pp. 407-418, 2003.

[5] V. Cholvi P. Felber, and E. Biersack. Efficient search in unstructured peer-to-peer networks. *Proc. 16th ACM Symposium on Parallelism in Algorithms and Architectures*, 2004.

[6] A. Crespo and H. Garcia-Molina. Semantic overlay networks for p2p systems. Technical report, 2002.

[7] C. Gkantsidis, M. Mihail, and A. Saberi. Hybrid search schemes for unstructured peer-to-peer networks. *Proc. of INFOCOM*, 2005.

[8] C. Papadakis P. Fragopoulou E. Athanasopoulos M. Dikaiakos, A. Labrinidis, and E. Markatos. A feedback-based approach to reduce duplicate messages in unstructured peer-to-peer networks. *Proc. of the CoreGRID Integration Workshop*, 2005.

[9] R. Rejaie, Shanyu Zhao, and D. Stutzbach. Characterizing files in the modern Gnutella network: A measurement study. *Proc. SPIE/ACM Multimedia Computing and Networking*, 2006.

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

[11] D. Stutzbach and R. Rejaie. Characterizing the two-tier gnutella topology. *Proc. of the ACM SIGMETRICS, Poster Session*, 2005.

[12] B. Yang and H. Garcia-Molina. Improving search in peer-to-peer networks. *Proc. of the 22nd International Conference on Distributed Computing Systems (ICDCS02)*, 2002.

[13] B. Yang and H. Garcia-Molina. Designing a Super-Peer Network. *Proc. Int. Conference on Data Engineering (ICDE 2003)*, pp. 49-60, 2003.

[14] Fisk, A. Gnutella Ultrapeer Query Routing, v. 0.1. LimeWire Inc. 2003