Institut EURECOM
Research Report N^o 90 — RR-03-090

Efficient Search in Unstructured Peer-to-Peer Networks

V. Cholvi, P. Felber, E.W. Biersack

September 22, 2003

Efficient Search in Unstructured Peer-to-Peer Networks

***Abstract*—The huge popularity of recent peer-to-peer (P2P) file sharing systems has been mainly driven by the scalability of their architectures and the flexibility of their search facilities. Such systems are usually designed as *unstructured* P2P networks, because they impose few constraints on topology and data placement and support highly versatile search mechanisms. A major limitation of unstructured P2P networks lies, however, in the inefficiency of their search algorithms, which are usually based on simple flooding schemes.**

In this paper, we propose novel mechanisms for improving search efficiency in unstructured P2P networks. Unlike other approaches, we do not rely on specialized search algorithms; instead, the peers perform local dynamic topology adaptations, based on the query traffic patterns, in order to spontaneously create communities of peers that share similar interests. The basic premise of such semantic communities is that file requests have a high probability of being fulfilled within the community they originate from, therefore increasing the search efficiency. We propose further extensions to balance the load among the peers and reduce the query traffic. Extensive simulations under realistic operating conditions substantiate that our techniques significantly improve the search efficiency and reduce the network load.

I. Introduction

A. Motivations

The last few years have witnessed the appearance of a growing number of peer-to-peer (P2P) file sharing systems. Such systems make it possible to harness the resources of large populations of networked computers in a cost-effective manner, and are characterized by their high scalability.

P2P file sharing systems mainly differ by their search facilities. The first hugely successful P2P data exchange system, Napster [1], incorporates a centralized search facility that keeps track of files and peer nodes; queries are executed by the central server, while the resource-demanding file transfers are performed using P2P communication. This hybrid architecture offers powerful and responsive query processing, while still scaling well to large peer populations. The central server needs, however, to be properly dimensioned to support the user query load. In addition, it constitutes a single point of failure and can easily be brought down in the face of a legal challenge, as was the case for Napster. Consequently, most recent P2P file sharing systems have adopted more decentralized architectures.

Roughly speaking, the P2P networks that do not rely on a centralized directory can be classified as either *structured* or *unstructured*. Structured P2P networks (e.g., Chord [2], CAN [3], Pastry [4], and Tapestry [5]) use specialized placement algorithms to assign responsibility for each file to specific peers, as well as "directed search" protocols to efficiently locate files. In contrast, unstructured P2P networks (e.g., Gnutella [6] and Freenet [7]) have no precise control over the file placement and generally use "flooding" search protocols.

Directed search protocols are particularly efficient, because they accurately route queries toward the peers responsible for a given file. They require few communication steps, generate little traffic, and do not produce false negatives (i.e., the search fails only if there is no matching file in the system). Flooding protocols are less efficient, because queries are generally broadcast indiscriminately in a whole neighborhood and may yield false negatives. They have, however, very little management overhead, adapt well to the transient activity of P2P clients, take advantage of the spontaneous replication of popular content, and allow users to perform more elaborate queries than with directed search protocols, which only support exact match queries. These properties make unstructured P2P systems more suitable for mass-market distributed file sharing.

The main objective of this work is to develop techniques to render the search process in unstructured P2P file sharing systems more efficient and scalable, by taking advantage of the common interests of the peer nodes and effectively implement a "directed flooding" search protocol.

B. Overview and Contributions

In this paper we propose $\mathcal{A}cquaintances$, an extension to Gnutella-like unstructured P2P networks that uses dynamic topology adaptation to improve search efficiency. As in Gnutella, our search mechanism uses TTL-limited flooding to broadcast queries in a neighborhood. By associating a TTL (time to live) value to the query messages, one can restrict the search diameter, i.e., the size of the flooded neighborhood, and limit the adverse effects of exponential message generation on the network links. However, the probability of finding a file that does exist in the network strongly depends on the chosen TTL value: bigger values increase the success rate but may quickly lead to network congestion.

To minimize this problem, we propose novel techniques to build self-organized communities of peer nodes that share similar interests. These communities are maintained by dynamically adapting the topology of the P2P overlay network based on the query patterns and the results of preceding searches. Some of the neighbor links are explicitly reserved for building semantic communities and are continuously updated according to some link replacement algorithm; these links allow peers to quickly locate files that match their interests. The other links are mostly static and random; they help maintain global connectivity and locate more marginal content.

Semantic communities can have the adverse effect of creating hot-spots with well-connected peers. To address this problem, we introduce a load-balancing mechanism that delegates,

whenever possible, the responsibility to answer a query to less-loaded peer nodes. Finally, we propose a dynamic TTL update scheme to further limit network congestion without significantly degrading the query success rate.

To evaluate the effectiveness of our techniques, we have built a network simulator and conducted extensive simulations under realistic operating conditions. Results demonstrate that, when extending a basic Gnutella-like network with $\mathcal{A}cquaintances$, one can significantly improve the search efficiency and reduce the network load.

C. Paper Organization

The rest of this paper is organized as follows: In Section II, we discuss related work, and we introduce the design of $\mathcal{A}cquaintances$ in Section III. Section IV describes the methodology used for the evaluation of $\mathcal{A}cquaintances$, and the simulation results are presented in Section V. Finally, Section VI concludes the paper.

II. Related Work

Our system builds on top of unstructured P2P networks, such as Gnutella [6], but dynamically adapt the network topology to build semantic communities. Several alternative approaches have been proposed to improve search efficiency by taking advantage of the common interests of the peer nodes.

In [8], the authors propose the use of *shortcuts* to exploit interest-based locality: peers that share similar interests create shortcuts to each other. Queries are first disseminated through shortcuts and, if the search fails, they are flooded through the underlying P2P overlay. In contrast, our approach does not create additional links, nor does it require a specialized search process; it rather dynamically modifies the topology of the overlay network to reflect the shared interests of the peers, and can thus be incorporated seamlessly into existing Gnutella-like P2P networks.

In [9], the authors propose techniques to reduce the number of nodes that process a query, with the premise that by intelligently selecting subsets of peers to send queries to, one can quickly obtain relevant results. This work focuses on the peer selection algorithms, which yield various performance gains, but does not consider the dynamic evolution of the structure and connectivity of the P2P overlay.

In [10], the authors propose a query algorithm based on multiple random walks. This technique reduces message overhead compared with flooding, but at the expense of an increase in the number of hops necessary to find the requested file.

Gia [11] is a P2P file sharing system that extends Gnutella with *one-hop replication*. Each peer maintains an index of the files stored on its neighbors, and can thus process queries on their behalf for increased scalability. *Local indices* [9] implement the same concept, but extend the scope of indexes to all the peers located within a predefined hop distance. We also rely on similar techniques to increase search efficiency, but we further use them for load balancing purposes and we show that they are effective even when indexes have stringent size limitations.

In [12], the authors propose to use file associations to build a so-called *associative overlay* and present various algorithms to steer the search process to peers that are more likely to have an answer to the query. In contrast, our approach does not require specialized searching rules; it rather drives the search process by dynamically adapting the network topology.

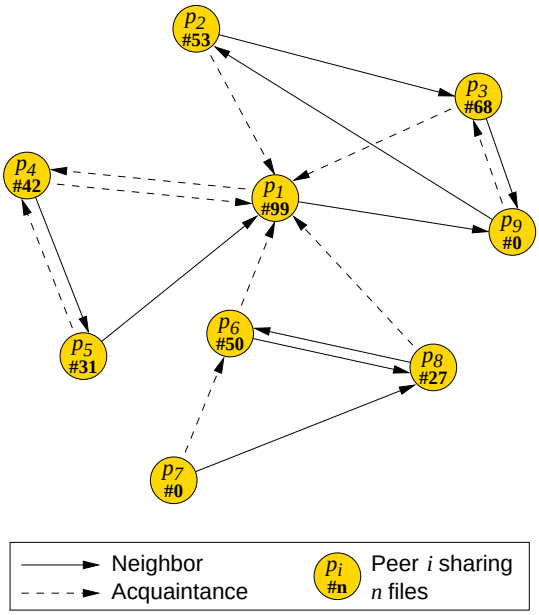

Fig. 1. Sample minimal network, with a single neighbor and acquaintance link per peer.

III. $\mathcal{A}cquaintances$ Design

In this section, we introduce the basic principles of $\mathcal{A}cquaintances$ and we present the key components and algorithms used in its design.

A. Definitions and Terminology

Each peer is connected to a set of other peers in the network via *uni-directional* links, that is, each peer can locally select the other peers it wishes to link to. We distinguish between two types of links:

- *Neighbor links* connect a peer p to a set of other peers (p's neighbors) chosen at random, as in typical Gnutella-like networks.
- *Acquaintance links* connect a peer p to a set of other peers (p's acquaintances) chosen based on common interests.

Each peer has a bounded number of neighbor and acquaintance links. We call p's *friends* the set of peers that have p among their acquaintances. The number of friends of a peer is its *in-degree* (which is unbounded, let alone by the size of the network).

A peer can make some of its local files accessible to other peers. Peers that do not share any file are called *free-riders*. *Non-free-riders* or *serving peers* are those peers that contribute files to the community. A successful request yields a list of peers that have a file matching the original query. We assume that, when several peers have the desired file, the peer that is closest to the requester (in number of hops) is chosen. We call that peer the *answerer*. Note that answering a query typically implies sending a file to the requester.

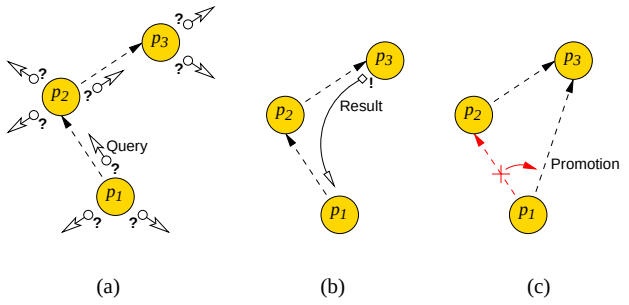

Fig. 2. Basic principle of dynamic topology adaptation: (a) p_1 issues a query. (b) p_3 returns a positive response. (c) p_1 promotes p_3 as acquaintance.

Some of the mechanisms that we will introduce shortly require peers to maintain state information about their friends. The *state* of a peer consists of the list of the names of its shared files. For load-balancing purposes, peers also need to know the in-degree of their friends.

To illustrate these definitions, consider the sample network depicted in Fig. 1, in which each peer has a single neighbor and acquaintance link. Peer p_1 shares 99 files and its state consists of the names of all these files. It has p_9 as random neighbor; p_4 as acquaintance; and p_2, p_3, p_4, p_6, and p_8 as friends, which corresponds to an in-degree of 5. Peers p_7 and p_9 are free-riders and have no friends. As we will see later, a high in-degree generally indicates that a peer shares many files, or is well-connected to peers that share many files; in contrast, free-riders typically have a null in-degree. Pair-wise acquaintance relationships between serving peers that have similar interests (e.g., between p_1 and p_4) are also common in practice, and effectively yield bi-directional links.

B. Dynamic Topology Adaptation

The basic principle of $\mathcal{A}cquaintances$ consists in dynamically adapting the topology of the P2P network so that the peers that share common interests spontaneously form "well-connected" semantic communities. It has been shown that users are generally interested in only some specific types of content [13], therefore being part of a community that shares common interests is likely to increase search efficiency and success rate.

Dynamic topology adaptation is implemented by directing acquaintance links toward the peers that have returned relevant results in the past (see Fig. 2). Indeed, a peer that consistently returns good results is likely to have common interests with the requesting peer and/or to serve a large number of files. A consequence of this scheme is that selfish peers and free-riders are likely to be acquainted with almost no other peer.

Each peer maintains an bounded list of acquaintances. The decision of replacing a peer from this list, i.e., promoting a peer as acquaintance, depends on the history of the responses to previous requests issued by each peer (note that this decision is local). In this paper, we evaluate two acquaintance replacement policies.

The *Least Recently Used (LRU)* policy is the simplest. After a successful request, a peer adds the answering peer in front of its list, and drops the last peer of the list (see Algorithm 1). If the answering peer is already an acquaintance, it is moved to the front. This scheme guarantees that a promoted peer always replaces the peer that was promoted least recently, and that a peer that regularly answers queries can remain an acquaintance for a long duration. However, when peers have diverse interests and have only few acquaintance links at their disposal, the composition of the acquaintance list may well change after every query; this volatility can yield non-negligible connection management costs.

Algorithm 1 *LRU* replacement policy at requester p_r

Variables:
  AcqList: Ordered list of N acquaintances, initially chosen at random

Upon successful query answered by p_a:
  if $p_a \in$ *AcqList* **then**
    remove p_a from *AcqList*
  else
    remove last element from *AcqList*
  end if
  add p_a to front of *AcqList*

Algorithm 2 *MOU* replacement policy at requester p_r

Variables:
  AcqList: Ordered list of N acquaintances, initially chosen at random
  CandList: List of $\{peer; ranking\}$ pairs, ordered by ranking,
    initially filled with peers from *AcqList* and null rankings
  α: Aging factor, with value in $(0; 1]$

Upon successful query answered by p_a, reached via $(p_r, p_1, \cdots, p_{n-1}, p_n = p_a)$:
  for all $\{p; r\} \in$ *CandList* **do**
    $\{p; r\} \leftarrow \{p; \alpha r\}$
  end for
  $i \leftarrow 1.0$
  for j **from** n **downto** 1 **do**
    if $\{p_j; r\} \in$ *CandList* **then**
      $\{p_j; r\} \leftarrow \{p_j; r + i\}$
    else
      insert $\{p_j; i\}$ in *CandList*
    end if
    $i \leftarrow i/2$
  end for
  AcqList $\leftarrow$ first N peers of *CandList*

To alleviate this problem, we use the *Most Often Used (MOU)* policy, which maintains rankings of the peers and elects as acquaintances those that have the highest rankings. A peer has a high ranking if it answers to many queries, or (to a lesser extent) if it is close to peers that have answered many queries, i.e., it is well connected and could thus be a valuable acquaintance. After each successful query, each peer on the path followed by the query, in reverse order from the answerer to the requester, has its rank increased by an exponentially decreasing value (see Algorithm 2). To better adapt to the dynamics of the peer population and shared content, we also introduce an aging factor that gives more weight to recent answers by decreasing the rankings over time. This scheme clearly yields acquaintance lists with low volatility, which give preference to peers that stay longer in the system and are expected to be more stable.

To best illustrate the effect of dynamic topology adaptation in $\mathcal{A}cquaintances$, we have represented in Fig. 3(a) and 3(b) the acquaintance links of a small P2P network, before and after running a simulation (as described in Section IV). The network evolves from a random configuration toward a graph with well-connected communities.

(a) At the begin of the simulation.

(b) At the end of the simulation.

Fig. 3. Graphical representation of the effect of dynamic topology adaptation on a 200-peers network, with one acquaintance link per peer (neighbor links are not shown for clarity).

C. Search

As previously mentioned, $\mathcal{A}cquaintances$ does not require complex or specialized search algorithms. It uses the same TTL-limited flooding scheme as in Gnutella-like P2P networks, and yet exhibits much improved search efficiency.

First, by organizing peers in communities that share common interests, we improve response time by increasing the chances that matching files are found inside the community, i.e., within a short distance, of the requester. We can therefore use smaller TTL values for queries and thus reduce the network traffic, without significant impact on the success rate.

Second, peers can be configured to maintain a (partial) index of the files stored on their friends. Using this state information, a peer can explore several other peers with similar interests at no communication cost. Clearly, this also increases the success rate and reduces the network traffic.

Semantic communities also have some drawbacks. First, the network can quickly become divided in several disconnected subnetworks with disjoint interests. Second, peers searching for unpopular or marginal content (not part of their interests) may experience very low success rates. For these reasons, we enforce some random connectivity by means of the neighbor links. Searches are performed by forwarding queries on both the acquaintance and neighbor links.

D. Load Balancing

Flooding algorithms naturally direct much of the traffic toward highly connected peers. In our system, a peer that has many friends can quickly become a hot-spot, not only because it receives more queries, but also because it typically sends more files to requesting peers. Although we do not explicitely address the issue of file transfers in this paper, it is a large source of overhead in P2P file-sharing networks and should not be overlooked.

We therefore use the following mechanism to better balance the file traffic. Before successfully answering a query, a peer p first checks if any of its friends also has the requested file. If so, it delegates the responsability for answering the query to the peer among those serving the file that has the *smallest in-degree* (note that this peer may be p). Otherwise, p sends the file itself.

The rationale behind this approach is that well-acquainted peers are likely to be more loaded, i.e., receive more requests and serve more files, than peers with fewer friends. Further, there is a good probability that some of the friends of a peer also have the same files. Therefore, we force the less loaded peer to assume part of the load.

E. Dynamic TTL

To further reduce the query traffic generated by the flooding algorithm, we propose an extension for the case where peers are "conscious" of the sematic communities and can check if a requested file falls within their interests. Intuitively, a query that enters a community of peers to which the requested file belongs, is likely to be answered by a peer of that community within a few hops. Conversely, a query for a file that does not match the interests of the community is likely to traverse more peers before being satisfied. Therefore, we propose decrementing the TTL value twice (i.e., by 2 instead of 1) when a received query falls within the interests of the traversed peer. Clearly, this mechanism reduces the number of messages sent by the flooding algorithm. Our basic premise, which we substantiate later, is that this extension significantly reduces network traffic without affecting much the success rate.

IV. Experimental Setup

We now present the experimental setup and methodology used for the evaluation of $\mathcal{A}cquaintances$. All results were obtained from simulations, with extra care taken at reproducing realistic operating conditions.

A. System Model

a) Network: Similarly to Gnutella, we consider a system model where peers are organized in an overlay network. Each peer has a set of neighbors with which it communicates by message passing. Links are directed: a peer p may have another peer p' as neighbor without p' considering p as its neighbor. Traffic can however flow in both directions on the links.

We consider a peer-to-peer network made of $20,000$ peers, which corresponds to an average-size Gnutella network [14]. Each peer has 6 of outgoing links. Some of them are chosen randomly (neighbor links), and others adapt dynamically based on the query traffic (acquaintance links). The number of acquaintance links varies between 0 and 5 in our experiments (the remaining links being neighbor links). As we need to maintain global network connectivity during the whole simulation to obtain consistent results, we make sure that the network is initially fully connected through the peers' neighbors links (which do not change over time). The number of incoming links is unbounded and varies over time as acquaintance links get updated.

b) Content: It has been observed [13] that users are generally interested in a subset of the content available in a peer-to-peer network. Furthermore, they are interested in only a limited number of "content categories" (e.g., music styles, literature genres). Among these categories, some are more popular and contain more content than others; for instance, pop music files are more widely held than jazz or classical music files. Similarly, within each category, some files are much more popular than others. The popularity of the Gnutella content and queries has been shown to follow a Zipf distribution, with a skew factor between 0.63 and 1.24 [15]; in the rest of this paper, unless mentioned otherwise, we use a Zipf skew of 1.0 for all our experiments.

We model content by creating 50 distinct categories. Each category has an associated popularity index, chosen according to a Zipf distribution. We then create $200,000$ distinct files and assign each of them to exactly one category chosen according to the categories' popularities: the more popular a category is, the more files it contains. We also associate a Zipf popularity to the files inside each category. Finally, each peer is assigned a random number (uniformly distributed between 1 and 6) of categories of interest, that it chooses according to their popularity index. The categories of interest of a peer are ranked using a Zipf distribution: the peer is more interested in (i.e., requests and shares more files from) the first chosen category than the second one. This behavior models the few peers that are highly interested in marginal content.

c) Cooperation: All peers do not share the same number of files and do not exhibit the same "social behavior". As observed in [16], a large proportion of the user population is made of so-called free-riders, who do not make any file accessible to other users and essentially behave as clients. On the other hand, a small proportion of the users (less than 5%) contribute more than two thirds of the files shared in the system and essentially behave as servers. Based on the study in [16], we assign the following storage capacity to the peers in the network: 70% of the peers do not share any file (free-riders); 20% share 100 files or less; 7% share between 101 and $1,000$ files; finally, 3% of the peers share between $1,001$ and $2,000$ files (actual storage capacities are chosen uniformly at random). With this distribution, we have observed in our experiments a total storage capacity of more than $1,600,000$ files, with more than $150,000$ distinct files being shared.

Parameter	Value
Active peers	$20,000$
70% share	0 file $\equiv$ free-riders
20% share	$[1 \ldots 100]$ files (uniform)
7% share	$[101 \ldots 1000]$ files (uniform)
3% share	$[1001 \ldots 2000]$ files (uniform)
Distinct files	$200,000$
Categories	50
Categories per peer	$[1 \ldots 6]$ (uniform)
Links per peer	6
Acquaintance links	$\{0, 1, 3, 5\}$
Query TTL	6
Query rate	10%

TABLE I
Parameters used in the simulations.

B. Simulation Methodology

Our simulator proceeds in a sequence of synchronous rounds. In each round, a subset of the peers (10% in our experiments) issue requests. Similarly to Gnutella, searches are conducted using TTL-limited flooding. Each request is assigned a time-to-live (TTL) value and is disseminated via neighbor and acquaintance links. When receiving a request for the first time, a peer decreases the TTL value and, if it is strictly positive, propagates the request further.

To generate a request, a peer first selects one category among its categories of interest based on their rankings. Then, it selects a file (that it does not already hold) from that category according to the file popularities. The peer then issues a request for that file. For simplicity, we always request individual files, i.e., we do not consider broad queries that match several distinct files.

When a serving peer receives a positive response to his query and it still has some storage capacity available, it creates a local copy of the file and makes it accessible to other users. A positive response to a query can also result in an update of a peer's acquaintance links. We used an aging factor of $\alpha = 1$ for the MOU acquaintance replacement policy (see Algorithm 2).

The simulation is made of two phases:

1) During the first phase (bootstrap phase), we populate the system and establish acquaintance link connectivity between the peers. To that end, serving peers issue queries, at a rate proportional to their storage capacity, and create local copies of the files they request. In case a file is not found (e.g., because it does not yet exist in the system), we "inject" it at the requesting peer; this models the behavior of peers joining the network with a pre-existing set of files. Acquaintance links are dynamically updated

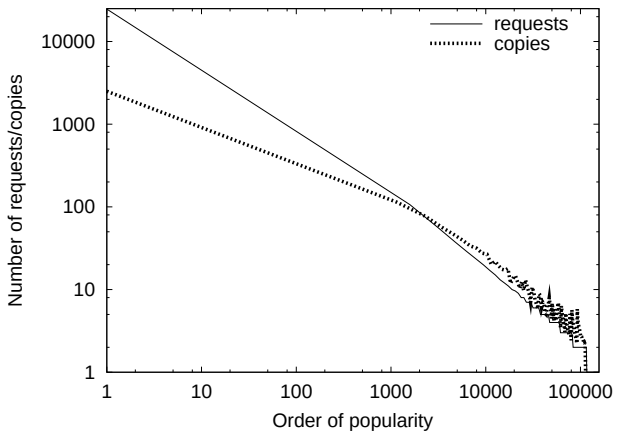

Fig. 4. The number of requests for, and copies of, each file strongly depends on the file popularity.

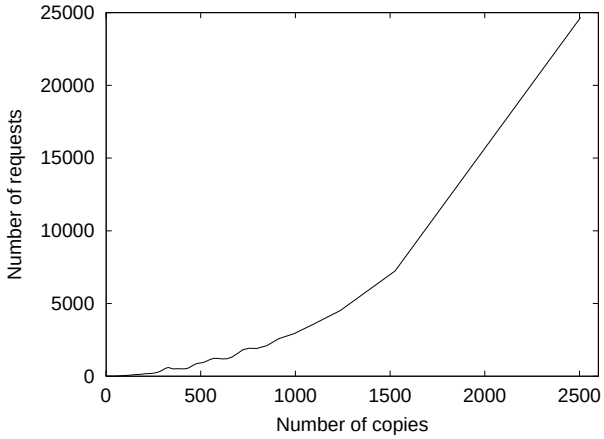

Fig. 5. The number of requests for a file is correlated with its number of copies.

based on the query traffic and the acquaintance replacement policy in use. The first phase ends when 80% of the storage capacity has been filled. At the end of the first phase, the storage and acquaintance link connectivity of the core network—composed on the serving peers that actually contribute to the content of the system—have been established. This corresponds to the expected state of a pre-existing peer-to-peer network at the time a new user connects.

2) During the second phase, we take measurements and observe the network's behavior under traffic load from both free-riders and serving peers (which fill up the remaining 20% of their storage capacity). This phase allows us in particular to observe the evolution of the connectivity of the free-rider population with respect to the serving peers in the core of the network.

We run the simulation for at least $1,000$ rounds in the second phase. Table I summarizes the main parameters used in our simulations.

Based on the content and query models, it appears clearly that the number of requests to each file, as well as the number of copies held in the system, are strongly correlated with the popularity of the file. Fig. 4 shows the number of requests and copies observed for each file based on its order of popularity. Fig. 5 further exhibits the strong correlation between the number of requests for a file and its number of copies.

V. $\mathcal{A}cquaintances$ Evaluation

In this section, we present and analyze the results of the experimental evaluation of $\mathcal{A}cquaintances$. We first start by studying the overall impact of acquaintances on our system. We then analyze search and load balancing improvements when each peer knows the state of some of its friends. Finally, we evaluate the effect of the dynamic TTL optimization.

A. Acquaintance Links

For all the experiments in this section, we assume that the peers have no knowledge of the state of their friends. This case corresponds to a traditional Gnutella-like network with no extra information being transmitted between peers.

1) Hops: The first metric used in our evaluation is the *number of hops* necessary to reach the first peer that serves the requested file. We compute the average over all successful requests issued during each round. The number of hops is a measure of the response time and allows us to choose adequate TTL values to experience a good query success rate without overloading the network.

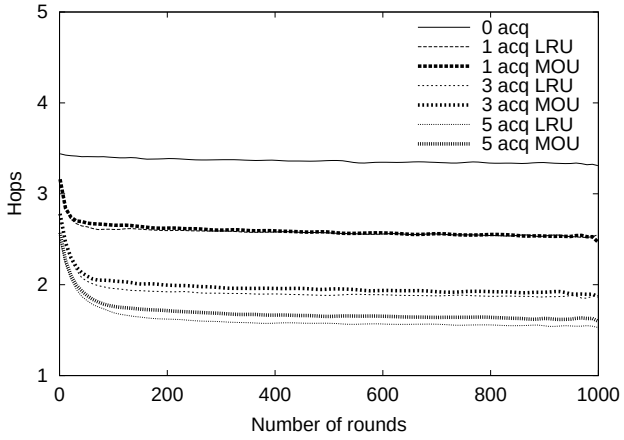

Fig. 6. Number of hops to the closest peer that serves the requested file.

Fig. 6 shows that the system, initially with random connectivity, needs only a few rounds to set up acquaintance links and stabilize in an efficient configuration. Regardless of the acquaintance replacement policy, the number of hops is reduced by around 30% with 1 acquaintance, by 60% with 3 acquaintances, and by 80% with 5 acquaintances. Note that the minimum number of hops necessary to reach a file is 1, as peers do not search for files that they already have. We have observed that more popular files are generally found closer to the requester; this behavior can be explained by the fact that popular files have more copies (see Fig. 4).

2) In-degree: The *in-degree* of a peer p is defined as the number of other peers that have chosen p as acquaintance. We have computed the maximum over all peers at the end of each round. As the number of queries received by a peer is clearly

proportional to its in-degree, it is important to keep this value within reasonable bounds.

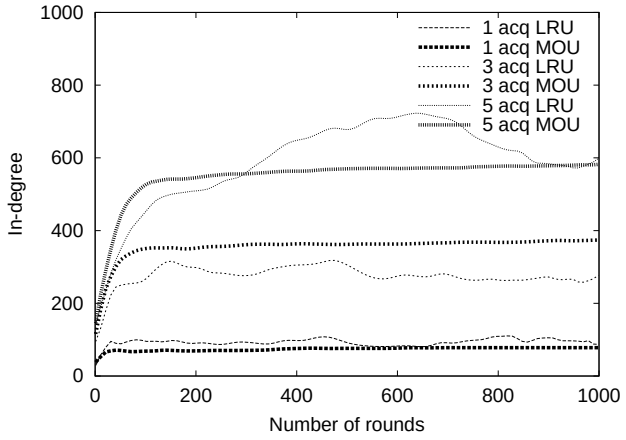

Fig. 7. Maximum in-degree over all the peers in the network.

Fig. 7 shows that, by introducing acquaintances, we increase the maximum in-degree in the network. This is not surprising as the peers that serve many files are more likely to be chosen as acquaintances by the other peers. Conversely, free-riders should be acquainted with almost no other peer. The *LRU* acquaintance replacement policy exhibits more volatility than *MOU*.

3) Promotions: To quantify the stability of links, we use the *percentage of promotions*, i.e., the proportion of successful requests that have induced a dynamic topology adaptation (update of the requester's acquaintance list). We compute the average over all successful requests issued during each round. A small value means that the connectivity of the system is stable.

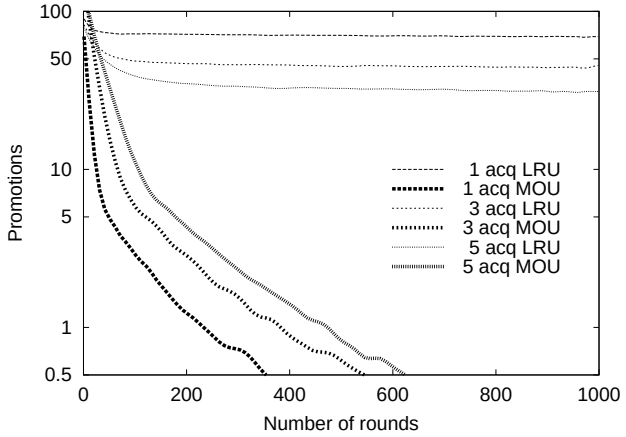

Fig. 8. Percentage of successful requests that yield an acquaintance promotion.

Fig. 8 shows that the *LRU* acquaintance replacement policy introduces much volatility, while the *MOU* policy yields a very stable network after a few rounds. Stability is particularly important when updates to acquaintance links have a significant connection management cost, or require extra messages to be transmitted between peers (e.g., to transfer state).

	Coefficient of variation of answered requests		# requests answered by the busiest peer	
# acq	LRU	MOU	LRU	MOU
0	2.3	2.3	5.5	5.5
1	3.1	3.0	10.0	9.7
3	3.7	3.6	11.5	12.7
5	4.1	3.8	13.8	13.5

TABLE II

EFFECT OF ACQUAINTANCES ON THE LOAD DISTRIBUTION.

4) Load distribution: In order to analyze how the introduction of acquaintances affects the load distribution of the system, we have computed the *coefficient of variation* of the number of requests answered by each peer (i.e., the standard deviation of the values divided by their mean). This metric shows how the file transfer load, which is a bandwidth- and time-consuming operation, is distributed among the peers. Small values indicate that the load is well balanced among the peers. Table II shows that acquaintances have a relatively small influence on the coefficient of variation, and consequently do not significantly degrade the load distribution of the system (when compared to a basic Gnutella-like network). The acquaintance replacement policy has no noticeable impact on the load distribution.

We also represent the *number of requests answered by the busiest peer*, computed over all peers during each round (remember that the query rate is 10%, which means that approximately $2,000$ requests are issued during each round). This metric helps us to identify hot-spots. Table II shows that, while the introduction of acquaintances leads to a higher request load on the busiest peer, this increase remains quite moderate and does not indicate the creation of hot-spots. The values show not influence from the acquaintance replacement policy.

5) TTL values: With TTL-limited flooding, the success rate of queries strongly depends on the chosen TTL value. Consequently, it is very important to choose a value that simultaneously provides a high success rate and limits the number of messages sent over the network.

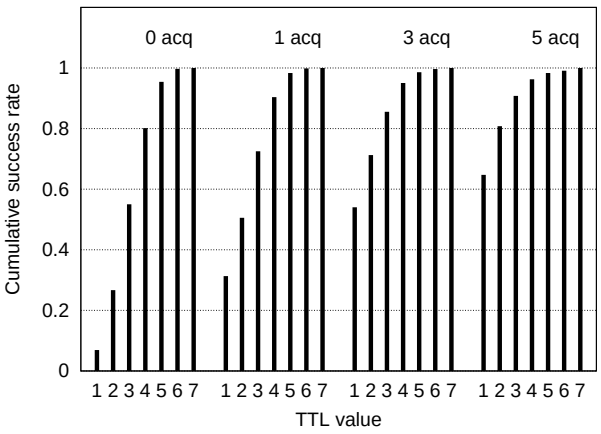

Fig. 9. Cumulative success rate, averaged over the last 800 rounds of the simulation, with a *MOU* acquaintance replacement policy.

The metric used to study the effect of the TTL value is the

cumulative success rate, i.e., the cumulative probability of success in the TTL-limited neighborhood of the requester. High success rates for small TTL values indicate that search is more efficient. Fig. 9 shows the results for various number of acquaintances. We can observe that more acquaintances leads to higher success rates for any TTL value. Without acquaintance, we need a TTL value of 5 to have a 90% success rate. With 5 acquaintances, we only need a TTL of 3 to get the same success rate, and a TTL of 2 still provides better than 80% success rate.

We have also measured the *cumulative number of hits*, i.e., the cumulative number of positive responses to a query in the TTL-limited neighborhood of the requester. We have computed the average over all successful requests issued during each round. Having several positive responses can reduce the download times as we can request a file from the least loaded or topologically-closest peer, or even use multi-source parallel download techniques [17].

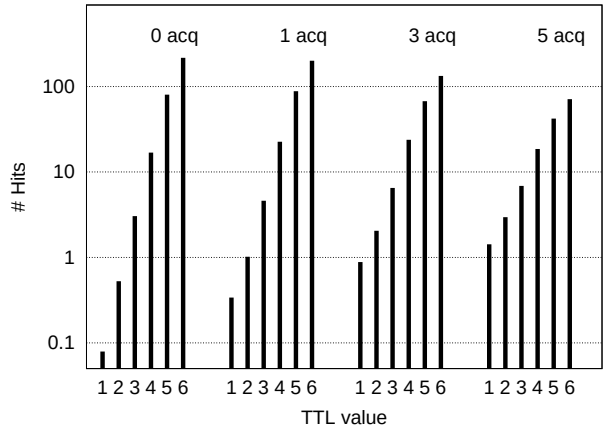

Fig. 10. Cumulative number of hits, averaged over the last 800 rounds of the simulation, with a *MOU* acquaintance replacement policy.

Fig. 10 shows that, for small TTL values, the number of hits increases with the number of acquaintances. This behavior results from the fact that acquaintance links are explicitly designed to connect to peers that have a high probability of serving the requested files. However, for high TTL values we observe an opposite behavior: the number of hits is higher when using less acquaintances. This can be explained by the fact that, when using more acquaintance links, we have less randomness and it becomes harder to find the extra copies of a file that are located outside of the requester's semantic communities. This problem is more severe when a peer searches for marginal content that does not belong to its semantic communities. Therefore, it is desirable to maintain a good balance of acquaintance and random links.

B. Friends Awareness

We now consider the case where each peer maintains an index of the files stored on its friends and uses this knowledge to answer to queries on their behalf, when possible. Our experiments show that, even with a single acquaintance, the number of hops needed to reach the first peer that serves, or has a friend that serves, the requested file drops to the optimal value 1 after a few rounds. This occurs regardless of the acquaintance replacement policy (*LRU* or *MOU*). As almost all requests are satisfied after a single hop, peers have no incentive to change their acquaintances; we have indeed observed that the number of promotions with both the *LRU* and *MOU* acquaintance replacement policies is almost null.

The experiments also show that the in-degree of the busiest peer grows quickly to almost attain the total number of peers in the system. This indicates that one peer acts as a central hub that all other peers choose as acquaintance; it is chosen initially because it serves many files, and later because it has many friends and consequently can answer to almost all queries. This snowball effect leads the system to spontaneously self-configure as a system with a central "index" peer, like Napster. The major difference with a centralized system is that, if the central index fails or leaves, the system quickly reconfigures and chooses another index peer.

A configuration with a central index has the major drawback of overloading the index peer (experiments show a significant degradation of the load distribution). In addition, the index peer must have enough resources to maintain the state of all its friends, i.e., the list of almost all the files served by all the peers in the P2P network.

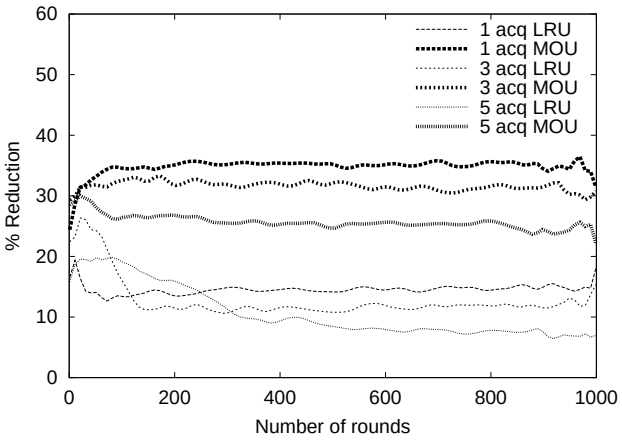

Fig. 11. Improvement of the average number of hops per request when maintaining the state of 25 friends (w.r.t. maintaining no state).

To overcome these drawbacks, we adopt a less extreme approach and we bound the maximum number of friends that a peer needs to keep track of. We have run simulations with this limit set to 25. Fig. 11 shows the improvement of the number of hops needed to reach the first peer that answers a query, with respect to the case where peers keep no state. We observe gains ranging from 8% to 16% with the *LRU* acquaintance replacement policy, and from 25% to 35% with *MOU*. The lower performance of *LRU* can be explained by the higher volatility of the network connectivity, which leads peers to only perform short-term optimizations.

Fig. 12 shows the increase of the maximum in-degree resulting from the introduction of friend state knowledge. This increase is pretty important with *MOU* (between 80% and 95%) but, as we shall see shortly, moderate enough to not cause hot-

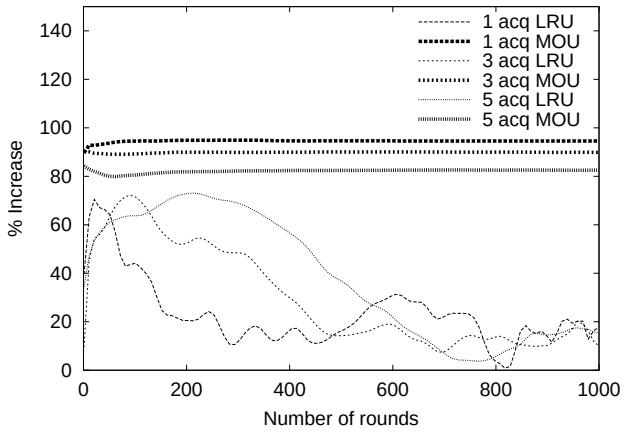

Fig. 12. Increase of the maximum in-degree when maintaining the state of 25 friends (w.r.t. maintaining no state).

	Coefficient of variation of answered requests (sent files)		# requests answered (# files sent) by the busiest peer	
# acq	LRU	MOU	LRU	MOU
0	2.4	2.4	5.8	5.8
1	3.0 (2.5)	4.1 (3.2)	9.7 (7.2)	30.9 (13.8)
3	3.4 (2.6)	4.3 (3.1)	11.8 (7.6)	36.7 (14.7)
5	3.9 (2.7)	4.7 (2.9)	14.7 (8.4)	27.5 (11.3)

TABLE III

EFFECT OF ACQUAINTANCES ON THE LOAD DISTRIBUTION WHEN MAINTAINING THE STATE OF 25 FRIENDS.

spots. With *LRU*, the increase is almost negligible. The evolution of the number of promotions is not shown, as it follows the very same trend observed in Fig. 8 (except for values being approximately 15% lower with *LRU*).

Table III shows that the load distribution of the system is not much affected by (compare with Table II). The increase of the number of requests answered by the busiest peer is moderate even with the *MOU* acquaintance replacement policy, which confirms that friends awareness does not cause hot-spots.

We have also analyzed the effectiveness of the load balancing technique described in Section III-D. The distribution of the file traffic in the system with the load balancing optimization is also shown in Table III (values are in parentheses). Note that, in that case, the peer that answers a query is not necessarily the one that sends the requested file. We observe low variation and moderate load on the busiest peer, which indicates that our techniques balance effectively the file traffic load.

C. Dynamic TTL

We finally evaluate the effectiveness of the dynamic TTL mechanism described in Section III-E, which decrements the TTL value twice when a query falls within the interests of the peer being traversed during query flooding. As queries require fewer hops to be satisfied within the scope of a semantic community, we expect this optimization to have little impact on the success rate while significantly decreasing the query traffic.

We have measured the *total number of query messages* sent across the network, and computed the average over all requests

	# messages					% faults		
# acq	static (TTL=6)	dyn (TTL=6)	$\frac{\text{static}}{\text{dyn}}$	dyn7 (TTL=7)	$\frac{\text{static}}{\text{dyn7}}$	static	dyn	dyn7
0	41312	4953	8.3	22524	1.8	0.27	10.47	0.03
1	33624	4217	7.9	18559	1.8	0.14	6.64	0.02
3	17613	2883	6.1	10267	1.7	0.36	4.76	0.02
5	6765	1709	3.9	4474	1.5	0.90	4.48	0.02

TABLE IV

NUMBER OF MESSAGES AND PERCENTAGE OF FAULTS PER REQUEST, AVERAGED OVER THE LAST 800 ROUNDS OF THE SIMULATION, WITH A *MOU* ACQUAINTANCE REPLACEMENT POLICY.

(successful or not) issued during the last 800 round of the simulation, after the network topology has stabilized. Table IV shows a important reduction of the query traffic when using the dynamic TTL optimization, by factors of 4 to more than 8. Note that adding acquaintances has the effect of decreasing the number of messages sent in the network, because well acquainted peers receive higher query traffic but forward each query only once. We have also performed experiments with the dynamic TTL optimization and a TTL value of 7 (instead of 6). Results show that query traffic is still reduced by factors of 1.5 to 1.8.

We have also measured the *query failure rate*, i.e., the proportion of requests that yield a negative result although the requested file is available in the system. Unsurprisingly, the percentage of failures with the dynamic TTL optimization is high (10%) with no acquaintances, because it is designed to take advantage of the semantic communities that are created by the acquaintance links. When using acquaintances, the failure rate decreases (4% to 6%) but remains significantly higher than with a static TTL value. In contrast, when increasing the TTL value to 7, the dynamic TTL optimization exhibits much lower failure rates (less than 0.1%). This optimization can thus at the same time improve the success rate and reduce the query traffic, when using an adequate TTL value.

VI. CONCLUSION

$\mathcal{A}cquaintances$ is a novel approach for improving search efficiency in unstructured P2P network. Its fundamental design principle lies in the dynamic adaptation of the network topology, driven by the history of successful requests, and achieved by having each peer maintain a list of acquaintances that are likely to best answer queries. Acquaintance links connect peers that share similar interests and spontaneously build semantic communities. They provide a short path to content that belongs to the core interests of a requesting peer. To guarantee some diversity and help find more marginal content, each peer also maintains a set of random neighbors. This combination of semantic and random links provides efficient, yet robust, search facilities to unstructured P2P networks.

Query forwarding is implemented by the same TTL-limited flooding mechanism found in Gnutella-like P2P file sharing systems. $\mathcal{A}cquaintances$ does therefore represent a non-intrusive extension to legacy P2P networks, where each peer modifies the network topology by locally optimizing its connectivity. It also incorporates load-balancing mechanisms that offload potential hot-spots in popular semantic communities, as

well as a dynamic TTL optimization that further reduces the network traffic. Experimental evaluation has shown that our techniques are effective at improving search efficiency. Optimizations to the actual search algorithm, such as random walks [10], are orthogonal to our techniques and could thus be used to further improve the efficiency of $\mathcal{A}cquaintances$.

REFERENCES

[1] "Napster," `http://www.napster.com`.

[2] I. Stoica, R. Morris, D. Karger, M. Kaashoek, and H. Balakrishnan, "Chord: A scalable peer-to-peer lookup service for internet applications," in *ACM SIGCOMM*, Aug. 2001.

[3] S. Ratnasamy, M. Handley, R. Karp, and S. Shenker, "A scalable content-addressable network," in *ACM SIGCOMM*, Aug. 2001.

[4] 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)*, Nov. 2001.

[5] B.Y. Zhao, J. Kubiatowicz, and A.D. Joseph, "Tapestry: An infrastructure for fault-tolerant wide-area location and routing," Tech. Rep. UCB/CSD-01-1141, Computer Science Division, University of California, Berkeley, Apr. 2001.

[6] "Gnutella," `http://gnutella.wego.com`.

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

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

[9] B. Yang and H. Garcia-Molina, "Improving search in peer-to-peer systems," in *International Conference on Distributed Computing Systems (ICDCS)*, July 2002.

[10] C. Lv, P. Cao, E. Cohen, K. Li, and S. Shenker, "Search and replication in unstructured peer-to-peer networks," in *ACM International Conference on Supercomputing (ICS)*, June 2002.

[11] Y. Chawathe, S. Ratnasamy, L. Breslau, N. Lanham, and L. Breslau, "Making gnutella-like P2P systems scalable," in *ACM SIGCOMM*, Aug. 2003.

[12] E. Cohen, H. Kaplan, and A. Fiat, "Associative search in peer to peer networks: Harnessing latent semantics," in *INFOCOM*, Apr. 2003.

[13] A. Crespo and H. Garcia-Molina, "Semantic overlay networks for P2P systems," submitted for publication, 2003.

[14] S. Saroiu, K.P. Gummadi, and S.D. Gribble, "A measurement study of peer-to-peer file sharing systems," in *Multimedia Computing and Networking (MMCN)*, Jan. 2002.

[15] K. Sripanidkulchai, "The popularity of gnutella queries and its implications on scalability (http://www.openp2p.com)," Feb. 2001.

[16] E. Adar and B.A. Huberman, "Free riding on gnutella," *First Monday*, Sept. 2000.

[17] P. Rodriguez and E.W. Biersack, "Dynamic parallel-access to replicated content in the internet," *IEEE/ACM Transactions on Networking*, vol. 10, no. 4, 2002.