Efficient processing of rare queries in Gnutella using a hybrid infrastructure*

Mark Kornfilt[1] and Manfred Hauswirth[2]

[1] LimeWire LLC
[2] DERI Galway

Abstract. Gnutella is still one of the most popular P2P systems with millions of users. The advantages of Gnutella are its low maintenance overhead, its excellent robustness, and its query processing flexibility. Recent improvements, such as the introduction of ultrapeers and augmented node degrees significantly reduced its excessive network bandwidth usage which was one of Gnutella's major drawbacks. Despite these improvements, Gnutella is still inefficient for rare queries in terms of low success rates and massive message propagation overhead. In this paper we augment the unstructured Gnutella network with a structured overlay network of ultrapeers based on the Kademlia DHT to address the problem of rare queries in Gnutella. We present the query, maintenance, and ultrapeer election algorithms which use both overlays at their optimal efficiency, describe the protocols and architecture of our hybrid system, and present our implementation on the basis of the LimeWire Gnutella client and the Azureus Kademlia implementation. To demonstrate the advantages and efficiency of our hybrid approach we provide experimental results from large-scale experiments with hybrid ultrapeers running on PlanetLab which were connected to the live LimeWire Gnutella and Azureus Kademlia networks, with approximately 4 million (LimeWire) and 800 thousand (Azureus) connected users during the experiments.

1 Introduction

Recent P2P research has focused to a large extent on structured systems, most prominently DHTs which offer a very high search performance and low bandwidth overheads at the cost of having to use sophisticated protocols to deal with churn and still limited expressiveness of supported search predicates. In contrast, unstructured overlays, most prominently Gnutella, are very robust and offer flexible support for query processing, but pay these advantages with excessively high bandwidth consumption and low success rates and massive message propagation overhead for rare queries. To address these problems of Gnutella, we propose a hybrid overlay network which combines the advantages of both worlds.

Before we take a closer look at our proposal, it is important to review the development of the Gnutella network as many improvements have been introduced over the years and several assumptions which were correct for the original Gnutella overlay of 2001 no longer hold, despite still being used as the basis of most work on Gnutella.

* The work presented in this paper was supported (in part) by the National Competence Center in Research on Mobile Information and Communication Systems (NCCR-MICS), a center supported by the Swiss National Science Foundation under grant number 5005-67322 and (in part) by the Líon project supported by Science Foundation Ireland under Grant No. SFI/02/CE1/I131.

Most importantly, the topology and performance of the Gnutella network have evolved. In respect to topology a super-peer architecture of higher-layer ultrapeers and lower-layer leaf nodes was introduced. The most popular clients (LimeWire, Bearshare), which account for more than 90% of the network, use this architecture and enforce a constant number of open connections between the clients and between the ultrapeers and the leaf layer. This results in a flatter node degree distribution so that the node degrees can no longer be assumed to follow a power-law distribution. Additionally, this architecture makes the Gnutella network even more resilient to failure.

Furthermore, the user base has grown considerably: Since its original conception, the Gnutella network has evolved to more than 4 million simultaneous users. But thanks to the introduction of the two-tier topology, dynamic querying, and query routing protocol (QRP) improvements, the Gnutella network has scaled to match this substantial growth of its user base. Ultrapeers also suppress unnecessary maintenance network traffic as leaves no longer participate in the continuous ping-pong interactions to discover peers, thus the required bandwidth overhead for maintenance was significantly reduced.

The augmented node degree and dynamic querying have maintained the desirable characteristics of Gnutella's message flooding. The network crawls described in [1] show that the number of peers reached per hop is stable compared to the original studies [2–4] which were performed when the network was considerably smaller, and that the prediction done by the dynamic querying mechanism is very accurate up to a certain threshold.

Despite all these improvements which have reduced the resource consumption considerably, some drawbacks of Gnutella remain, such as no upper bound on query latency and the inefficient processing of rare queries. While the latency bound is an unsolvable theoretical problem, rare queries can be made more efficient by practical means. For this purpose we propose to augment Gnutella with a structured overlay network of ultrapeers based on the Kademlia DHT [5].

In the following we first give a concise description of the current protocols and optimizations used in the Gnutella network and give a detailed problem description to enable the reader to assess the advantages of our hybrid approach. In contrast to most of the related work we base our approach on the latest Gnutella version, thus taking into account the considerable improvements Gnutella has already undergone. Still, as we will show in this paper, our approach can improve considerably on several of Gnutella's disadvantages. As the motivation and practical basis of the work presented in this paper, we then provide a discussion how rare queries and diminishing return can be characterized and identified in the current Gnutella infrastructure and back up our findings with experimental data from the live Gnutella network. Then we present the query, maintenance, and network election algorithms that provide the best efficiency depending on whether a query is popular or rare. Our approach constantly monitors the popularity of queries, detects rare queries and uses the Kademlia DHT [5] to answer rare queries, while popular queries are answered using the normal Gnutella infrastructure. We will show that this hybrid system not only provides reliable search results, but also considerably decreases the bandwidth overhead in the Gnutella system created by the message flood induced by highly propagated searches. We will discuss the specific problems of detecting and managing rare data in Gnutella's distributed environment and show how our mechanisms can be integrated into the existing Gnutella infrastructure. Finally, the efficiency of our approach is demonstrated by large-scale experiments with hybrid ultrapeers running on PlanetLab which were connected to the live LimeWire Gnutella and Azureus Kademlia networks, with approximately 4 million (LimeWire) and 800 thousand (Azureus) connected users during the experiments. The experiments were done with a production-quality implementation which will be included in the LimeWire P2P software.

2 Current Gnutella technology

Gnutella was one of the first completely decentralized P2P systems and it has been evolving constantly since its original conception. The initial, primitive version of the protocol has been extended and augmented to address several shortcomings such as excessive bandwidth consumption and query delays. The improvements include a super-peer topology (ultrapeers), query routing, and dynamic querying. In the following we briefly present these additions to give an up-to-date picture of the currently deployed Gnutella infrastructure.

A significant improvement to the original model is to create a hierarchy within the network, partitioning the peers into leave nodes and super-peers, called ultrapeers in Gnutella. The goal is to reduce bandwidth consumption without compromising Gnutella's robustness. Ultrapeers are connected as in the original Gnutella while leaves are not part of this network but are connected to at least one ultrapeer which shields them from undesired traffic and handles the query processing for them. An ultrapeer has multiple leaves and is connected to multiple other ultrapeers. LimeWire's implementation currently uses 3–5 ultrapeer connections for each leaf and each ultrapeer services up to 32 leaves and has connections to 30 other ultrapeers. Ultrapeers are selected based on long uptime, higher bandwidth, and reachability (not behind a firewall) of a peer. For uptime it has been shown in [4] that the probability that a host stays online is directly related to how long it has been connected to the network. Hosts should therefore have a reasonably high uptime to become ultrapeer candidates. When a new node joins the network, it receives a list of potential ultrapeers to try to connect to. Each node also keeps a list of ultrapeers it has encountered through pong replies. If a leaf loses a connection to one of its ultrapeers, it will try to connect to another node in this list.

To fully exploit this topology, ultrapeers require some knowledge of the data their leaves expose to the rest of the network. To this end, leaf peers periodically send a set of hashes of their data to the ultrapeer. This set of hashes is called a *query routing protocol table* (QRP table). When an ultrapeer receives a data query, it checks its QRP tables and forwards the query only to those leaves which have a potential match.

In the original Gnutella specification, query responses were routed back to the originator along the path of the query. This uses significant bandwidth and increases the probability that messages are lost. To address this problem, search results can now be returned directly to the query originator. This so-called *out-of-band messaging* requires that a host can accept unsolicited UDP packets, which is not always the case. A vendor-specific flag has thus been added in the query message to inform the responding peers if the query originator can receive out of band responses. If a leaf cannot receive out of band messages then its ultrapeer can act as a proxy.

To further reduce bandwidth consumption also the concept of *dynamic querying* was introduced. The underlying idea is that a leaf first sends a probe query to a subset of its ultrapeers to estimate the popularity of the query and based on the number of returned hits, it either sends a regular query to some of its ultrapeers or uses a more aggressive search strategy with $TTL \geq 2$ to a larger number of ultrapeers (similar to expanding ring search). This strategy makes a lot of sense in file-sharing applications as users are typically not interested in a complete result but in a reasonable number of hits they can use for downloads.

The popularity of a query is calculated as the ratio of returned hits vs. the number of contacted peers. The number of contacted peers can be estimated by $\sum_{i=0}^{TTL-1}(d-1)^i$, where d is the ultrapeer nodes' out-degree (all ultrapeers are assumed to have the same d). Depending on this ratio, three scenarios are possible:

1. If the ratio is low, the query is considered rare and sent again with a high TTL.
2. If the ratio is medium, the query is sent to a bigger number of ultrapeers with a low TTL.
3. If the ratio is high, the search stops.

For even finer-grained control of query flooding LimeWire uses a *time_to_ wait_per_hop* variable which determines the aggressiveness of the search in terms of the time to wait before sending the next query (flow control), i.e., it sends the query with *TTL=3* and then waits *time_to_wait_per_hop* before sending the next query with a higher TTL. The new TTL is calculated with respect to the ratio but in LimeWire is never greater than 6. Again, this timeout is fine-tuned depending on the query popularity.

Because leaves are constantly connected to at least 3 ultrapeers who perform dynamic querying and because replies to those queries can be sent out-of-band without coming back through the ultrapeers, a notification mechanism has been introduced: When a leaf has received sufficient results, it sends a `QueryStatusResponse` message to its ultrapeer which then considers the query as completed and discards it.

3 Rare queries and diminishing return

Message flooding as used in unstructured networks works well for discovering popular data because a query can be propagated to a large number of nodes with a relatively low TTL, i.e., low number of hops [3, 6], and popular data items have a high replication factor in the network [7]. However, for rare items, message flooding in unstructured networks performs poorly as it consumes a large amount of bandwidth due to the large number of messages flooded into the network, queries have a high latency as it increases which each hop until a hit is found, and it is unreliable as the search has a low probability of reaching a host which has the required data.

In contrast, DHTs offer a very good search performance, typically $O(\log n)$, for any data item in the system independent of its popularity and bandwidth consumption is low. However, DHTs require sophisticated protocols to deal with network dynamics (churn) and still only support queries of limited expressivity. Moreover, DHTs are commonly not optimized for mass-market file sharing applications, where most requests are for a small number of very popular files and where network churn is extremely high. Therefore, the rational of the approach presented in this paper is to use Gnutella as the basic communication infrastructure to connect peers and perform popular searches, and to use a DHT to publish and query for rare items.

3.1 Defining rare data items

The basic design question for such a hybrid system is, how rare data items are defined. Previous studies [8, 9, 7, 6] have evaluated the data distribution in Gnutella but have mostly used simulations or produced artificial queries to measure query replies and result sets. Moreover, these studies have focused on network characteristics such as overall query and file distribution across connected peers. In contrast to these studies we need user-centric statistics in order to gain knowledge of individual peer behaviors.

To this end, we had to perform a number of experiments to collect the required statistics. We deployed 50 ultrapeer nodes on PlanetLab [10] and linked them into the live LimeWire network. These nodes ran a modified LimeWire client and recorded queries and corresponding results in the Gnutella network. More precisely, we used ultrapeer probes with a custom implementation of the LimeWire core and performed passive measurements, i.e., measurements that did not interfere with the network by actively generating messages. In the experiments we recorded more than 100'000 incoming queries which produced over 4.5 million results. Three sets where produced by the probes on 19/01/2006, 21/01/2006 and 29/01/2006, recording queries for 1 day in the first experiment and for 2 days in the other two experiments. To normalize the statistics, the following changes were applied to the data sets:

1. Queries that did not complete because the leaves disconnected from the ultrapeer before the end of the search, were discarded, as this does not provide relevant information.
2. Leaves have 3–5 open ultrapeer connections. Even though the probes have been deployed in dispersed locations around the world, some leaves connected to more than one probe at a time. This led to duplicate entries in the data set. These were identified and removed.
3. When leaves receive enough results (150 in LimeWire), they notify their ultrapeers to stop querying the network by `QueryStatusResponse` as described above. We have therefore replaced the value of the result set of these queries with the following formula: $f(\gamma) = \frac{150-\gamma}{d-1}$, where γ is the result set size currently recorded by the probe and d is the out-degree of leaves. This formula calculates the average number of results returned by each of the leaf's ultrapeers. As the leaf has received 150 results in total, it means it has received $150-\gamma$ results from its other ultrapeers, which we finally divide by the out-degree minus the probe to find the average number of results routed per ultrapeer.

3.2 Experimental results

Query latency and result set size. Our first goal was to gather information on the query popularity of individual queries to assess the possibility of improvements to the current network. Figure 1 shows the cumulative distribution function (CDF) of query times recorded in the experiments.

A query stops only if it was successful and has generated enough results, or in the case of a failure, when the maximum search time has elapsed, i.e., 200 seconds in LimeWire. The first observation from this data set is the discontinuity in the query times which is due to the dynamic querying mechanism as it adapts the TTL of the search message depending on the query popularity. Searches are first sent to a small set of peers, and the search horizon is then increased progressively if necessary. This creates the waves of results in the figure. The second observation, which is the most relevant for our purpose, is that 80 percent of the queries are successful before 120 seconds while approximately 18 percent of searches are killed and never get enough results.

Additionally, as shown in Table 1, successful queries generally have a good mean response time of 15.958 seconds for the first response and a mean result set size 94.04. The corresponding figures for queries which returned insufficient result set sizes are 138.149 seconds and 13.09 hits (these number does not include queries that did not return any result).

These data indicate that the Gnutella network is very efficient in finding the majority of data, providing quite large result sets in a small amount of time, but that almost 20% of the queries could benefit from an improved resource location mechanism such as the one we propose in this paper.

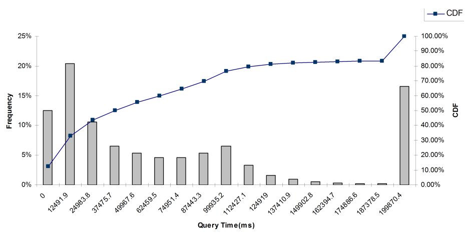

Fig. 1. CDF of query times

Table 1. Result set size and first result latency with respect to the query outcome

Query outcome	Mean result set size	mean first response latency (sec)	Total (%)
successful	94.04	15.958	81.76
failure	13.09	138.149	18.24

4 Hybrid Gnutella Topology

In order to optimally exploit both network types, it is important to use them in situations where they perform best: Gnutella is efficient with respect to high network churn, popular files, and range queries, whereas DHT resource location algorithms are extremely efficient in finding exact matches for data keys in the network, but generally incur high efforts to deal with churn. It was therefore clear that we had to exclude a considerable subset of the unstructured network, and use the DHT only with those peers that could provide a higher degree of stability. As this coincides with the requirements for ultrapeers, we decided to only use ultrapeers in the DHT.

In the resulting topology, leaves have no access to the DHT, and only a subset of stable ultrapeers are connected to the structured network. The goal is to connect only the innermost layers of Gnutella's onion-like overlay, as described in [1]. This is achieved by ensuring that only ultrapeers with a sufficiently high uptime connect to the hybrid network. Stable ultrapeers ensure that the maintenance traffic in the DHT is minimized.

As only a subset of the ultrapeer population is stable enough to participate in the DHT we need to gather and disseminate stability characteristics to enable the system to find good candidates. In addition to all the required characteristics detailed in Section 2 to become an ultrapeer in the Gnutella network, we define an additional variable, `DHT_CAPABLE`, maintained by each peer, which takes into account the current session and average session uptimes. Following the study in [1], we propose an initial minimum session uptime of 24 hours in order to ensure that the ultrapeer is part of the stable core of the unstructured network.

The rest of the Gnutella network does not participate in the DHT and is only allowed to interact with it to query and publish (rare) data. Therefore, the addresses of the DHT nodes need to be announced in the Gnutella network, which we achieve by extending the existing ping-pong scheme, such that the pong message carries an additional vendor message part containing the sender's participation status in the DHT. This method does not add any additional overhead to the network, as we just add a small piece of information to standard ping-pong interactions which also does not break compatibility as we follow the recommended procedure for extensions to Gnutella.

Consequently, regular ultrapeers hold an additional routing table of hosts connected to the DHT to which they can route DHT-related messages. We introduce two new vendor messages into the Gnutella protocol: `DHT_QUERYREQUEST` and `DHT_STOREREQUEST`, which are used by leaves and non-DHT ultrapeers to interact with the structured overlay.

When a peer wishes to query the DHT, it sends a Gnutella `QueryRequest` message encapsulated into a `DHT_QUERYREQUEST` message to a DHT peer it knows. If the peer sending the query is a leaf node shielded by ultrapeers, it sends the message to one of its ultrapeers which forwards it to a DHT node. Subsequent interactions, such as returning results or downloading data, are done through the standard Gnutella protocol. The `DHT_STOREREQUEST` is used to insert rare data into the DHT and is handled in the same way as the `DHT_QUERYREQUEST` message.

5 Hybrid Resource Location

Our hybrid system strives to provide better resource location by combining the advantages of unstructured and structured networks. However, this can only be achieved by relying on techniques that optimize the use of both networks depending on query popularity. In the following we propose algorithms that can be applied in a hybrid system based on the Gnutella network. We do not provide exact tuning parameter values for the variables introduced in these algorithms, due to the fact that the optimum behavior of the overall system has to be determined through a large-scale deployment and continuous empirical studies which are on the way at the moment.

5.1 Dynamic Querying

The dynamic querier in LimeWire tries to efficiently locate resources in the unstructured network. It trims the message flooding to match query popularity and controls the aggressiveness of a search. It is hence straightforward to integrate the DHT search into the dynamic querier. Listing 1.1 shows the hybrid search algorithm we use at ultrapeers participating in the DHT:

```
for queries in dynamic_querier{
        while (t < QUERY_TIMEOUT){ // the dynamic querying timeout
                if(!forwardedToLeaves){
                        forwardQueryToLeaves(); // first send query to our direct leaves
                        forwardedToLeaves = true;
                }
                else if(!sentProbeQuery){
                        sendProbe(); // then send probe query to estimate data availability
                        sentProbeQuery = true;
                }
                else if((!queriedDHT) AND
                        ((t > FIRST_T_DHTLIM AND resultSet == 0) OR
                         (t >= T_DHTLIM  AND resultSet < RESULTSET_DHTLIM)) {
                                sendDHTQuery();
                                queriedDHT = true;
                        }
                }
                else{
                        sendDynamicQuery(); // An adaptative-TTL query is sent in Gnutella
                }
        }
}
```

Listing 1.1. Hybrid dynamic querying algorithm

The dynamic querier starts by sending the query to all the ultrapeer's direct leaves. Next, it dispatches the probe query, which enables it to estimate the data availability for this query. If a certain time expires, which was allocated for standard Gnutella queries to return meaningful results, i.e., the query is supposed to be rare, it queries the DHT for results. The criteria when to query the DHT are derived from our empirical studies presented in Section 3.2 and are as follows:

- Our empirical studies show that more than 99 percent of the successful queries get their first result before 100 seconds. We therefore decided to start querying the DHT if no results have arrived before that time. To that end, we introduce a `FIRST_T_DHTLIM` constant.
- Only searches that did not return enough result within `T_DHTLIM` time are taken into consideration. This timeout is set to 120 seconds. The minimum result set size

is determined by RESULTSET_DHTLIM, which we set to 23, as it is the average size for unsuccessful queries in our experiments.

For this algorithm to work we require the leaves to implement the standard QueryStatusResponse synchronization mechanism to notify their ultrapeer if they received enough results through out-of-band replies.

5.2 Managing rare data in the DHT

Publishing rare data in the DHT is a non-trivial problem, as individually at each peer there is no a-priori knowledge on the availability of a particular file in the network. As the hybrid system only uses the DHT for rare files, it is also not an option to systematically allow every host to publish information about its entire data library. Thus we propose the following techniques for publishing data to the DHT:

Client-based publishing. The first mechanism for client-based publishing we suggest is to associate a counter with each data item shared by a peer. This counter is persistent over sessions and counts the requests received for a data item over a period of time to assess its popularity. When the client detects that the demand for a particular item is low, it sends a DHT_STOREREQUEST to store it in the DHT layer.

The second mechanism for client-based publishing is a two-step process that relies on file downloads. When a peer downloads a rare file, i.e., with a low number of location, it adds itself to the DHT as an additional location using DHT_STOREREQUEST. If the location the file was downloaded from did not exist in the DHT yet, the peer also inserts the original location into the DHT. Additionally, to comply with Kademlia's specification, we require every peer to republish its data every hour so that the DHT can expire values for disconnected hosts after this timeout. When a host starts a new session with a different IP and port, it republishes all its data, such that the new values erase the previous ones.

Network-based publishing. Ultrapeers have two opportunities to detect and publish rare data items in case the search is proxied. The first is after the dynamic querier stops because it did not get enough results before the timeout or because it contacted too many hosts. In this case the dynamic querier iterates through the list of the replies that have arrived and publishes rare files as shown in Listing 1.2.

```
if((t > QUERY_TIMEOUT) AND (resultSet < RESULTSET_DHTLIM)){
        foreach response in query.responses{
                if(isSpam(response)) continue; // don't publish spam responses
                else {
                        if (response.numberOfLocations < NUMLOCS_DHTLIMIT) // low file availability
                                publishDHTFile(response);
                }
        }
}
```

Listing 1.2. Post-query publishing algorithm

The only additional variable introduced in Listing 1.2, with respect to the hybrid resource location algorithm, is NUMLOCS_DHTLIMIT. It ensures that replication of the data to be inserted is really low and is an additional protection against spam, as malicious nodes that systematically answer queries also fake a large number of available locations. We use an initial value of 2 for that variable (based on heuristics).

The second opportunity to publish rare files it to detect late responses. When a query is unpopular, it incurs a large number of hops before reaching a host that has the data. Consequently, some responses may arrive after the dynamic querier stops, and must be intercepted and published in the DHT using the same conditions as above.

Finally, as a prerequisite for both techniques, the ultrapeer has to verify that the file it wants to publish is not already in the DHT. As the resource location algorithm presented in Listing 1.1 queries the DHT for rare items, it is straightforward to know if the element that is going to be published is already stored in the structured overlay or not. For each search, the ultrapeer therefore keeps track of the responses received from the DHT and compares those with the file it wishes to publish before doing so.

5.3 Removing popular data from the DHT

The popularity of a file in the DHT can be measured by the number of times it has been downloaded. With client-based publishing, peers search the DHT for a file, download it, and add themselves to the list of locations available for that file. Therefore, it is easy to detect and remove entries in the DHT which have become popular: If the number of locations for a DHT entry goes beyond a threshold (`NUMLOCS_LIMIT`), this means that the file has in fact become popular and thus is no longer to be considered a rare data item and can simply be removed from the DHT. In our hybrid implementation, this is accomplished by storing an empty value for a given key, i.e., the corresponding $\langle key, originatorNode\rangle$ entry in the DHT is deleted.

6 Experimental evaluation

To fully evaluate our system, a large-scale deployment of our hybrid client would have been required. As this was not feasible, the goal of our evaluation has been to simulate queries for rare data items available in the Gnutella network and in the DHT and analyze both networks' characteristics and behaviors. Our evaluation was therefore focused on the efficiency of our hybrid algorithms in detecting and publishing rare items in the structured overlay.

In the experiments we deployed 50 hybrid ultrapeers on PlanetLab [10]. Each ultrapeer ran on a dedicated PlanetLab node. Then the ultrapeers were connected to the live LimeWire Gnutella network (approximately 4 million users during the experiments) and the Azureus Kademlia network (approximately 800 thousand users during the experiments). Then we used the network-based publishing algorithm presented in Section 5.2 to publish rare data items returned in the responses of queries coming from the LimeWire leaves connected to our hybrid ultrapeers. After a few hours—to receive sufficient amounts of rare data items—we issued queries for rare files by iterating through rare files published in the DHT, simultaneously querying the Gnutella network and the DHT, and recording the latency of the search for both networks.

In the setup of the experiments we also considered the following issues:

Data availability: As the DHT only indexes rare data but does not store the corresponding physical files, a search may succeed but the storing peers may be offline. In Gnutella in contrast, only nodes that have the queried file respond to a query. To make a fair comparison between both networks, thus we had to ensure that the nodes holding rare files indexed in the DHT were still online during the test. To that end, we systematically sent a ping message to every host before starting the tests to verify their availability. As firewalled hosts are shielded from this kind of traffic, we could not include them into our evaluation.

Node availability: In LimeWire's implementation, a node does not answer queries in the case that it cannot upload the data, for example, when the node already has too many open connections or when the node only has parts of the data item. Thus ping messages were not sufficient to verify that the node could respond to a query. To get around this problem we additionally used LimeWire's proprietary `HEADPING` and `HEADPONG` messages, where the latter contains information about the availability of the file and the node.

Query trimming: Our first evaluation showed a linear increase in the DHT's response time. That was due to the fact that each hybrid ultrapeer was starting thousands of queries simultaneously on the DHT, and that each query performs multiple lookups in parallel. In order to correct this, we enforced a 5 seconds break between the queries, in order not to overload the system.

Unbiased routing tables: After the collection phase of rare data items, we reinitialized the Gnutella and Kademlia routing tables of each hybrid ultrapeer before starting the query experiments. This ensured that the routing tables were not biased towards contacts that had already been seen while intercepting queries and publishing data items.

The results presented below have been recorded on 08/02/2006, 09/02/2006, 11/02/2006, 12/02/2006, 14/02/2006 and 15/02/2006.

For the Gnutella overlay the query success rate was 27% with a mean query latency of 75989ms, while the DHT had 99% success rate with a mean latency of 3878ms.

Although we only queried for rare data which was available in the network (data and node availability was ensured as described above, Gnutella could only find 27 percent of these data items, whereas the DHT had a success rate of 99 percent. The missing 1 percent is due to individual node failures in republishing data in the DHT. Moreover, the search latency for the DHT is approximately 20 times lower than Gnutella's. As shown in Figure 2, more than 50 percent of the answers from the DHT come in less than a second, even though 800'000 nodes participated in the DHT during our tests (the figure shows only the first 5 seconds of the full plot which was the most interesting interval for us; thus the CDF does not reach 100%).

These results prove that our algorithms were successful in identifying and publishing rare data items, and show the potential gain in success rate and latency. As expected, they demonstrate Gnutella's unreliability and inefficiency in finding rare items.

In order to evaluate the influence of the parameters `RESULTSET_DHTLIM` (result set threshold for unsuccessful queries) and `NUMLOCS_DHTLIMIT` (replication threshold) discussed in Section 5, we split the hybrid ultrapeers into four groups with different combinations of these parameter values: Group 1 – (10, 1), Group 2 – (20, 1), Group 3 – (30, 1), and Group 4 – (30, 3).

Figure 3 shows the search success rates for Gnutella for the different groups. The figures indicate that relaxing the parameters that select rare files directly affects the efficiency of the hybrid platform. This test also demonstrates the importance of fine-tuning the hybrid algorithm's parameters in order to stop the Gnutella network's query message flooding at the optimal time.

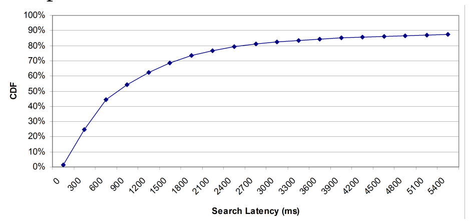

Fig. 2. DHT search latency

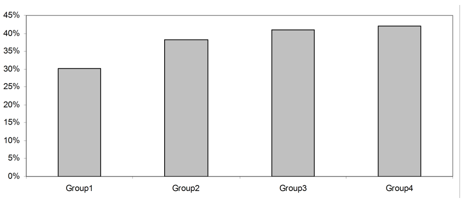

Fig. 3. Success rates for different parameters

The bandwidth overhead in the hybrid approach mainly consists of the bandwidth required to insert data items into the DHT, i.e., the lookup costs to find the node to store the value plus the *put* message's size. The lookup cost is a function of the network's size and the degree of parallelism of the system, which is represented by the α parameter in Kademlia. If for each hop, α nodes are contacted ($\alpha = 4$ in Azureus) and we need approximately 20 hops to reach the target in a network of 800'000 connected Kademlia nodes, a rough estimate of the number of required lookup messages is 80 messages of size 297 bytes (41 bytes header + 256 bytes data), i.e., 23760 bytes overall in the DHT per insert operation. Querying produces a similar message load.

By analyzing the statistics from our plain Gnutella ultrapeers, we see that the average number of nodes queried for searches that last less than 100 seconds is 73'710, while the average number of nodes queried for searches that last more than 100 seconds is 697'050, i.e., use 10 times more bandwidth. Consequently, if the longer queries are performed through the DHT, we should be able to reduce the message flooding in the network by approximately an order of magnitude. Therefore, even after adding maintenance, publishing and querying, the DHT can potentially be of great benefit to the Gnutella network.

Finally, load balancing in the hybrid system is automatically done by removing overly popular queries from the DHT. With the algorithms presented in Section 5.2, only items belonging to the tail of the file distribution in the Gnutella network are inserted in the DHT. Consequently, these items should never generate a high demand at a single node.

7 Related Work

Our approach extends the case for a hybrid search infrastructure made in [7]. This paper provides an initial proof-of-concept proposal for a hybrid system, but does not define the topology of the hybrid Gnutella network and the required interactions to an extent necessary to assess the practical applicability of the proposal. Furthermore, it does not take into account the recent Gnutella improvements which we use to greatly improve on some of the described problems. Our contributions beyond the work described in [7] are:

Our proposal tightly integrates hybrid querying and data publishing algorithms into the most recent Gnutella specification and provides a real-world, large-scale, experimental evaluation of the algorithms in a live Gnutella network to back up our claims. By exploiting the hybrid dynamic querying mechanism, rather than using a fall-back approach as proposed in [7], we not only improve the search delay but also effectively reduce the bandwidth of Gnutella's message flooding technique by an order of magnitude. Furthermore, the "selective publishing" technique proposed in [7] cannot be used by itself due to the redundancy and transience of Gnutella's ultrapeer-to-leaf connections. In contrast, we propose a client-side data publishing algorithm that monitors the data and behavior of peers in order to discover rare data items.

We extend the Gnutella protocol and the ultrapeer election techniques to be able to selectively build the DHT, and provide the nodes in the unstructured network with the means to discover and bootstrap to the structured network.

We provide a detailed description of the subsequent interactions between the structured and the unstructured overlay in order to achieve a stable and scalable system. Additionally, we address the possibility of a surge in the popularity of a data item and propose techniques to deal with that situation.

We address the problem of spam in the Gnutella network which can seriously impact on the detection of rare queries and deal with the consequences for a hybrid querying technique.

The work of [7] is extended in [11]. In contrast to the work presented in this paper, [11] does not propose a concrete hybrid network topology as it does not discuss the strategies how and when nodes connect to the DHT, whereas we propose a scalable architecture and extensions to the existing Gnutella network interactions in order to facilitate DHT node discovery.

[11] also proposes that the rare files of Gnutella leafs are only identified and published by their ultrapeers. This is not feasible for the following reasons:

Leafs are connected to 1–5 ultrapeers at the same time which means that rare files are published multiple times as the paper does not address this redundancy. Additionally, ultrapeer-leaf connections can be very transient. Thus the leaf can drop connections and select better ones as well as ultrapeers can drop connections to their leafs. Both will result in considerable redundant publishing which is not addressed. Ultrapeers have very limited knowledge of a leaf's files as only a bloom filter is used in [11]. This does not provide the required information. If the proposed solution as used in the experimental deployment of [11] is to use the *Browse Host* Gnutella protocol message to list all the leaf's files, then this solution is obviously not scalable. Only leafs can effectively monitor the hit rate on each of their files, i.e., how often a file is searched and how often it is published. Thus, leafs should also be part of the identification and publication process of rare data item. Our paper proposes a client-based publishing mechanism to address this.

Also [11] does not address the very probable situation of a surge in a file popularity while we describe and propose a solution to that problem. The proposed analytical model does not take into account dynamic querying, which already limits Gnutella's bandwidth consumption considerably. And in contrast to our work, [11] does not address the additional problem of spam and how it particularly affects rare queries.

The implementation presented in [11] is based on PIERSearch, which offers keyword-based searching through inverted lists on files and distributed joins in the DHT. Although the paper acknowledges the problem of hotspots in DHTs, it does not explain how the inverted index will work with popular keywords. As we, the approach in [11] selects only rare files for publishing. However, file names for rare files can and often will contain very popular keywords. Therefore, a system based on keyword search seems to be of limited feasibility in real-world scenarios.

Our paper fully integrates the use of the structured network into the current Gnutella standard by taking advantage of the dynamic querying technique, extending the existing Ping/Pong scheme to ensure that nodes are able to bootstrap to the DHT at any given time.

Other related work includes the approach by Castro et al. [12] which proposes a hybrid system in which the network maintenance is handled by a structured network and the search and data replication is done in an unstructured network. This study is based on the obsolete original Gnutella network and therefore is not applicable to the current Gnutella system anymore.

Several other approaches [13, 14, 3, 15] have tried to address the scalability problems of the original Gnutella protocol by modifying the network topology, the query algorithms or the data replication strategies in the network. These approaches have proposed techniques that exploit node heterogeneity and introduce some flow control for queries, techniques which are now already included in the current Gnutella standard which we base our approach on.

8 Conclusions

In this paper we have presented an extension of Gnutella with a DHT to address the problem of queries for rare files, which are approximately 20% of the total queries in

Gnutella but account for significant network traffic. We presented experimental results from a large-scale experimental study that show that Gnutella handles such queries very inefficiently und unsuccessfully and that such queries cause excessive bandwidth consumption. Our hybrid approach uses Gnutella for popular files which it can handle efficiently and a Kademlia DHT of ultrapeers for rare files. We presented the algorithms to set up the hybrid infrastructure, to detect and manage rare data items, and to query for such data, and demonstrated the efficiency and validity of our approach by a large-scale experimental deployment in the live Gnutella (4 million users) and Azureus Kademlia (800 thousand users) networks. Our results show that Gnutella can benefit considerably from our hybrid approach as it increases success rates from 27% to 99% and decreases bandwidth consumption by an order of magnitude. The experiments were done with a production-quality implementation which will be included into the LimeWire P2P software.

References

1. Stutzbach, D., Rejaie, R.: Characterizing Unstructured Overlay Topologies in Modern P2P File-Sharing Systems. In: Internet Measurement Conference. (2005)
2. Jovanovic, M., Annexstein, F., Berman, K.: Modeling peer-to-peer network topologies through small-world models and power laws. TELFOR (2001)
3. Lv, Q., Cao, P., Cohen, E., Li, K., Shenker, S.: Search and replication in unstructured peer-to-peer networks. In: International Conference on Supercomputing. (2002)
4. Saroiu, S., Gummadi, P.K., Gribble, S.D.: A Measurement Study of Peer-to-Peer File Sharing Systems. In: Proceedings of Multimedia Computing and Networking. (2002)
5. Maymounkov, P., Mazieres, D.: Kademlia: A Peer-to-Peer Information System Based on the XOR Metric. In: First International Workshop on Peer-to-Peer Systems (IPTPS'01), London, UK (2002)
6. Stutzbach, D., Rejaie, R., Zhao, S.: Characterizing Files in the Modern Gnutella Network: A Measurement study. In: SPIE/ACM Multimedia Computing and Networking, San Jose, CA (2006)
7. Loo, B.T., Huebsch, R., Stoica, I., Hellerstein, J.M.: The Case for a Hybrid P2P Search Infrastructure. In: 3rd International Workshop on Peer-to-Peer Systems (IPTPS'03). (2004)
8. Chu, J., Labonte, K., Levine, B.N.: Availability and locality measurements of peer-to-peer file systems. In: ITCom: Scalability and Traffic Control in IP Networks. (2002)
9. Klemm, A., Lindemann, C., Vernon, M., Waldhorst, O.: Characterizing the query behavior in peer-to-peer file sharing work-loads. In: Internet Measurement Conference, Taormina, Italy (2004)
10. Chun, B., Culler, D., Roscoe, T., Bavier, A., Peterson, L., Wawrzoniak, M., Bowman, M.: PlanetLab: An Overlay Testbed for Broad-Coverage Services. ACM SIGCOMM Computer Communication Review **33**(3) (2003)
11. Loo, B.T., Hellerstein, J.M., Huebsch, R., Shenker, S., Stoica, I.: Enhancing P2P File-Sharing with an Internet-Scale Query Processor. In: 30th International Conference on Very Large Databases (VLDB). (2004)
12. Castro, M., Costa, M., Rowstron, A.: Peer-to-peer overlays: structured, unstructured, or both. Technical report, Microsoft Research, Cambridge, UK (2004)
13. Chawathe, Y., Ratnasamy, S., Breslau, L., Lanham, N., Shenker, S.: Making Gnutella-like P2P Systems Scalable. In: ACM SIGCOMM. (2003)
14. Krishnamurthy, B., Wang, J., Xie, Y.: Early Measurements of a Cluster-based Architecture for P2P Systems. In: ACM SIGCOMM. (2001)
15. Osokine, S.: The Flow Control Algorithm for the Distributed 'Broadcast-Route' Networks with Reliable Transport Links (2001) http://www.grouter.net/gnutella/flowcntl.htm.