Structured and unstructured overlays under the microscope

A measurement-based view of two P2P systems that people use

Yi Qiao and Fabián E. Bustamante
Department of Electrical Engineering & Computer Science, Northwestern University
{yqiao,fabianb}@cs.northwestern.edu

Abstract

Existing peer-to-peer systems rely on overlay network protocols for object storage and retrieval and message routing. These overlay protocols can be broadly classified as structured and unstructured – structured overlays impose constraints on the network topology for efficient object discovery, while unstructured overlays organize nodes in a random graph topology that is arguably more resilient to peer population transiency. There is an ongoing discussion on the pros and cons of both approaches. This paper contributes to the discussion a multiple-site, measurement-based study of two operational and widely-deployed file-sharing systems. The two protocols are evaluated in terms of resilience, message overhead, and query performance. We validate our findings and further extend our conclusions through detailed analysis and simulation experiments.

1 Introduction

Peer-to-peer Internet applications for data sharing have gained in popularity over the last few years to become one of today's main sources of Internet traffic [31, 12]. Their peer-to-peer approach has been proposed as the underlying model for a wide variety of applications, from storage systems and cooperative content distribution to Web caching and communication infrastructures. Existing peer-to-peer systems rely on overlay network protocols for object storage/retrieval and message routing. These overlay protocols can be classified broadly as either structured or unstructured based on the constraints imposed on how peers are organized and where stored objects are kept. The research community continues to debate the pros and cons of these alternative approaches [5]. *This paper contributes to this discussion the first multi-site, measurement based study of two operational and widely deployed P2P file-sharing systems.*

Most P2P systems in use today [8, 13] adopt fully distributed and largely unstructured overlays. In such *unstructured* systems there are few constraints on the overlay construction and data placement: peers set up overlay connections to a (mostly) arbitrary set of other peers they know, and shared objects can be placed at any node in the system. While the resulting random overlay structures and data distributions may provide high resilience to the degrees of transiency (i.e., churn) found in peer populations, they limit clients to nearly "blind" searches, using either flooding or random walks to cover a large number of peers.

Structured, or DHT (Distributed Hash Table)-based protocols [28, 33, 36, 25], on the other hand, reduce the cost of searches by constraining both the overlay structure and the placement of data – data objects and nodes are assigned unique identifiers or keys, and queries are routed based on the searched object keys to the node responsible for keeping the object (or a pointer to it). Although the resulting overlay provides efficient support for exact-match queries (normally in $O(log(n))$), this may come at a hefty price in terms of churn resilience, and the systems' ability to exploit node heterogeneity and efficiently support complex queries.

This paper reports on a detailed, measurement-based study of two operational file-sharing systems – the unstructured Gnutella [8] network, and the structured Overnet [23] network. In a closely related effort, Castro et al. [5] presents a simulation-based, detailed comparison of both approaches using traces of Gnutella nodes arrival and departures [30]. Our study complements their work, focusing on the *characterization* – not comparison – of two operational instances of these approaches in terms of resilience, query and control message overhead, query performance, and load balancing.

Some highlights of our measurement results include:

- Both systems are efficient in terms of control traffic (bandwidth) overhead under churn. In particular, Overnet peers have surprisingly small demands on bandwidth.
- While both systems offer good performance for

exact-match queries of popular objects, Overnet surprisingly yields almost twice the success rate of Gnutella (97.4%/53.2%) when querying for a set of shared objects extracted from a Gnutella client.

- Both systems support fast keyword searches. Flooding in Gnutella guarantees fast query replies, especially for highly popular keywords, while Overnet successfully handles keyword searches by leveraging its DHT structure.

- Overnet does an excellent job at balancing search load; even peers responsible for the most popular keywords consume only 1.5x more bandwidth than that of the average peer.

We validate our findings and further extend our conclusions (Sections 7 and 8) through additional measurements as well as detailed analysis and simulation experiments. The measurement and characterization of the two large, operational P2P systems presented in this paper will shed light on the advantages/disadvantages of each overlay approach and provide useful insights for the design and implementation of new overlay systems.

After providing some background on unstructured and structured P2P networks in general and on the Gnutella and Overnet systems in particular, we describe our measurement goals and methodology in Section 3. Sections 4-6 present and analyze our measurement results from both systems. Section 9 discusses related work. We conclude in Section 10.

2 Background

This section gives a brief overview of general unstructured and structured P2P networks and the deployed systems measured in our study – Gnutella and Overnet.

2.1 The Gnutella Protocol

In unstructured peer-to-peer systems, the overlay graph is highly randomized and difficult to characterize. There are no specific requirements for the placement of data objects (or pointers to them), which are spread across arbitrary peers in the network. Given this random placement of objects in the network, such systems use flooding or random walk to ensure a query covers a sufficiently large number of peers. Gnutella [8] is one of the most popular unstructured P2P file-sharing systems. Its overlay maintenance messages include *ping*, *pong* and *bye*, where *pings* are used to discover hosts on the network, *pongs* are replies to pings and contain information about the responding peer and other peers it knows about, and *byes* are optional messages that inform of the upcoming closing of a connection. For query/search, early versions of Gnutella employ a simple *flooding* strategy, where a query is propagated to all neighbors within a certain number of hops. This maximum number of hops, or *time-to-live*, is intended to limit query-related traffic.

Two generations of the Gnutella protocols have been made public: the "flat" Gnutella V0.4 [7], and the newer loosely-structured Gnutella V0.6 [14]. Gnutella V0.6 attempts to improve query efficiency and reduce control traffic overhead through a two-level hierarchy that distinguishes between superpeers/ultrapeers and leaf-peers. In this version, the core of the network consists of high-capacity superpeers that connect to other superpeers and leaf-pears; the second layer is made of low-capacity (leaf-) peers that perform few, if any, overlay maintenance and query-related tasks.

2.2 The Overnet/Kademlia Protocol

Structured P2P systems, in contrast, introduce much tighter control on overlay structuring, message routing, and object placement. Each peer is assigned a unique hash ID and typically maintains a routing table containing $O(log(n))$ entries, where n is the total number of peers in the system. Certain requirements (or invariants) must be maintained for each routing table entry at each peer; for example, the location of a data object (or its pointer) is a function of an object's hash value and a peer's ID. Such structure enables DHT-based systems to locate an object within a logarithmic number of steps, using $O(log(n))$ query messages. Overnet [23] is one of the few widely-deployed DHT-based file-sharing systems. Because it is a closed-source protocol, details about Overnet's implementation are scarce, and few third-party Overnet clients exist. Nevertheless, some of these clients, such as MLDonkey [21], and libraries like KadC [11] provide opportunities for learning about the Overnet protocol.

Overnet relies on Kademlia [20] as its underlying DHT protocol. Similar to other DHTs, Kademlia assigns a 160-bit hash ID to each participating peer, and computes an equal-length hash key for each data object based on the SHA-1 hash of the content. *<key,value>* pairs are placed on peers with IDs close to the key, where "closeness" is determined by the XOR of two hash keys; i.e., given two hash identifiers, x, and y, their distance is defined by the bitwise exclusive or (XOR) ($d(x, y) = x \oplus y$). In addition, each peer builds a routing table that consists of up to $log_2(n)$ buckets, with the ith bucket B_i containing IDs of peers that share a i-bit long prefix. In a 4-bit ID space, for instance, peer 0011 stores pointers to peers whose IDs begin with 1, 01, 000, and 0010 for its buckets B_0, B_1, B_2 and B_3, respectively (Fig. 1). Compared to other DHT routing tables, the placement of peer entries in Kademlia buckets is quite flexible. For example, the bucket B_0 for peer 0011 can contain any peers having an ID starting with 1.

Kademlia supports efficient peer lookup for the k clos-

Bucket ID	Common Prefix Length	Cached Peers (Bucket Entries)
B_0	0	1001, 1100, 1101
B_1	1	0110, 0100
B_2	2	0001
B_3	3	0010

Figure 1: Routing table of peer 0011 in a 4-digit hash space.

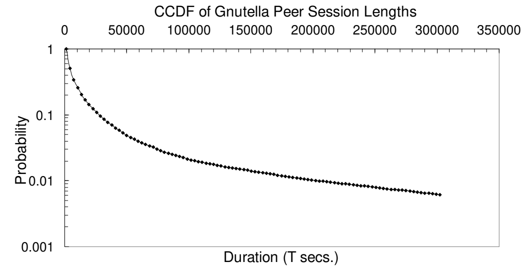

Figure 2: Peers' session length in Gnutella.

est peers for a given hash key. The procedure is performed in an iterative manner, where the peer initiating a lookup chooses the α closest nodes to the target hash key from the appropriate buckets and sends them $FIND\_NODE$ RPCs. Queried peers reply with peer IDs that are closer to the target key. This process is thus repeated, with the initiator sending $FIND\_NODE$ RPCs to nodes it has learned about from previous RPCs until it finds the k closest peers. The XOR metric and the routing bucket's implementation guarantee a consistent, $O(log(n))$ upper bound for the hash key lookup procedure in Kademlia. [1]

Overnet builds a file-sharing P2P network with an overlay organization and message routing protocol based on Kademlia. Overnet assigns each peer and object a 128-bit ID based on a MD4 hash. Object search largely follows the $FIND\_NODE$ procedure described in the previous paragraph with some modifications. We will introduce additional details on Overnet's search mechanism as we present and analyze its query performance in Section 5.

3 Measurement Goals and Methodology

Our study focuses on the *characterization* of two operational instances of the unstructured (Gnutella) and unstructured (Overnet) approaches to P2P networks in terms of churn resilience (Section 4.1), query and control message (Sections 6 and 4.2) overhead, query performance (Section 5.1 and 5.2)), and load balancing (Section 6.1). A fair head-to-head comparison of the two deployed systems would be impossible as one cannot control key parameters of the systems such as the number of active peers, the content and the query workload.

We employed a combination of passive and active techniques to carry out our measurement study. For Gnutella, we modified Mutella-0.4.3 [22], an open-source Gnutella client. Mutella is a command-based client, which conforms to Gnutella specifications V0.4 and V0.6. Our measurements of Overnet are based on a modified MLDonkey [21] client, an open-source P2P client written in Objective Caml, that supports multiple P2P systems including Overnet. Modifications to both clients included extra code for parameter adjustment, probing and accounting, among others. None of these modifications affected the outcome of the collected metrics.

Our modified Gnutella and Overnet clients, each performing a similar batch of experiments, were instantiated at four different locations [2] around the world and run concurrently to identify potential geographical biases and factor out time-of-day effects from our measurements. All experiments were conducted from April 1st to the 30th, 2005. For brevity, unless otherwise stated, the data presented in the following sections as well as the associated discussions are based on clients placed behind a DSL connection in Evanston, Illinois. Measurements and analysis from the remaining three sites yield similar results and will be briefly discussed in Section 7.

4 Churn and Control Traffic Overhead

The transiency of peer populations (*churn*), and its implications on P2P systems have recently attracted the attention of the research community. A good indication of churn is a peer's session length – the time between when the peer joins a network until it subsequently leaves. Note that a single peer could have multiple sessions during its lifetime by repeatedly joining and leaving the network. We performed measurements of session length for peers in both the Gnutella and Overnet networks, and studied the level of churn of these two systems.

In the context of file-sharing P2P systems, the level of replication, the effectiveness of caches, and the spread and satisfaction rate of queries will all be affected by how dynamic the peers' population is [1, 3, 6, 15, 27]. For P2P networks in general, control traffic overhead is also a function of the level of churn. Control traffic refers to protocol-specific messages sent between peers for overlay maintenance, including peers joining/leaving the overlay, updating routing tables or neighbor sets, and so on. It does not include any user-generated traffic such as query request and replies. In this section, we study the control traffic overhead for both networks and discuss our findings.

4.1 Level of Churn

We performed session-length measurements of Gnutella by modifying the Mutella client [22]. We first collected a large number of *(IP, port)* tuples for Gnutella peers by examining ping/pong messages that our Mutella client received. From the set of all the *(IP, port)* tuples, we probe a randomly selected subset to collect data representative of peers' session-length distribution for the Gnutella network. While performing the measurement, our client periodically (every 20 minutes) tries to initiate a Gnutella-specific connection handshake with each peer in the list. The receiving peer at the probed *IP* and *port*, if active, either accepts or refuses the connection request (indicating its "BUSY" status). Our session length measurement for Gnutella lasted for 7 days, and captured approximately 600,000 individual peer sessions.

Session-length probing for Overnet was conducted using our modified Overnet (MLDonkey) client [21]. Each peer in Overnet is assigned a unique 128-bit hash ID that remains invariant across sessions. To search for a particular user by hash ID, Overnet provides the *Overnet-Search* message type. Peers connect using a *Overnet-Connect* message; when a peer receives an *OverentConnect* message, it responds with an *OvernetConnectReply*, which contains 20 other peers' IDs known by the replying peer. To begin the session-length measurement, we collected the hash IDs of 40,000 Overnet peers by sending *OvernetConnect* messages and examining IDs contained in the corresponding reply messages. We then periodically probed these peers to determine whether they were still online. Since it is possible for a peer to use different *(IP, port)* pairs for different sessions, we rely on *OvernetSearch* messages to iteratively search for and detect peers that start new sessions using different *(IP, port)* tuples. As in the case of Gnutella, the session length measurement for Overnet also lasted 7 days. During this time we continuously probed 40,000 distinct peers and measured over 200,000 individual sessions.

Figures 2 and 3 give the Complementary Cumulative Distribution Function (CCDF) of peers' session lengths for Gnutella and Overnet, respectively. Figure 2 shows that peers in the Gnutella network show a medium degree of churn: 50% of all peers have a session length smaller than 4,300 seconds, and 80% have session lengths smaller than 13,400 seconds ($<$ 4 hours). Only 2.5% of the session lengths are longer than one day. The median session length of an Overnet peer (Fig. 3) is around 8,100 seconds, 80% of the peers have sessions that last less than 29,700 seconds, and 2.7% of all session lengths last more than a day. Overall, the Overnet network has a measurably lower, but still similar, level of churn when compared to the Gnutella network. In the next section, we analyze the impact of these levels of churn on control traffic overhead for each of these systems.

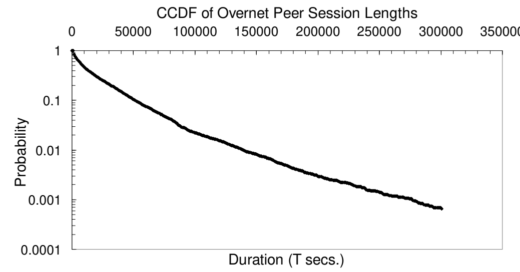

Figure 3: Peers' session length in Overnet.

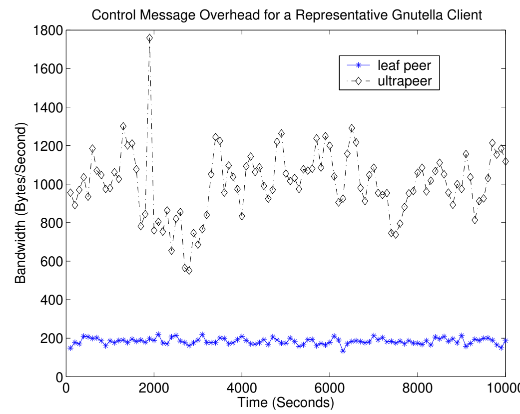

Figure 4: Bandwidth consumption of control messages for the Gnutella client.

4.2 Control Traffic Overhead

In this section, we present and discuss a three-day measurement of the control traffic for Gnutella and Overnet. To better illustrate changes on bandwidth demands at finer time resolution, each figure in this section shows a representative measurement window of 10,000 seconds taken from a client behind a DSL connection in Evanston, Illinois. Each data point corresponds to the average bandwidth demands for every 100 seconds. Data on bandwidth demand for other measurement periods and measurement sites produced similar results.

The measurement of the Gnutella network was done for Gnutella V0.6 using our modified client, which can act either as a leaf peer or as an ultrapeer. The modified client records all control-related messages that it generates in addition to those messages originating from other peers and routed through it. The majority of control-related messages in Gnutella are pings and pongs, along with small percentages of other types of control messages such as those used to repair routing tables. We opted for Gnutella V0.6 as this is the most common version in the Gnutella network.

Gnutella uses ultrapeers to exploit peers' heterogeneity in attributes such as bandwidth capacity and CPU speed, thereby making the system more scalable and efficient. Ultrapeers typically connect to a larger number of neighbors than do leaf peers and they are assigned more responsibility for responding to other peers' messages,

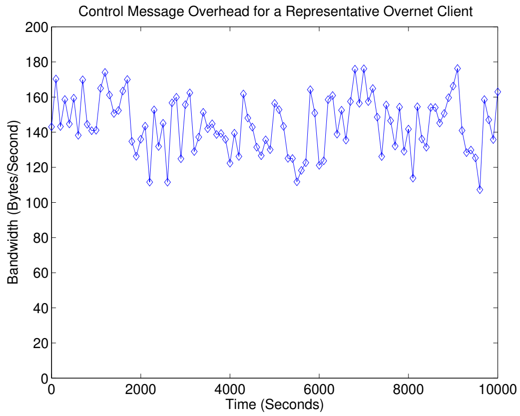

Figure 5: Bandwidth consumption of control messages for the Overnet client.

thus consuming several times more bandwidth for control messages than leaf peers. Figure 4 illustrates this for a leaf peer connected to no more than 4 ultrapeers, and an ultrapeer connected to a maximum of 5 ultrapeer neighbors and 8 leafpeer children. As would be expected, while a leaf peer typically only consumes around 200 Bytes/second for control messages, an ultrapeer normally needs to contribute 5 to 6 times more bandwidth for control traffic (between 800 and 1,400 Bytes/second). Despite this high relative difference between peer types, the bandwidth consumption for an ultrapeer is still reasonably low and never exceeds 2,000 Bytes/second in our measurement.

Overall, a Gnutella peer does not consume a large amount of bandwidth for control-related messages, as it would be expected given the loose organization of its overlay infrastructure. Peers joining and leaving the Gnutella network have little impact on other peers or on the placement of shared data objects, and thus do not result in significant control traffic.

Figure 5 shows the control traffic overhead for our modified Overnet client. Contrary to common belief, we found that Overnet clients consume surprisingly little bandwidth: only around 100 to 180 Bytes/second. The control message overhead for a peer is determined by a number of factors, such as the peer's number of neighbors, the peer's (and its neighbors') probing intervals, the size of the control message, etc. It is thus difficult to directly compare control overhead across different protocols. Nevertheless, the reader should consider that while the measured Gnutella client has a significantly shorter probing interval (10 seconds), it also limits the number of neighbors to 13 (in the ultrapeer case). The Overnet client, on the other hand, often has over 300 neighbor peers in its buckets, which are probed at 1,800-sec intervals.

Although a structured (DHT-based) system has strict rules for neighbor selection and routing table maintenance, these invariants can be maintained in a variety of ways, resulting in quite different levels of control message overhead for different DHT-based systems. Recall that an Overnet peer p only needs to maintain certain numbers of buckets to peer with others and perform routing. Any peer whose hash ID shares the first bit with that of peer p can be in the bucket B_1 of p. Moreover, an Overnet peer never immediately repairs bucket entries corresponding to peers that have left the system. Instead, it fills the missing entries either by periodically asking some of its active neighbors for fresh peers to fill the entries or by performing lazy updates when the peer receives control- or query-related messages for a peer whose key matches an empty bucket entry. Interestingly, this flexibility built into the Overnet's bucket entry system shares much in common with the routing table entry flexibility in Pastry [5], while the periodical, lazy bucket entry repair is quite similar to the periodical, passive routing table (or leaf set) maintenance and repair mechanism that MSPastry [4] and Bamboo [27] employ. Their simulation results and our measurement data on the large, operational Overnet demonstrate the low control message overhead for a DHT-based system with these two properties.

5 Query Performance

Querying is a key component of any P2P file-sharing network, and the effectiveness and the scalability of queries largely determine the user's experience. Queries can be broadly divided into two categories: keyword searching and exact-match queries. Keyword searching is an attempt to find one or more objects containing a specific keyword, while an exact-match query aims at specific objects. Exact-match queries can be further classified as searches for popular (and thus highly replicated) objects and for rare objects (or *needles*) which appear on very few peers. In this section, we evaluate query performance for all three cases in both networks.

Query performance is evaluated in terms of the following three metrics: *Query Success Rate*, the ratio of successful queries to all queries, *Query Resolution Time*, the time between query submission and the arrival of the first query reply, and *Z Query Satisfaction Time* [35], the time it takes for a query to get back at least Z query hits. The last metric is mainly used for keyword search, since it is desirable for such a search to return multiple objects with the requested keyword.

5.1 Keyword Searching

In a P2P file-sharing network, a user often wants to perform a broad search for one or more keywords (such as *Piazzolla*, *Adios Nonino* or *US Open*) to get a list of possible interesting objects before deciding what to download. Intuitively, unstructured P2P systems appear to be

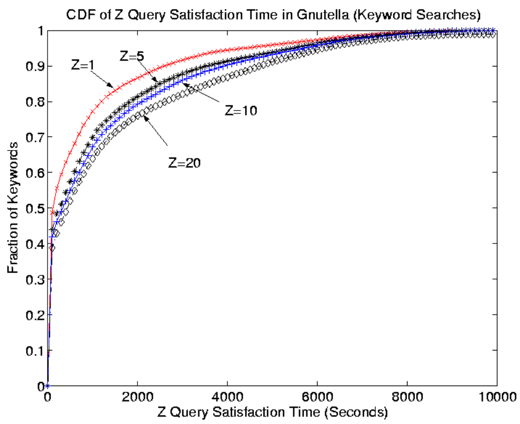

Figure 6: Keyword search in Gnutella.

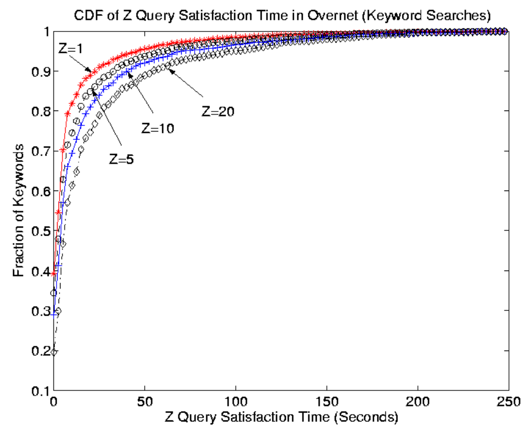

Figure 7: Keyword search in Overnet.

better suited for keyword searching than do structured ones. For example, a simple implementation of keyword searching uses flooding to propagate the query, and requires that each peer check its local file index for possible matches and reply to the request originator if a match is found. Structured systems, on the other hand, usually have single one-to-one mapping between each individual object and one particular peer that is based on the closeness of the object and the peer hash.

To evaluate the efficiency of each system in supporting keyword searching, we performed experiments using our modified peer clients. For each system, our client sequentially issued queries for 10,000 different keywords. For each search we record the time we start the search and the time when a query hit matching the keyword is received. Since a keyword search can often return multiple query hits indicating different objects containing that keyword, we record all hits for that keyword.

Note that the keywords used were the 10,000 most popular keywords extracted from the query strings routed through our Gnutella client. Thus, our measurement methodology may give some performance advantages to Gnutella, since it is possible that some keywords searched by Gnutella clients never appear in the Overnet system.

Figure 6 shows the CDF of Z query satisfaction time of searches for the 10,000 keywords in Gnutella. Each curve depicts a CDF of query satisfaction times for a particular query satisfaction level (Z). As can be seen, 50% of all keyword searches can be resolved in 100 seconds when $Z = 1$, but it takes over 400 seconds for the same percentage of searches to be satisfactory when $Z = 10$.

Although an unstructured system can easily support keyword searches, it may not be able to do this in the most efficient manner. Recall that for flooding to be scalable, a TTL value must limit the scope of a search. While this may not cause any problems for popular keywords, for less popular ones a single controlled flooding may not be able to find enough peers that collectively contain Z different matching objects.

Figure 7 gives the CDF of Z query satisfaction time for keyword searches in Overnet. Surprisingly, the DHT-based Overnet does an excellent job: 50% of all searches can be resolved in 2.5 seconds with at least a single query hit. Even when $Z = 20$, half of all queries can be satisfied in around 7 seconds.

A careful inspection of the MLDonkey source code and relevant documentation [20, 11] revealed the reason behind these results. To publish keywords for a shared object in Overnet, a client first parses the object name into a series of keyword strings. For each keyword string, the client sends the object's metadata block (containing the object's name, file type and size, MD4 hash of the content, etc) to a certain number of peers with hash IDs close to the keyword's MD4 hash. Note that this technique does not require a "perfect" match between the keyword hash and the peer ID. In Section 6, we will show that this type of inexact match is important for load balancing and search efficiency in Overnet.

When a user performs a search for one of the keywords, the query will be directed to peers whose hash IDs are close enough to it. As a result of the keyword publishing process just described, these peers are very likely to store metadata blocks for objects that match the keyword. For a search that contains multiple keywords the peer can simply filter the results based on the additional keywords. Upon receiving the query, a peer with matching keywords returns a query hit to the initiator and also sends the metadata for the objects matching the keyword(s).

5.2 Exact-match Queries

The performance of exact-match queries is another important factor in determining the user experience. With the results from a keyword search, a user typically chooses one or more specific objects to download; at this point, exact-match queries are performed to locate and download these objects.

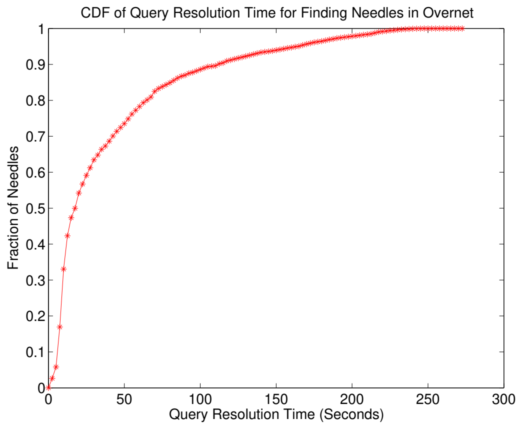

Figure 8: Finding "needles" in Overnet.

Previous work has shown that the object download process often takes from several hours to days or even weeks to complete [29, 9]. During this period, some peers providing a specific shared object may be too busy to handle clients' download requests, or may become disconnected after some time. To improve download times in this environment, most recent P2P systems support parallel downloading, where a downloading peer can fetch multiple, different pieces of the same object from various peers. Thus, the ability of a peer to continuously search and find alternative sources for a particular object is extremely important for fast and reliable downloads.

5.2.1 Finding Rare Objects – "Needles"

We now look at the performance of Overnet and Gnutella when searching for rare objects. We do this using a worst-case approach, evaluating system performance when each object being searched for has exactly one replica in the entire system.

For each experiment, we ran our modified Gnutella or Overnet clients at the four different measurement locations (Section 3). We alternatively name the four clients A, B, C and D. Peer A shares 1,000 different, randomly generated binary files, each of which has a 50-byte long, random string-based file name; these files are stored only on peer A. Peers B, C and D each issue queries searching for these 1,000 objects. Upon receiving a query hit from peer A, we mark the search as successful and record the time it took to resolve the query. Overall, we are interested in not only the success rate of such queries, but also the resolution speed of successful queries. The process is repeated 10 times before choosing a new node A for the next "round". In total we ran four rounds of experiments in both Gnutella and Overnet.

Figure 8 shows the query performance for finding "needles" in Overnet. Clearly, our Overnet client does an impressive job at finding "needles" – over half of the needles can be found in 20 seconds, while 80% of them can be located within 65 seconds.

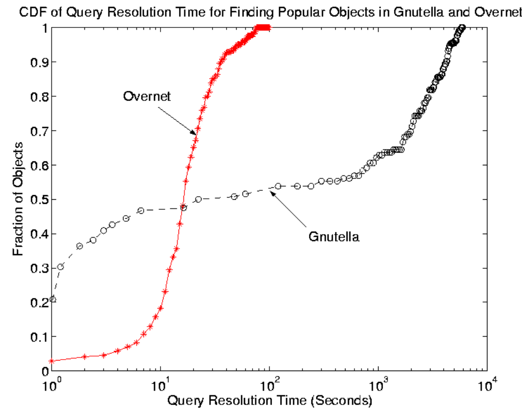

Figure 9: CDF of query resolution time for 20,000 distinct objects from a Gnutella and a Overnet client (x-axis is in log-scale).

Peers whose hash IDs are the closest to that of a shared object will be responsible for storing the object pointer (i.e. the location of the peer sharing the object). Exact-match queries in Overnet first perform the node lookup procedure (Section 2) using the object hash key as the target, then obtain object pointers from these peers to finish the query. Due to the DHT structure, this procedure can be efficiently finished in $O(log(n))$ steps. One minor difference with other DHT-based systems is that pointers for a specific object not only exist at the peers whose IDs are the closest, but also at peers whose IDs are "close enough" to the object hash. As with the case of keyword publishing and searching (Subsection 5.1), this results in a faster search as well as better load balancing.

The performance for finding "needles" from our Gnutella client is significantly worse than expected: an average success rate of about 1.5% across all four rounds of the experiments. Recall that each search in Gnutella is a "controlled" flooding process; thus, it is possible for such a query never to reach peer A, host to the sole replica of the searched object. The lack of mapping between objects and their locations in unstructured systems makes the task of finding needles particularly difficult, even when using flooding as the query mechanism.

5.2.2 Finding Popular Objects

To study the performance of exact-match queries for popular objects, we examined query hit messages from one of our Gnutella clients and extracted 20,000 distinct shared objects. We then performed queries for all these objects on both the Gnutella and Overnet networks using our modified clients and measured their query performance.

Figure 9 shows the CDF of the query satisfaction time for finding these objects in both systems. We use a log scale for the x axis to accommodate the wide time scale over which queries are resolved. Note that each curve

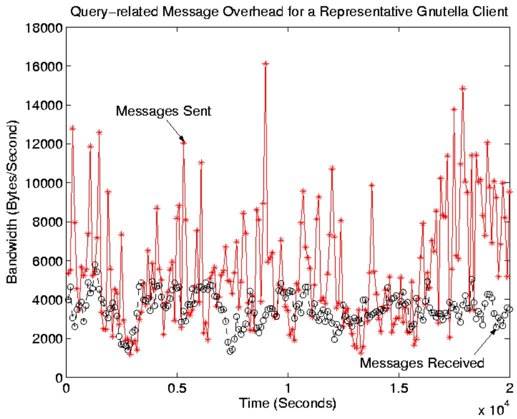

Figure 10: Query-related message overhead for a Gnutella client.

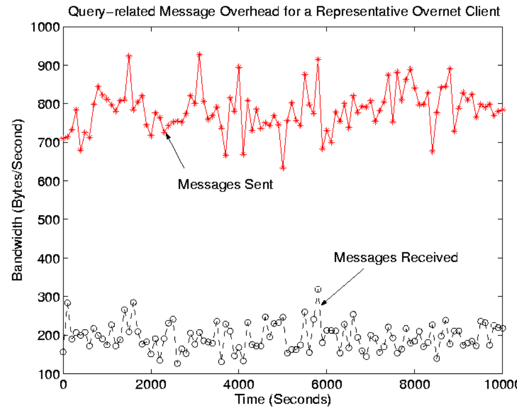

Figure 11: Query-related message overhead for a Overnet client.

in the figure is normalized by the number of successful queries; failed queries are not considered. Both systems can usually return results for such queries very quickly: 50% of all successful queries are finished within 47 seconds in Gnutella and in less than 17 seconds in Overnet. This is not surprising since Gnutella quickly floods a large number of peers within a few hops, while Overnet takes advantage of its $O(log(n))$ search mechanism for each object. A particularly interesting difference, however, is the query success rate in each system – while the Overnet client successfully resolves 97.4% of all these queries, the Gnutella client only yields a success ratio of 53.2%, where most failed searches were for objects that are relatively less popular.

6 Query-related Traffic Load

Query-related traffic includes query messages and query replies and can be generated either by the peer itself due to its own queries, or by some other peers for which the peer needs to forward, route, or answer query-related messages. Thus, the query traffic load serves as a good indication of query efficiency and system scalability.

Note that the numbers we present here include only the query or query-reply messages our client received or sent on behalf of other peers. In other words, our Gnutella or Overnet client did *not* issue any queries during the measurement, which removes the potential bias introduced by our own queries. We performed this measurement for both Gnutella and Overnet at all our four measurement sites, with each measurement lasting for about three days. Each figure presented below shows only a representative measurement window of 10,000 seconds taken from our measurement client behind a DSL line in Evanston, Illinois. Each data point corresponds to the average bandwidth for every 100 seconds. Bandwidth data from other measurement periods and other sites are similar.

Figure 10 gives the bandwidth consumption of query-related traffic for the Gnutella client. Note that we only give numbers for the case of ultrapeer, since leaf peers in Gnutella do not receive or relay any query messages on behalf of other peers. Similar to Figure 4, our Gnutella peer has up to 5 ultrapeer neighbors and 8 leaf-peer children. As the figure shows, our client typically consumes 5 to 10 KBytes/second for sending query and query-hit messages, and 2 to 6 KBytes/second for receiving such messages. Clearly, even under the default settings of our Gnutella implementation (Mutella) of an ultrapeer, query-related traffic could overwhelm a dial-up a modem user and can only be safely handled by broadband peers. This result is not surprising given the flooding query mechanism at the ultrapeer layer of the network. Overall, Gnutella does a reasonable, but not excellent job for query efficiency and query traffic load. We will see later in Section 8 that the introduction of ultrapeers in unstructured systems greatly improved query success rate and system scalability compared with the purely flat, old Gnutella V0.4, making the inherently unscalable flooding query mechanism acceptable in practice.

Upon receiving a query message in Overnet, a peer will select, from its own buckets, peers whose hash IDs are among the closest to the target keyword or file hash of the search, and send these peers' identities back to the query initiator. Additional query messages could then be sent to those peers during the next iteration of the search until the query is resolved. Figure 11 gives the bandwidth consumption of our Overnet client for sending and receiving all these types of query-related messages. Queries here incur much less overhead than in the case of Gnutella: our Overnet client only consumes around 200 Bytes/second for incoming query messages, and 700 to 900 Bytes/second for sending out replies. Queries in Overnet only involve up to $O(log(n))$ iterative steps, with the number of messages involved in each step being at a constant level and never growing exponentially as in the case of flooding. As a result, our Overnet client only needs about one tenth the bandwidth of our Gnutella ultrapeer for handling query-related messages.

Both the Gnutella and Overnet clients consume more outgoing bandwidth than incoming bandwidth for query traffic. Recall that our measurements here are passive. Both our Gnutella ultrapeer in Figure 10 and the Overnet peer in Figure 11 do not initiate any queries. On the other hand, they need to respond to other peers' query messages with replies. Since query reply messages in both systems are typically larger than query messages, more bandwidth thus needs to be consumed for the outbound traffic in our passive measurement.

6.1 Load Balancing

The query load measurement results presented so far are representative of peers with typical configurations in Gnutella and Overnet. Load balancing is critical to system scalability.

Due to the flooding approach to queries, the main contributor of query load for a Gnutella peer is the large number of queries and replies that this peer has to forward. A peer's query load thus largely depends on its number of neighbors. To verify this, we conducted a measurement on our ultrapeer client, where we initially limit the number of its neighbors to 5, then gradually increase the number of neighbors to 100. We record the average query message bandwidth consumption with different number of neighbors. As expected, the query load of the client grows roughly linearly with increasing number of neighbors. In fact, when the number of neighbors exceeds 50, the outgoing bandwidth consumption alone at our ultrapeer consistently rises above 50 KBytes/second, a volume that cannot be handled by common DSL connections. Therefore, query load at different Gnutella peers can be highly skewed and greatly affected by their outdegrees. Still, a peer can easily adjust its query load by changing its number of neighbors.

Previous research reveals that object popularity in P2P systems is highly skewed, and can be well modeled as a Zipf distribution [32]. Thus, for Overnet peers, the main concern about load balancing is that some peers may have IDs that are very close to those of some highly popular keywords, making them potential hot spots of large number of keyword searches. To study this, we change our Overnet client's hash key to be the same as different keywords with different popularities, so that it would be responsible for storing pointers of objects containing these keywords. Due to the diversity of the hash keys for different keywords, we only change the peer's hash key to that of one popular keyword at a time, and test how the query load changes. Overall, we modified our Overnet client to change its own hash ID every 2 hours to be the same as a particular keyword, and mea-

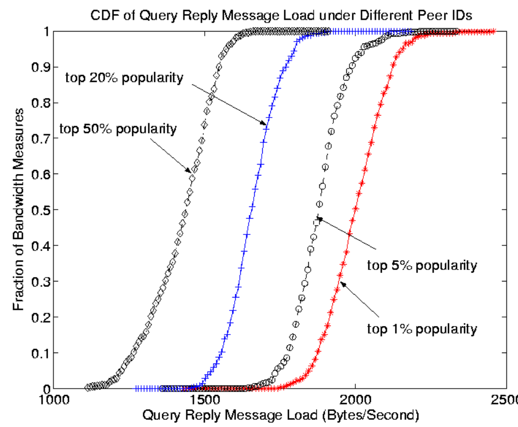

Figure 12: CDF of query-related traffic load when the Overnet client is responsible for different keyword groups with different popularities.

sured its query load during that period. We tested 200 different keywords in four groups, each group containing 50 keywords with similar popularity and different groups representing different popularities. All 50 keywords in Group A belong to the top 1% most popular keywords, while those in Groups B, C, and D belong to the top 5%, 20% and 50% most popular keywords, respectively. These 200 keywords are a small subset of the 10,000 keywords used in Subsection 5.1, and the keyword populartity is based on the local observation of our Gnutella client by examining all query strings that come through.

Figure 12 gives the CDF of query traffic bandwidth consumption for our Overnet client. Each curve corresponds to the CDF of average bandwidth consumption during which our Overnet client has been sequentially assigned the same hash ID as each of the 50 keywords in a particular popularity group. Surprisingly, we did not observe any significant query traffic load for any of these data points: the load for Group A (the top 1% most popular) is generally only 50% higher than that of Group D (50% most popular, or equivalently, median popularity) under different percentages. On the other hand, the highly skewed keyword popularity distribution indicates that the most popular keywords should get at least one order of magnitude more queries than keywords with median popularity.

One possible explanation of the aforementioned disparity between query load and keyword popularity is that Overnet does a good job at distributing query load for a popular keyword among multiple peers whose hash ID is close enough to the hash of the keyword. As briefly mentioned in subsection 5.1, Overnet is not restricted to put file pointers containing a given keyword precisely at the peer whose hash ID is the closest to the keyword's hash. Instead, these pointers can be replicated and distributed among multiple peers whose hash keys are close enough

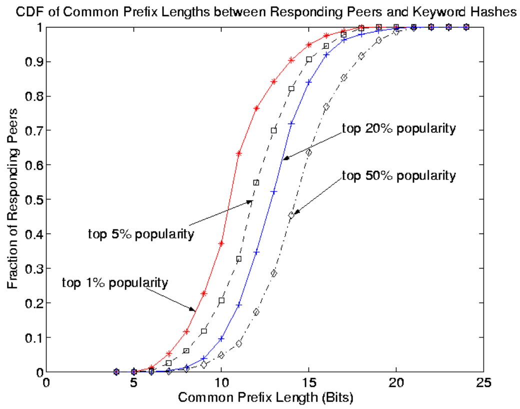

Figure 13: CDF of common prefix lengths between hash keys of responding peers with keyword hashes.

to that of the keyword.

To validate this, we conducted another experiment where we issue searches for keywords in each of the four popularity groups. To each peer that answers a particular keyword search with object pointers, we issue another message to retrieve its hash ID and compare it with the keyword hash to get their common prefix. The longer the common prefix, the closer the two hash keys are.

Figure 13 shows the CDF of the common prefix length between IDs of peers that answered our keyword queries with those of the keywords. Different curves correspond to keyword searches in different popularity groups. Clearly, peers that answer more popular queries tend to share shorter prefixes with the keyword hashes. For example, for keywords in the top-50% popularity range, the common prefix lengths between a keyword and a responding peer is typically between 13 to 15 bits, but often drops to around 10 to 11 bits for the top 1% most popular keywords. This indicates that object pointers for the top 1% most popular keywords are replicated and distributed on about 8 to 16 times more peers than are those of the top 50% most popular keywords. Thus, query load for popular keywords is widely distributed across a larger number of peers in Overnet, effectively achieving load balancing as illustrated in Figure 12.

7 The Impact of Geographical Placement of Peers

Most measurement and discussions presented so far were illustrated using data from our North America measurement site in Evanston, Illinois. To validate that the observed trends and corresponding conclusions are general and do not depend on the particular geographical locations of the measured peers, we repeated our study in all four sites previously mentioned.

For example, Figure 14 depicts the *Z query satisfaction time* of keyword searches using the four measurement sites in three different continents for both Overnet and Gnutella. As the figure shows, at the US measurement site searches for 80% of all keywords can be satisfied in 2,462 and 27.16 seconds, respectively, for Gnutella and Overnet. At the Switzerland site, these two results only change slightly to 2,527 seconds for Gnutella and 23.60 seconds for Overnet. It is clear from the table that search performance across different sites shows only minor difference for either system across the different sites.

Fraction of Keywords	0.50	0.80	0.90	0.95
Gnutella – U.S.	402.63	2462.57	4511.07	5820.36
Gnutella - Switzerland	421.83	2527.22	4301.53	5574.51
Gnutella – France	430.79	2729.32	4599.89	6220.87
Gnutella – China	483.18	2946.60	4741.86	6184.62
Overnet – U.S.	7.03	27.16	54.97	97.49
Overnet – Switzerland	6.38	23.60	49.12	82.21
Overnet - France	6.65	26.85	51.14	100.47
Overnet - China	7.54	28.19	58.18	109.82

Figure 14: Keyword search performance of four clients across different continents for both the Gnutella and Overnet networks. Numbers shown in the table are Z query satisfaction times (in seconds) for different fractions of keywords with $Z = 20$.

In general, all measured data show similarly high degrees of consistency across the different sites. With the same network and same client configurations, for instance, the average control traffic across different measurement sites for both Overnet and Gnutella always remains well below 10%, while the degree of churn of other peers in either Gnutella or Overnet systems as observed by our clients is virtually the same, regardless of the client location.

8 Simulation Study

Having presented measurement data and their analysis for both Gnutella and Overnet, we now use trace-driven simulation of both systems to help further validate and consolidate our findings. Simulations also enable us to obtain aggregated performance numbers of the whole system besides those from individual peers.

Due to space constraints, we only present a small subset of our simulation results. For Gnutella, we explore its message overhead with respect to flooding queries. We also examine the advantage of ultrapeers in terms of system scalability and efficiency. For Overnet, we focus on the lower-than-expected control message overhead and its potential impact on query performance, as well as how effective search and load balancing is attained in Overnet.

All simulations were driven by our collected traces in each system as described in Subsection 4.1. Joins and leaves of peers strictly follow individual sessions captured in the traces. The number of online peers at any time during a simulation ranges between 12,000 and 16,000 for both the Gnutella and Overnet network. Each

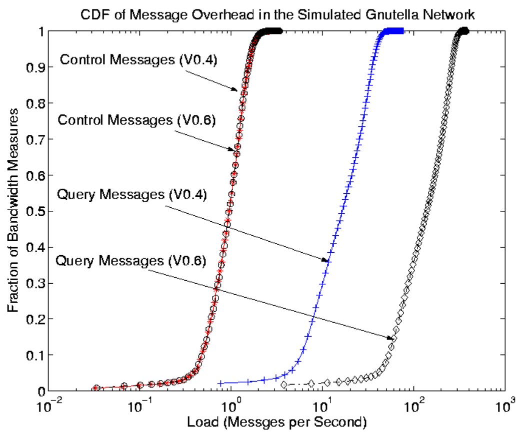

Figure 15: CDF of message overhead for Gnutella V0.4 peers and V0.6 ultrapeers.

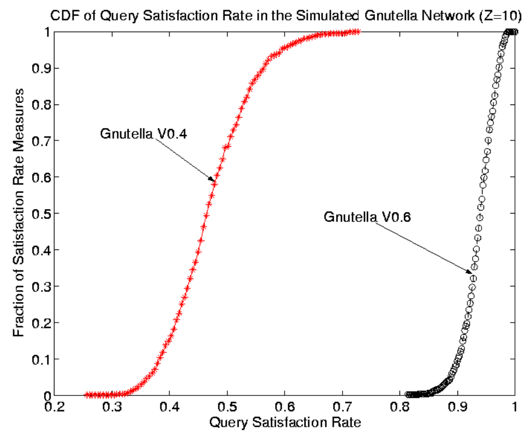

Figure 16: CDF of query success rates for Gnutella V0.4 and V0.6.

simulation runs for one week of simulated time, during which we capture and log various performance numbers including control message overhead, query performance, load balancing condition, etc.

8.1 Gnutella

Our event-based Gnutella simulator was written in 5,000 lines of C++ code. The simulator follows the Gnutella specification, and supports all membership management functionalities as well as query related tasks. We run simulations for both Gnutella V0.4 and V0.6. Each active peer issues a query every 100 seconds on average. Since our network is much smaller than the whole Gnutella network, a smaller value of TTL, 5, is used for flooding. For Gnutella V0.4, each peer is connected with 3 to 10 other peers and has an average outdegree of 5. For Gnutella V0.6, each ultrapeer is connected with 4 leafpeers and 5 other ultrapeers on average.

Figure 15 show the CDF of control message and query message overhead for Gnutella peers in our simulation. Results for both normal peers in Gnutella V0.4 and ultrapeers in Gnutella V0.6 are included. Note that for Gnutella V0.6 we only show numbers for ultrapeers since they dominate the message overhead of leafpeers. As can be seen in the figure, control message overhead for Gnutella is usually very low. Under our simulation settings, it is usually around 1 message/second for both normal V0.4 peers and V0.6 ultrapeers. On the other hand, the flooding query mechanism in Gnutella, although being controlled by the TTL value, results in much higher bandwidth consumption. Normal V0.4 peers typically need to support 5 to 60 messages per second while an ultrapeer could experience 30 to 300 query messages per second. This is not surprising since for Gnutella V0.6, flooding is only restricted to the ultrapeer layer which has a much smaller number of peers than the total population. Thus a flooding query has much higher chance of imposing load on ultrapeers than of reaching normal peers in V0.4. On the other hand, leaf peers in V0.6 have no query-related responsibility. As a result, the aggregated query loads of all peers for V0.4 and V0.6 stay about the same. Clearly, Gnutella V0.6 takes full advantage of peers' heterogeneity by placing peers with lower bandwidth capacity as leaves.

Another potential benefit of ultrapeers in V0.6 is improved query performance. As already mentioned, flooding in V0.6 is only performed at the ultrapeer layer, which consists of a small fraction of all peers in the system. With similar TTL settings, a query message in V0.6 reaches a much larger fraction of all peers in the network when one considers that each ultrapeer indexes all its leaf peers' shared objects. As a result, query performance in V0.6 is much better than in V0.4. Figure 16 shows the CDFs of keyword *query satisfaction rate* for both Gnutella V0.4 and V0.6, with the same TTL setting of 5. Here we set the value of Z to be 10. We sample query satisfaction rates of all queries in the network every 100 seconds to obtain the CDF curve. Gnutella V0.6 shows a clear advantage, yielding 30-50% higher query satisfaction rates for the same fraction of satisfaction rate measures.

The introduction of ultrapeers not only improves the usability of the system for lower-bandwidth capacity users but also boosts query performance. Nevertheless, the scalability problem of flooding queries in Gnutella V0.4 persists in V0.6.

8.2 Overnet

Our Overnet simulator was written in 3,000 lines of C++ code and supports all membership management, routing, as well as query functionalities of Overnet. We use a 64-bit key space for all hash keys. Each peer maintains a set of buckets storing its neighbor peers, each bucket B_i stores up to 5 neighbors whose hashes share a common prefix length of i with the ID of the host peer. Each peer sends out a query every 100 seconds.

Pinging Interval	200 S	400 S	800 S	1600 S	3200 S
Control Message Overhead	0.5351	0.2669	0.1319	0.0657	0.0328
Valid Bucket Entry Percentage	0.9808	0.8936	0.5679	0.4892	0.3986
Average Query Iterations	2.8826	3.0227	3.4725	3.6021	3.7248
Query Success Rate	1.0000	1.0000	1.0000	1.0000	1.0000

Figure 17: Control Message Overhead and Query Performance for Different Probing Intervals.

To update and populate bucket entries, a peer periodically contacts its neighbor peers, i.e. peers in its buckets, both indicating its own liveness and asking them for identities of other peers. The control message for a peer thus includes these probing messages as well as replies from other peers. The control message overhead, the freshness of bucket peers, and eventually, the efficiency of searches, would all be affected by the frequency of this probing.

Figure 17 illustrates this well. We vary the probing interval from 200 to 3,200 seconds, doubling it at each step. As expected, the average control message overhead of a peer proportionally decreases as one increases the probing interval. As shown in the figure, the reduced control overhead comes at the cost of worse bucket "freshness" and slower query resolution, as indicated by the average number of iterations a query takes. Nevertheless, even when using a probing interval of 3,200 seconds and with a bucket freshness of 0.4, we can still resolve queries very efficiently within a few iterations. This degree of robustness and efficiency can be largely attributed to the construction of bucket peers. Each bucket contains multiple peer entries; even if some peer entries become stale, we can still rely on the remaining fresh entries for the search. Missing or stale peer entries can be easily replaced since prefix matching (Kademlia's XOR metric) provides enough flexibility for many peers to qualify for a bucket entry. On the other hand, even when using a short probing interval of 200 seconds, the overhead is smaller than 0.5 control messages per second. Control overhead clearly benefits from the bucket entry flexibility and the lazy or periodic repair of buckets in the face of churn. In the following analysis, we employ a probing interval of 800 seconds.

We also explore through simulation how effectively one could balance the load imposed by popular keyword searches. We compare two approaches to object pointer placement, one naive where we simply put pointers to objects on the peer that shares the longest prefix with the keyword, i.e., the "root" peer, and another load-balance conscious one. The load-balancing algorithm applies to any peer x that stores object pointers containing a popular keyword K as follows:

1. Peer x periodically checks its query load for the keyword K. Assume the common prefix length of x and K is L bits.

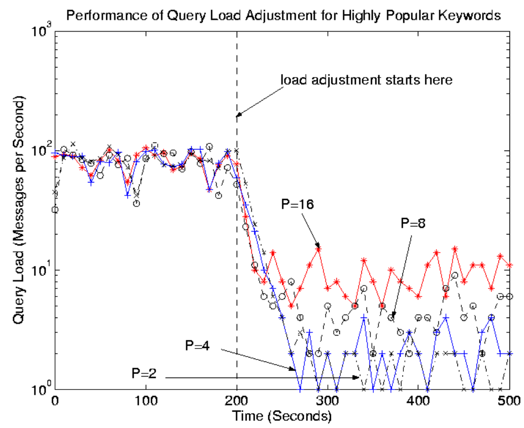

Figure 18: Query load adjustment for the most popular keywords under different load threshold P.

2. If the load is smaller than certain threshold P, go back to step 1. If the load exceeds the threshold, find n other peers whose hashes shared a common prefix of at least M bits with K. The initial value of n is set to 2, and M is set to $L-1$.

3. Store (some) of the object pointers to these n peers.

4. Set $M = M-1$, and $n = n*2$, and start from step 1 again.

For each round of the load-balancing algorithm, we distribute the keyword searching load to more peers by reducing M, the length of the required common prefix between the keyword K and the potential peers that share the load, by 1. Each round thus increases the number of peers that share the query load by a factor of 2, achieving fast and effective load balancing. Even for the most popular keywords, the load balancing algorithm can be finished within a few rounds.

Figure 18 illustrates the responsiveness of load balancing for keyword searches, where we show the average query load for peers that store pointers for a keyword with top 0.1% popularity. We begin by strictly limiting the data pointers on the "root" peer for the keyword. After 200 seconds (shown with a vertical line in the graph), we start the load balancing algorithm by putting object pointers on more qualified peers. The search loads of the keyword under different thresholds P, as indicated by the arrows, can all be reduced to below or around the desired level within 100 seconds.

Load balancing for searches in Overnet is essentially achieved by scattering query load across multiple peers. The more popular a keyword, the more peers will be involved in sharing the load. Figure 19 shows the CDF of common prefix lengths for IDs of peers that answer a keyword search, where each curve corresponds to a set of keywords with certain level of popularity. The load upperbound for keyword search is set to 4 messages per

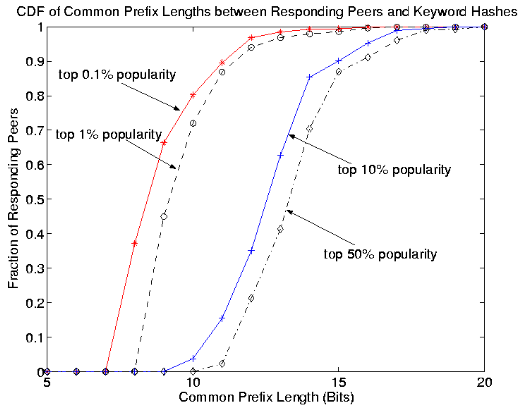

Figure 19: CDF of common prefix lengths between hash keys of responding peers with keyword hashes for keys with different popularities.

second. As expected, the more popular a keyword, the shorter the common prefix length: the common prefix lengths corresponding to the top 0.1% popular keywords are usually 5 to 7 bits shorter than those for the top 50% popular keywords. In other words, we would be able to get query answers from 32 to 128 times more peers for the most popular keywords. Note that our simulation results here are consistent with our measurement-based findings shown in Figure 13.

Load balancing in an Overnet-like system also yields two other desirable consequences. Since object pointers are replicated across multiple peers, there is no need to react to nodes' joins and leaves. These pointers can be lazily replicated to new-born peers to compensate for the pointers lost because of dead peers. In addition, replicated pointers at multiple peers help reduce the average iteration steps that a peer has to take to resolve a query. We have verified this through simulations, but omit our results for brevity.

9 Related Work

The work by Castro et al. [5] compares both unstructured and structured overlay via trace-based simulation, focused on leveraging structural constraints for lower maintenance overhead, exploiting heterogeneity to match different peer capacities and novel techniques of performing floodings or random walks on structured overlays for complex queries. Our study complements their work with the characterization, through extensive measurements, of two operational P2P systems, trying to gain additional insights into the design and implementation of more scalable and efficient peer-to-peer systems.

A lot of measurement works has been done to improve our understanding of different aspects of peer-to-peer file-sharing networks. Saroiu et al. [30] were among the first to perform a detailed measurement study of Napster and Gnutella file-sharing systems using crawlers and probers. In their later works, Gummadi et al. [9] performed a larger-scale P2P workload study for Kazaa at the border routers of the University of Washington, with the main focus on the object content retrieval aspect of the system. Sen et al. [31] also performed aggregated P2P traffic measurement by analyzing flow-level information collected at multiple border routers across a large ISP network, and revealed highly skewed traffic distribution across the network at three different aggregation levels and high dynamics of the three measured P2P systems. Bhagwan et al. [2] present an earlier study of Overnet as they question the usefulness of session times in understanding the availability of objects in a P2P file sharing network. Our study focuses on the measurement-based characterization of two unstructured and structured P2P networks in terms of resilience to churn, control traffic overhead, query performance and scalability. To the best of our knowledge, this is the first study discussing the advantages/disadvantages of each approach based on actual measurement of operational systems.

The more strict routing table and rules in structured P2P systems (i.e., DHTs) compared with unstructured ones motivate the common concern on the potentially large control message overhead and degraded routing performance of structured systems under churn. Bamboo [27] addresses this issue by relying on static resilience to failures, accurate failure detection and congestion-aware recovery mechanisms. Li et al. [16] present a performance versus cost framework (PVC) for evaluating different DHT algorithms in the face of churn. Some other works handling churn include [17, 19]. Our measurement study shows that Overnet does a very good job at handling the level of churn faced by a file-sharing network, yielding low control message overhead and good query performance. It is also interesting to note that Overnet (or Kademlia) already incorporates some important design lessons from previous works on resilient DHTs, including the use of periodic instead of proactive routing structure repair [27], parallel lookups [16], and flexible neighbor selection [5].

Given their key role in the functionality of this class of systems, queries have been actively studied for both the unstructured and structure approaches. A number of proposals have been made to enhance the scalability of queries in Gnutella-like systems. Lv et al. [18] proposes replacing flooding with random walks for queries in such systems. Gia [6] further adopts the idea of topology adaptation, capacity-aware flow control, and biased random walks to both improve query performance and scalability. Our previous work [3, 24] presents additional measurements of Gnutella's peer session lengths and suggests organizational protocols and query-related strategies that take advantage of their distribution to increase system scalability and boost query performance. The reported measurement of Gnutella shows that flooding does indeed result in high overhead, especially for high degree ultrapeers. Applying ideas such as those proposed in [6] and [24] could help improve system scalability.

Some recent work has focused on keyword searches in structured systems [26, 10, 34]. Most of the proposed solutions include the hashing of a keyword to a peer with the closest hash, and the storing of pointers to all objects containing the keyword at this peer. Perhaps one of the biggest concerns with this approach is the load on the peer that stores the pointers, especially if this peer is responsible for very popular keywords. Overnet uses a similar idea to support keyword search, except that instead of mapping a keyword to one particular peer, it replicates object pointers around a collection of peers whose hash IDs are close enough to that of the keyword. As we have shown in Section 6, this implicit load-balancing mechanism turns out to be very effective in distributing the load of popular keyword searches across multiple peers. As a result, keyword searches in Overnet are lightweight, effective, and achieve a balanced load.

10 Conclusion

We presented a multiple-site, measurement-based study of two operational and widely-deployed file-sharing systems. The two protocols were evaluated in terms of resilience, message overhead, and query performance. In general, our findings show that the structured overlay approach, as represented by Overnet, offers good scalability and efficiency in supporting file sharing networks. For example, while both Gnutella and Overnet show relatively low control message overhead, Overnet's overhead is particularly low in part as a result of its flexible bucket management policies. Also, both approaches seem to support fast keyword searches with Overnet successfully guaranteeing efficient, non-trivial keyword searching by leveraging the underlying DHT infrastructure and its $O(log(n))$ lookup mechanism. Finally, while Gnutella's ultrapeers can adjust their load by changing their outdegree, Overnet's peers implicitly distribute and replicate popular object pointers across multiple peers, achieving excellent load balancing and faster query replies.

We trust that the measurement and characterization presented in this paper will help the community to better understand the advantages and disadvantages of each approach as well as provide useful insights into the general design and implementation of new peer-to-peer systems.

Acknowledgments

We would like to thank Jeanine Casler, and David Choffnes for their invaluable help. In addition, we thank our shepherd Stefan Sariou and the anonymous reviewers who provided us with excellent feedback.

References

[1] BAWA, M., DESHPANDE, H., AND GARCIA-MOLINA, H. Transience of peers and streaming media. In *Proc. of the 1st Workshop on Hot Topics in Networks (HotNets)* (2002).

[2] BHAGWAN, R., SAVAGE, S., AND VOELKER, G. M. Understanding availability. In *Proc. of 2nd International Workshop on Peer-to-Peer Systems (IPTPS)* (2003).

[3] BUSTAMANTE, F. E., AND QIAO, Y. Friendships that last: Peer lifespan and its role in P2P protocols. In *Proc. of 8th International Web Content Caching and Distribution Workshop (WCW)* (2003).

[4] CASTRO, M., COSTA, M., AND ROWSTRON, A. Performance and dependability of structured peer-to-peer overlays. In *Proc. of the International Conference on Dependable Systems and Networks (DSN)* (2004).

[5] CASTRO, M., COSTA, M., AND ROWSTRON, A. Debunking some myths about structured and unstructured overlays. In *Proc. of the 2nd Symposium on Networked Sys. Design and Impl. (NSDI)* (2005).

[6] CHAWATHE, Y., RATNASAMY, S., BRESLAU, L., AND SHENKER, S. Making Gnutella-like P2P systems scalable. In *Proc. of SIGCOMM* (2003).

[7] CLIP2. The Gnutella protocol specification v0.4. RFC, The Gnutella RFC, 2000.

[8] GNUTELLA. Gnutella: Distributed information sharing. http://gnutella.wego.com, 2003.

[9] GUMMADI, K. P., DUNN, R. J., SAROIU, S., GRIBBLE, S. D., LEVY, H. M., AND ZAHORJAN, J. Measurement, modeling, and analysis of a peer-to-peer file-sharing workload. In *Proc. of 19th Symposium on Operating Systems Principles* (2003).

[10] HARREN, M., HELLERSTEIN, J. M., HUEBSCH, R., LOO, B. T., SHENKER, S., AND STOICA, I. Complex queries in dth-based peer-to-peer networks. In *Proc. of 1st International Workshop on Peer-to-Peer Systems (IPTPS)* (2002).

[11] KADC HOMEPAGE. KadC. http://kadc.sourceforge.net, 2005.

[12] KARAGIANNIS, T., BROIDO, A., FALOUTSOS, M., AND CLAFFY, K. Transport layer identification of P2P traffic. In *Proc. of ACM SIGCOMM Internet Measurement Conference* (2004).

[13] KAZAA. http://www.kazaa.com. 2001.

[14] KLINGBERG, T., AND MANFREDI, R. Gnutella 0.6. RFC, The Gnutella RFC, 2002.

[15] LI, J., STRIBLING, J., MORRIS, R., GIL, T., AND KAASHOEK, F. Routing tradeoffs in peer-to-peer DHT systems with churn. In *Proc. of 3rd International Workshop on Peer-to-Peer Systems (IPTPS)* (2004).

[16] LI, J., STRIBLING, J., MORRIS, R., KAASHOEK, M. F., AND GIL, T. M. A performance vs. cost framework for evaluating dht design tradeoffs under churn. In *Proc. of the IEEE Conference on Computer Communications (INFOCOM)* (2005).

[17] LIBEN-NOWELL, D., BALAKRISHNAN, H., AND KARGER, D. Analysis of the evolution of peer-to-peer systems. In *Proc. of the Symposium on Principles of Distributed Computing (PODC)* (2002).

[18] LV, Q., CAO, P., COHEN, E., LI, K., AND SHENKER, S. Search and replication in unstructured peer-to-peer networks. In *Proc. of ICS* (2002).

[19] MAHAJAN, R., CASTRO, M., AND ROWSTRON, A. Controlling the cost of reliability in peer-to-peer overlays. In *Proc. of 2nd International Workshop on Peer-to-Peer Systems (IPTPS)* (2003).

[20] MAYMOUNKOV, P., AND MAZIERES, D. Kademlia: A peer-to-peer information system based on the xor metric. In *Proc. of 1st International Workshop on Peer-to-Peer Systems (IPTPS)* (2002).

[21] MLDONKEY. MLDonkey Homepage. http://mldonkey.org, 2005.

[22] MUTELLA. http://mutella.sourceforge.net. 2003.

[23] OVERNET. http://www.overnet.com. 2003.

[24] QIAO, Y., AND BUSTAMANTE, F. E. Elders know best - handling churn in less structured P2P systems. In *Proc. of the IEEE International Conference on Peer-to-Peer Computing (P2P)* (Septeber 2005).

[25] RATNASAMY, S., FRANCIS, P., HANDLEY, M., KARP, R., AND SHENKER, S. A scalable content-addressable network. In *Proc. of SIGCOMM* (2001).

[26] REYNOLDS, P., AND VAHDAT, A. Efficient peer-to-peer keyword searching. In *Proc. of IFIP/ACM Middleware* (2003).

[27] RHEA, S., GEELS, D., ROSCOE, T., AND KUBIATOWICZ, J. Handling churn in a DHT. In *Proc. of USENIX Technical Conference* (June 2004).

[28] ROWSTRON, A., AND DRUSCHEL, P. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. In *Proc. of IFIP/ACM Middleware* (2001).

[29] SAROIU, S., GUMMADI, K. P., DUNN, R. J., GRIBBLE, S. D., AND LEVY, H. M. An analysis of Internet content delivery systems.

[30] SAROIU, S., GUMMADI, P. K., AND GRIBBLE, S. D. A measurement study of peer-to-peer file sharing systems. In *Proc. of Annual Multimedia Computing and Networking (MMCN)* (2002).

[31] SEN, S., AND WANG, J. Analyzing peer-to-peer traffic across large networks. In *Proc. of ACM SIGCOMM Internet Measurement Conference* (2002).

[32] SRIPANIDKULCHAI, K. The popularity of Gnutella queries and its implications on scalability. In *O'Reilly's OpenP2P* (2001).

[33] STOICA, I., MORRIS, R., KARGER, D., KAASHOEK, F., AND BALAKRISHNAN, H. Chord: A scalable peer-to-peer lookup service for internet applications. In *Proc. of SIGCOMM* (2001).

[34] TANG, C., XU, Z., AND DWARKADAS, S. Peer-to-peer information retrieval using self-organizing semantic overlay networks. In *Proc. of SIGCOMM* (2003).

[35] YANG, B., AND GARCIA-MOLINA, H. Efficient search in peer-to-peer networks. In *Proc. of the International Conference on Distributed Computing Systems (ICDCS)* (2002).

[36] ZHAO, B. Y., KUBIATOWICZ, J., AND JOSEPH, A. D. Tapestry: An infrastructure for fault-tolerant wide-area location and routing. Tech. Report UCV/CSD-01-1141, Computer Science Division, UC, Berkeley, 2001.

Notes

[1] The Kademlia protocol is discussed in detail in [20].

[2] Evanston, USA; Zurich, Switzerland; Paris, France and Beijing, China.