Efficient Content Location Using Interest-Based Locality in Peer-to-Peer Systems

Kunwadee Sripanidkulchai Bruce Maggs Hui Zhang
Carnegie Mellon University, Pittsburgh, PA 15213
{kunwadee,bmm,hzhang}@cs.cmu.edu

***Abstract*— Locating content in decentralized peer-to-peer systems is a challenging problem. Gnutella, a popular file-sharing application, relies on flooding queries to all peers. Although flooding is simple and robust, it is not scalable. In this paper, we explore how to retain the simplicity of Gnutella, while addressing its inherent weakness: scalability. We propose a content location solution in which peers loosely organize themselves into an interest-based structure on top of the existing Gnutella network. Our approach exploits a simple, yet powerful principle called *interest-based locality*, which posits that if a peer has a particular piece of content that one is interested in, it is very likely that it will have other items that one is interested in as well. When using our algorithm, called *interest-based shortcuts*, a significant amount of flooding can be avoided, making Gnutella a more competitive solution. In addition, shortcuts are modular and can be used to improve the performance of other content location mechanisms including distributed hash table schemes.**

We demonstrate the existence of interest-based locality in five diverse traces of content distribution applications, two of which are traces of popular peer-to-peer file-sharing applications. Simulation results show that interest-based shortcuts often resolve queries quickly in one peer-to-peer hop, while reducing the total load in the system by a factor of 3 to 7.

I. INTRODUCTION

Ensuring the availability of content on the Internet is expensive. Publishers who want high availability have few options. They can use premium content hosting services, build and manage their own content distribution infrastructures, or contract with Content Delivery Networks [1]. All of these options are prohibitively expensive for an average Internet user who wants to share a hundred megabytes of digital photographs with his friends. A low cost solution is to publish the content from one's own desktop into a peer-to-peer content distribution system like Gnutella [2]. While peers are downloading content, they can also create and make available replicas to increase content availability. As the system grows, the supply of resources scales with demand. There are enough resources, even during flash crowds when many people access the same content simultaneously.

There are many challenges in providing peer-to-peer content distribution systems. In this paper, we address one fundamental challenge: what is the appropriate strategy for locating content given that content may be continuously replicated at many locations in the peer-to-peer system? If content cannot be located efficiently, there is little hope for using peer-to-peer technology for content distribution.

There are two classes of solutions currently proposed for decentralized peer-to-peer content location. Unstructured content location, used by Gnutella, relies on flooding queries to all peers. Peers organize into an overlay. To find content, a peer sends a query to its neighbors on the overlay. In turn, the neighbors forward the query on to all of their neighbors until the query has traveled a certain radius. While this solution is simple and robust even when peers join and leave the system, it does not scale. Another class of protocols based on the Distributed Hash Table (DHT) abstraction [3] [4] [5] [6] and motivated by Plaxton et al. [7] have been proposed to address scalability. In these protocols, peers organize into a well-defined structure that is used for routing queries. Although DHTs are elegant and scalable, their performance under the dynamic conditions common for peer-to-peer systems is unknown [8].

Our design philosophy departs from existing work in that we seek to retain the simple, robust, and fully decentralized nature of Gnutella, while improving scalability, its major weakness. We identify a powerful principle: if a peer has a particular piece of content that one is interested in, then it is likely that it will have other pieces of content that one is also interested in. These peers exhibit *interest-based locality*. We propose a self-organizing protocol, *interest-based shortcuts*, that efficiently exploits interest-based locality for content location. Peers that share similar interests create shortcuts to one another. Peers then use these shortcuts to locate content. When shortcuts fail, peers resort to using the underlying Gnutella overlay. Shortcuts provide a *loose* structure on top of Gnutella's unstructured overlay. Although we use Gnutella as the primary example in this paper, shortcuts are also compatible with many other content location mechanisms, such as DHTs and hybrid centralized-decentralized architectures such as Kazaa [9].

We compare the performance of Gnutella with and without shortcuts using five traces collected at different locations and different times. We show that shortcuts significantly improve the performance of content location for Gnutella by providing large decreases in the amount of load in the system and the time to locate content. We find that simple algorithms are sufficient to capture interest-based relationships. Moreover, we provide some intuition on the factors that contribute to the degree of locality observed in our traces.

This research was sponsored by DARPA under contract number F30602-99-1-0518, and by NSF under grant numbers Career Award NCR-9624979 ANI-9730105, ITR Award ANI-0085920, and ANI-9814929. Additional support was provided by Intel. Views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of DARPA, NSF, Intel, or the U.S. government.

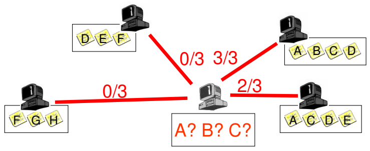


Fig. 1. Peers that share interests.

In Section II, we describe the design of interest-based shortcuts. In Section III, we present our evaluation metrics and simulation methodology. We present our results in Section IV and explore the potential and limitations of shortcuts in Section V. Section VI takes an in-depth look at factors that contribute to interest-based locality. In Section VII, we look at the effectiveness of shortcuts for a DHT-based protocol, Chord [5]. We discuss the implications of our results in Section VIII, and related work in Section IX.

II. Interest-based Locality

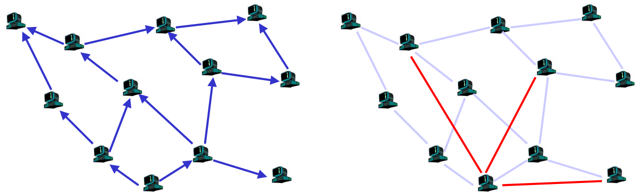
In this section, we present a technique called interest-based shortcuts. We will show in Section IV that this technique, while based on simple principles, can significantly improve the performance of Gnutella.

Figure 1 gives an example to illustrate interest-based locality. The peer in the middle is looking for files A, B, and C. The two peers in the right who have file A also each have at least one more matching file B or C. The peer on the upper right-hand corner has all three files. Therefore, it and the peer in the middle share the most interests, where interests represent a group of files, namely $\{A, B, C\}$. Our goal is to identify such peers, and use them for downloading files directly.

A. Shortcuts Architecture and Design Goals

We propose a technique called shortcuts to create additional links on top of a peer-to-peer system's overlay, taking advantage of locality to improve performance. Shortcuts are implemented as a separate performance enhancement layer on top of existing content location mechanisms, such as flooding in Gnutella. The benefits of such an implementation are two-fold. First, shortcuts are modular in that they can work with any underlying content location scheme. Second, shortcuts only serve as performance-enhancement hints. If a document cannot be located via shortcuts, it can always be located via the underlying overlay. Therefore, having a shortcut layer does not affect the correctness and scalability of the underlying overlay. In general, shortcuts are a powerful primitive that can be used to improve overlay performance. For example, shortcuts based on network latency can reduce hop-by-hop delays in overlay networks. In this paper, we explore the use of a specific kind of shortcut based on interests, for content location.

Figure 2(a) illustrates how content is located in Gnutella. A query initiated by the peer at the bottom is flooded to all peers in the system. Figure 2(b) depicts a Gnutella overlay with 3 shortcut links for the bottom-most peer. To avoid flooding, content is located first through shortcuts. A query is flooded to the entire system only when none of the shortcuts have the content.



(a) Gnutella. (b) Shortcuts.

Fig. 2. Content location paths.

Our design goals for interest-based shortcuts are simplicity and scalability. Peers should be able to detect locality in a fully-distributed manner, relying only on locally learned information. Algorithms should be lightweight. In addition, the dynamic nature of peer-to-peer environments requires that the algorithm be adaptive and self-improving. We incorporate the above considerations into our design, which has two components: shortcut discovery and shortcut selection.

B. Shortcut Discovery

We use the following heuristic to detect shared interests: peers that have content that we are looking for share similar interests. Shortcut discovery is piggy-backed on Gnutella. When a peer joins the system, it may not have any information about other peers' interests. Its first attempt to locate content is executed through flooding. The lookup returns a set of peers that store the content. These peers are potential candidates to be added to a "shortcut list." In our implementation, one peer is selected at random from the set and added. Subsequent queries for content go through the shortcut list. If a peer cannot find content through the list, it issues a lookup through Gnutella, and repeats the process for adding new shortcuts. Peers passively observe their own traffic to discover their own shortcuts. For scalability, each peer allocates a fixed-size amount of storage to implement shortcuts. Shortcuts are added and removed from the list based on their perceived utility, which is computed using the ranking algorithm described in Section II-C. Shortcuts that have low utility are removed from the list when the list is full.

There are several design alternatives for shortcut discovery. New shortcuts may be discovered through exchanging shortcut lists between peers, or through establishing more sophisticated link structures for each content category similar to structures used by search engines. In addition, multiple shortcuts, as opposed to just one, may be added to the list at the same time. In Section IV, we study a basic approach in which one shortcut is added at a time, based on results returned from Gnutella's flooding. In Section V, we explore the potential of two optimizations: adding k shortcuts at a time and learning about new shortcuts through one's current shortcuts.

C. Shortcut Selection

Given that there may be many shortcuts on the list, which one should be used? In our design, we rank shortcuts based on their perceived utility. If shortcuts are useful, they are ranked at the top of the list. A peer locates content by sequentially asking all of the shortcuts on its list, starting from the top, until content is found. Rankings can be based on many metrics, such as probability of providing content, latency of the path to the shortcut, available bandwidth of the path, amount of content at the shortcut, and load at the shortcut. A combination of metrics can be used based on each peer's preference.

Each peer continuously keeps track of each shortcut's performance and updates its ranking when new information is learned. This allows for peers to adapt to dynamic changes and incrementally refine shortcut selection. In Section IV, we explore the use of probability of providing content (success rate) as a ranking metric. In this context, success rate is defined as the ratio between the number of times a shortcut was used to successfully locate content to the total number of times it was tried. The higher the ratio, the better the rank on the list.

III. Performance Evaluation

In this section, we discuss the design of experiments to expose interest-based locality and evaluate the effectiveness of our proposed shortcuts scheme. We start by giving a brief overview of Gnutella. We then discuss the performance indices we use for our evaluation, and describe our methodology and experimental setup.

A. Gnutella Content Location

Gnutella uses flooding to locate content. Each query is tagged with a maximum Time-To-Live (TTL) to bound the number of hops it can travel. In addition, Gnutella employs a duplicate query detection mechanism so that peers do not forward queries that they have already previously forwarded. Despite such mechanisms, some amount of duplication is inherent to flooding algorithms and cannot be avoided. Peers reply to a query when the query string matches partially, or exactly, to files stored on their disk drives.

B. Performance Indices

The metrics we use to express the benefits and overhead of interest-based shortcuts are:

1) Success rate: How often are queries resolved through shortcuts? If success rates are high, then interest-based locality techniques have the potential to improve performance.

2) Load characteristics: How many query packets do peers process while participating in the system? Reducing the load at individual peers is desirable for scalability.

3) Query scope: For each query, what fraction of peers in the system are involved in query processing? A smaller query scope increases system scalability.

4) Minimum reply path lengths: How long does it take for the first reply to come back?

5) Additional state: How much additional state do peers need to maintain in order to implement shortcuts? The amount of state measures the cost of shortcuts and should be kept to a minimum.

C. Methodology

We use trace-based simulations for our performance evaluation. First, we discuss our query workloads. Next, we describe how we construct the underlying Gnutella overlay that is used for flooding queries, and map peers from the query workload onto nodes in the Gnutella overlay. We then discuss our storage and replication models, and our simulation experiments.

1) Query workloads: We use five diverse traces of download requests from real content distribution applications to generate query workloads. Our first three traces (labeled Boeing, Microsoft and CMU-Web in Table I) capture Web request workloads, which we envision to be similar to requests in Web content file-sharing applications [10] [11] [12] [13]. Our last two traces (labeled CMU-Kazaa and CMU-Gnutella in Table I) capture requests from two popular file-sharing applications, Kazaa and Gnutella.

The Boeing trace [14] is composed of one-day traces from five of Boeing's firewall proxies from March 1, 1999. The Microsoft trace is composed of one-day traces from Microsoft's corporate firewall proxies from October 22, 2001. The CMU-Web, CMU-Kazaa and CMU-Gnutella traces are collected by passively monitoring the traffic between Carnegie Mellon University and the Internet over a 24-hour period on October 22, 2002. Our monitoring host is connected to monitoring ports of the two campus border routers. Our monitoring software, based on tcpdump [15], installs a kernel filter to match packets containing an HTTP request or response header, regardless of port numbers. Although an HTTP header may be split across multiple packets, we find that it happens rarely (0.03% of packets). The packet filter was able to keep up with the traffic, dropping less than 0.026% of packets. We extend tcpdump to parse the packets online to extract source and destination IP addresses and ports, request URL, response code, content type, and cachability tags. We anonymize IP addresses and URLs, and log all extracted information to a log file on disk. Our trace consists of all Web transactions (primarily port 80), Kazaa downloads (port 1214), and Gnutella downloads (primarily port 6346) between CMU and the rest of the Internet.

Given the download requests in our traces, we generate query workloads in the following way: if peer P_1 downloads file A (or URL A) at time t_0, peer P_1 issues a query for file A at time t_0. We model the query string as the full URL, A, and perform exact matching of the query string to filenames. We assume that P_1's intention is to search for file A, and all hosts with file A will respond to the query. Not modeling partial matches does not affect our results for the Web or CMU-Kazaa query workloads as a URL typically corresponds to a distinct piece of content. However, URLs in the CMU-Gnutella workload are based on filenames, which may not correspond to distinct pieces of content. For example, a file for my favorite song by my favorite artist could be named "my favorite song" or "my favorite song, my favorite artist." In our simulations, these two files would be considered different, although they are semantically the same. We use exact matches because it is difficult to partially match over anonymized names. As a

TABLE I
TRACE CHARACTERISTICS.

Trace	Characteristics	1	2	3	4	5	6	7	8
Boeing	Requests	95,504	95,429	166,741	201,862	1,176,153	1,541,062	1,617,608	2,039,347
	Documents	42,800	44,153	75,833	79,306	305,092	391,229	434,766	513,264
	Clients	868	1,052	1,443	2,278	18,059	21,690	22,344	25,293
Microsoft	Requests	764,177	917,325	960,119	1,588,045	2,083,911	3,818,368	4,515,815	6,671,774
	Documents	102,548	164,505	198,559	285,711	416,784	662,986	718,444	956,617
	Clients	11,636	11,929	13,013	15,387	19,419	23,492	28,741	32,361
CMU-Web	Requests	125,138	104,781	132,405	155,847	338,656	358,778	432,843	495,119
	Documents	61,569	43,616	61,981	72,513	162,951	153,405	190,372	211,570
	Clients	6,322	6,426	7,054	7,602	11,176	12,274	13,892	15,408
CMU-Kazaa	Distinct Requests	7,757	7,779	8,086	9,075	9,243	13,307	13,760	15,188
	Documents	3,720	3,625	3,806	4,338	4,771	6,619	7,172	6,312
	Clients/Peers	6,482/6,985	6,514/6,968	6,732/7,217	7,468/8,064	7,601/8,542	10,977/11,983	11,362/12,660	12,558/13,590
CMU-Gnutella	Distinct Requests	392	389	395	415	480	502	581	884
	Documents	260	247	239	254	318	339	393	609
	Clients/Peers	256/464	270/383	271/373	296/405	320/543	341/477	383/590	542/735

result, it is likely that we underreport the number of peers who have a particular file, and overestimate the number of distinct files in the system.

We randomly selected eight one-hour segments from each query workload to use for our simulations. We limit our experiments to one hour, the median session duration reported for peer-to-peer systems [16]. The characteristics of all trace segments are listed in Table I, sorted by number of clients.

2) Gnutella connectivity graphs: Next, we discuss how we construct the underlying Gnutella overlay used for flooding queries, and how we map peers in the query workload described in the previous section to nodes in the Gnutella overlay.

To simulate the performance of Gnutella flooding, we use Gnutella connectivity graphs collected in early 2001. All graphs have a bimodal power-law degree distribution with an average degree of 3.4. The characteristic diameter is small at 12 hops. In addition, over 95% of the nodes are at most 7 hops away from one another. The number of nodes in each graph vary from 8,000 to 40,000 [17]. For simulations, we selected Gnutella graphs that had the closest number of peers to the ones in each one-hour trace segment. Then, nodes were randomly removed from the graph until the number of nodes matched. The resulting graphs and the original graphs had similar degree distribution and pair-wise path length characteristics. Peers from each one-hour segment were randomly mapped to nodes in the Gnutella graphs. We used a maximum query TTL of 7, which is the application default for many Gnutella clients. Although it is possible that some content cannot be found because of the TTL limit, this was a problem for less than 1% of the queries.

3) Storage and replication model for Web query workloads: Next, we describe how content is placed and stored in the system. For each trace segment, we assume that all Web clients participate in a Web content file-sharing system. To preserve locality, we place the first copy of content at the peer who makes the first request for it (i.e., this is a publish to the system, and a query lookup is not performed). Subsequent copies of content are placed based on accesses. That is, if peer P_1 downloaded file A at time t_0, P_1 creates a replica of file A and make it available for others to download after time t_0. Peers store all the content that they retrieve during that trace segment, and make that content available for other peers to download. Any request for content that a peer has previously downloaded (i.e., a repeated request) is satisfied locally from the peer's cache.

Only requests for static content in the Microsoft trace and the CMU-Web trace are used in our evaluation. Specifically, we removed requests for content that contained "cgi," ".asp," ".pl," "?," and query strings in the URL. In addition, for the CMU-Web trace we removed all requests for uncachable content as specified by the HTTP response headers, following the HTTP 1.1 protocol. The Microsoft trace did not have HTTP response header information. The Boeing trace did not contain sufficient information to distinguish between static and dynamic content. Therefore, all Boeing requests were used in our analysis.

4) Storage and replication model for CMU-Kazaa and CMU-Gnutella query workloads: We draw a distinction between two types of peers in the traces: peers that only serve files, and peers that download files. Peers that only serve files do not issue requests for content in the trace, but provide a set of files for which other peers may download. It is likely that these are hosts outside of CMU who are providing files to hosts at CMU, or hosts at CMU that are not actively downloading any files. We assume that any peer that downloads files must make those files available for other peers. Table I lists the number of clients (peers that download files) and the total number of peers (both types) in each trace segment. Both types of peers are participants in the peer-to-peer system, but only peers who download content issue queries in the simulation.

Before running the simulation, we make one pass through each trace segment and build up a list of content available at each peer. Specifically, if a peer P_1 served a file A at some time t_0 in the trace segment, we assume that P_1 makes that file available for any other peer to download any time during that trace segment, even before t_0. This simulates a peer that has the file on disk before the beginning of the trace segment. However, if P_1 originally obtained file A by a download earlier in the trace, we make sure that A is available for other peers to download only after P_1 has downloaded it. We have only partial knowledge about content available at each peer because we are limited by the information present in the trace. For example, let's assume that peer P_1 has a copy of file B on disk. However, P_1 did not download the file during the trace segment. In addition, no other peer downloaded the file from him, either. Therefore, we have no information in the trace that P_1 has file B. When P_2 sends a query looking for file B in our simulations, P_1 would not reply although in reality P_1 has the file. As a result, we underestimate the number of peers who could potentially supply a file, and report pessimistic results for the CMU-Kazaa and CMU-Gnutella workloads.

Queries are performed only for distinct requests. For example, a Kazaa peer usually downloads multiple fragments of a file from multiple peers in parallel and may issue multiple HTTP requests for that one file. In our simulations, that peer issues only one query to find that file.

We assume that all peers in the trace segment participate in the file-sharing session, including peers outside of CMU downloading files from peers at CMU. We also ran a set of experiments where we looked only at peers at CMU downloading content and found that the results were similar to using all peers in the trace. We present results when using all peers in the trace in the following sections.

5) Simulation experiments: We compare the performance of Gnutella, and Gnutella with shortcuts for each query workload. For each portion of the trace, we assume that peers that send any queries join the system at the beginning of the segment and stay until the end. Although participation is static in our simulations, we discuss the effects of more dynamic conditions in Section IV-B. Unless otherwise stated, peers maintain a fixed-size list of 10 shortcuts. Shortcuts are ranked based on success rates.

IV. Experimental Results

A. Comparison with Gnutella

In this section, we present evaluation results comparing the performance of Gnutella against Gnutella with shortcuts.

1) Success Rate: Success rate is defined as the number of lookups that were successfully resolved through interest-based shortcuts over the total number of lookups. If the success rate is high, then shortcuts are useful for locating content. Note that peers who have just joined the system do not have any shortcuts on their lists, and have no choice but to flood to locate the first piece of content. We start counting the success rate after the first flood (i.e., when peers have one shortcut on their list).

Figure 3(a) depicts the average success rate for shortcuts for each query workload. The vertical axis is the success rate, and the horizontal axis is the time after the start of the simulation when the observation was made. The average success rate at the end of 1 hour is as high as 82%-90% for the Web workloads, and 53%-58% for the CMU-Gnutella and CMU-Kazaa workloads.[1] The individual success rate (not depicted) observed at each peer increases with longer simulation times as peers learn more about other peers and have more time to refine their shortcut list. Although success rates for all workloads are reasonably high, success rates for Web workloads are distinctly higher than those for the CMU-Kazaa or CMU-Gnutella workloads. We believe that this is because we only have a partial view of the content available at each peer for the CMU-Kazaa/Gnutella workloads and are likely to see conservative results, as discussed in the previous section.

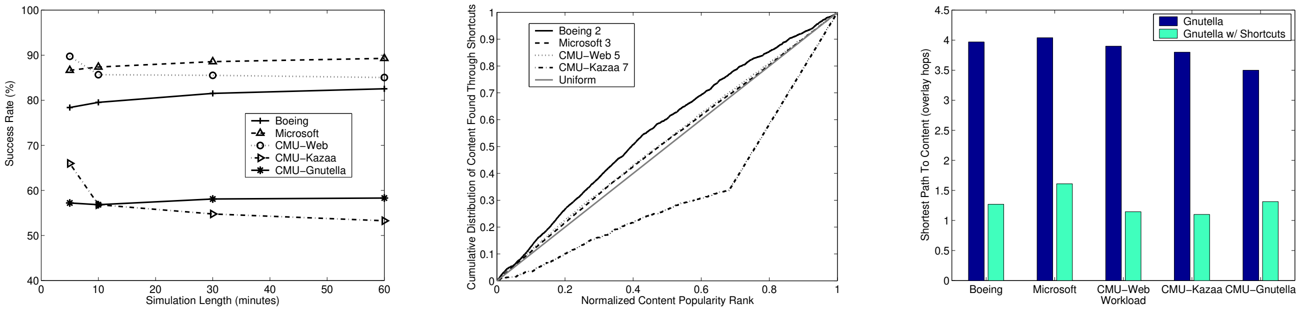
Next, we ask what kind of content is located through shortcuts? Are shortcuts useful for finding only popular content? Figure 3(b) depicts the cumulative probability of finding content with the specified popularity ranking through shortcuts. We present results from one representative trace segment from each query workload. The x-axis is content rank normalized by the total number of documents in the trace segment. The normalized rank values range from 0 (most popular) to 1 (least popular). Each document is classified as found or not found. That is, if content with rank 0 was found at least once through shortcuts, it is labeled as found. Only content that is found is depicted in the figure. A reference line for the uniform distribution, when all documents have equal probability of being found, is also given. We find that the distributions for the Microsoft and CMU-Web traces closely match the uniform distribution, indicating that shortcuts are uniformly effective at finding popular and unpopular content. The distribution for the Boeing trace is also close to a uniform distribution, but has a slight tendency towards finding more popular content. On the other hand, shortcuts tend to find more unpopular content in the CMU-Kazaa trace. The distribution on the right of the sharp inflection point represents finding extremely unpopular content that is shared by only two people. We do not present the results for CMU-Gnutella because there were not enough file accesses to determine document popularity. The most popular file was accessed by a handful of people.

2) Load and Scope: We achieve load reduction by using shortcuts *before* flooding so that only a small number of peers are exposed to any one query. We look at two metrics that capture load reduction: load at each peer and query scope. Less load and smaller scope can help improve the scalability of Gnutella.

Load is measured as the number of query packets seen at each peer. Table II lists the average load for Gnutella and Gnutella with shortcuts. Due to space limitations, we present results for the last 4 segments of the Boeing and Microsoft traces. For example, peers in Segment 5 of the Microsoft trace saw 479 query packets/second when using Gnutella. However, with the help of shortcuts, the average load is much less at 71 packets/second. Shortcuts consistently reduce the load across all trace segments. The reduction is about a factor of 7 for the Microsoft and CMU-Web trace, a factor of 5 for the Boeing trace, and a factor of 3 for the CMU-Kazaa and CMU-Gnutella traces.

We also look at the peak-to-mean load ratio in order identify hot spots in the system. The peak-to-mean ratio for flooding through Gnutella ranges from 5 to 12 across all traces, meaning that at some time during the experiment, the most loaded

[1] For comparison, we also conducted experiments to select random peers from *all participating peers* to add as shortcuts. Note that this is different from interest-based shortcuts where shortcuts are added based on replies from flooding through Gnutella. We find that the success rate for random shortcuts varied from 2-9% across all trace segments.



(a) Success rates of shortcuts.

(b) Popularity of content.

(c) Shortest path to content.

Fig. 3. The performance of interest-based shortcuts.

TABLE II

LOAD AT EACH PEER IN QUERY PACKETS/SECOND.

Trace	Protocol	5	6	7	8
Boeing	Gnutella Flooding	355.4	462.6	493.5	670.9
	Gnutella w/ Shortcuts	66.0	86.5	98.7	132.0
Microsoft	Gnutella Flooding	478.7	832.1	1,163.8	1,650.1
	Gnutella w/ Shortcuts	70.5	115.5	162.1	230.4

peer in the system saw 5 to 12 times more query packets than the average peer. For most trace segments, the peak-to-mean ratio for shortcuts is similar to Gnutella's, indicating that shortcuts do not drastically change the distribution of load in the system. However, for 3 segments in the Microsoft trace, the peak-to-mean ratio for shortcuts almost doubled compared to Gnutella. This is because shortcuts bias more load towards peers that have made a large number of requests. These peers have more content and are more likely to be selected as shortcuts compared to average peers. As a result, they tend to see more queries. We found that there were a number of peers that had significantly larger volumes of content in these 3 trace segments. Shortcuts have an interesting property that redistributes more load to peers that use the system more frequently. This seems to be fair as one would expect peers that make heavy use of the system to contribute more resources.

Scope for a query is defined as the fraction of peers in the system that see that particular query. For example, flooding has a scope of approximately 100% because all peers (except those beyond the TTL limit) see the query. Shortcuts, when successful, have a much smaller scope. Usually, only one shortcut will see a query, resulting in a query scope of less than 0.3%. When shortcuts are unsuccessful, then the scope is 100%, the same as flooding. Our results show that shortcuts are often successful at locating content and only a small number of peers are bothered for most queries.

3) Path Length: Path length is the number of overlay hops a request traverses until the first copy of content is found. For example, if a peer finds content after asking 2 shortcuts, (i.e., the first shortcut was unsuccessful), the path length for the lookup is 2 hops. Note that a peer locates content by sequentially asking shortcuts on its list. For Gnutella, path length is the minimum number of hops a query travels before it reaches a peer that has the content. Peers can directly observe an improvement in performance if content can be found in fewer hops. Figure 3(c) depicts the average path length in number of overlay hops for all workloads. On average, content is 4 hops away on Gnutella. Shortcuts, when successful, reduce the path length by more than half to only 1.5 hops. To further reduce the path length, all the shortcuts on the list could be asked in parallel as opposed to sequentially.

Next, we look at the amount of additional state required to implement shortcuts. On average, peers maintain 1-5 shortcuts. Shortcut lists tend to grow larger in traces that have higher volumes of requests. We placed an arbitrary limit on the shortcut list size to at most ten entries. Although we could have allowed the list to grow larger, it does not appear to be a limiting factor on performance.

We also look at opportunities for downloading content in parallel through multiple shortcuts and find that for all trace segments, 25%-50% of requests could have been downloaded in parallel through at least 2 shortcuts.

We summarize the results from our evaluation below:

- Shortcuts are effective at finding both popular and unpopular content. When using shortcuts, 45%-90% of content can be found quickly and efficiently.
- Shortcuts have good load distribution properties. The overall load is reduced, and more load is redistributed towards peers that make heavy use of the system. In addition, shortcuts help to limit the scope of queries.
- Shortcuts are scalable, and incur very little overhead.

Although all five workloads have diverse request volumes and were collected three years apart, they exhibit similar trends in interest-based locality.

B. Sensitivity to Participation Dynamics

In this section, we discuss the effects of participation dynamics on shortcuts. Dynamics can affect both the shortcut structure and the underlying Gnutella connectivity. We expect a peer's performance to be poorer if its shortcuts leave the system (or equivalently, die). However, interest-based structures are designed to be adaptive and the effect of a shortcut leaving the system is seen only for the lookup immediately following the leave. For example, peer P_1 uses peer P_2 as a shortcut, and P_2 just left the system. The next time P_1 sends a query to P_2, he will discover that P_2 has left, remove P_2 from his shortcut list, and ask his other shortcuts for content. In the worst case, P_1 may unfortunately and unknowing keep choosing shortcuts that leave the system, and will have to fall back to flooding for all queries. In this case, shortcuts are useless, but do not hurt Gnutella's flooding performance. Participation dynamics can also disrupt the performance of Gnutella by reducing the effectiveness of flooding. Even when the Gnutella network is partitioned, however, peers may still locate content through shortcuts.

V. Potential and Limitations of Shortcuts

In the previous section, we showed that simple algorithms for identifying and using interest-based shortcuts can provide significant performance gains over Gnutella's flooding mechanism. In this section, we explore the limits of interest-based locality by conducting experiments to provide insight on the following questions:

- What is the best possible performance when peers learn about shortcuts through past queries?
- Are there practical changes to the basic algorithm presented in the previous section that would improve shortcut performance to bring it closer to the best possible?
- Can we improve shortcut performance if we discover shortcuts through our existing shortcuts, in addition to learning from past queries?

In order to explore the best possible performance, we remove the practical limits imposed on the shortcuts algorithm evaluated in the previous section. First, peers add *all* peers returned from Gnutella's flooding as shortcuts. To contrast with the basic algorithm in the previous section, only *one* randomly selected peer was added at a time. Second, we removed the 10-entry limit on the shortcut list size and allowed the list to grow without bound.

Figure 4(a) depicts the best possible success rate averaged across all trace segments for all workloads. Also, note that the success rate is pessimistic for the CMU-Kazaa and CMU-Gnutella workloads as discussed previously. The average success rate at the end of 1 hour is as high as 97% and 65% for the Microsoft and CMU-Kazaa workloads. Although the upper-bound is promising, it is impractical for peers in the Boeing and Microsoft workloads because they need to maintain on average 300 shortcuts. Furthermore, the path length to the first copy of content is as long as tens of hops.

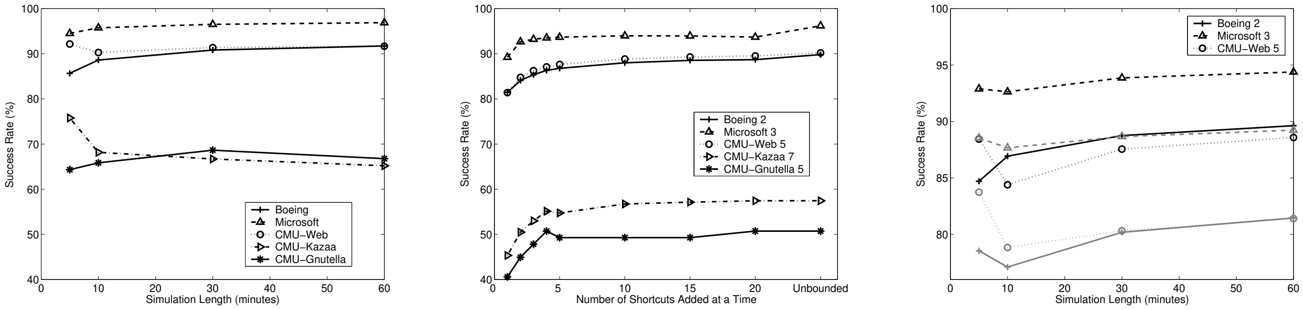
Rather than removing all practical constraints, we look at the performance when we relax some constraints to answer the second question posed at the beginning of this section. First, we observe that success rates for the basic shortcuts algorithm depicted in Figure 3(a) is only 7-12% less than the best possible. The basic algorithm, which is simple and practical, is already performing reasonably well. Now, we relax the constraints for adding shortcuts by adding k random shortcuts from the list of peers returned by Gnutella. Specifically, we looked at adding 2, 3, 4, 5, 10, 15, and 20 shortcuts at a time. We also changed the limit on the number of shortcuts each peer can maintain to at most 100.

Figure 4(b) depicts the success rates observed using this extended shortcuts algorithm. We report results for the segment with the lowest success rate when using the basic algorithm from each workload. The horizontal axis is k, the number of shortcuts added at a time, varying from 1 for the basic algorithm to "unbounded", where "unbounded" refers to adding as many shortcuts as possible for the best possible performance. The vertical axis is the success rate at the end of the 1-hour period. We find that the success rate increases when more shortcuts are added at a time. For instance, for segment 2 of the Boeing trace, when we add 5 shortcuts at a time, the success rate increases to 87% compared to 81% when adding 1 shortcut. Adding 5 shortcuts at a time produces success rates that are close to the best possible. Furthermore, we see diminishing returns when adding more than 5 shortcuts at a time. We find that the load, scope, and path length characteristics when adding 5 shortcuts at a time is comparable to adding 1 shortcut at a time. The key difference is the shortcut list size, which expands to about 15 entries. This is a reasonable trade-off for improving performance.

Next, we answer the third question. An additional improvement to the shortcut algorithm is to locate content through the shortcut structure in the following way: peers first ask their shortcuts for content. If none of their shortcuts have the content, they ask their shortcuts' shortcuts. This can be viewed as sending queries with a TTL of 2 hops along the shortcut structure. In our implementation, peers send queries to each peer in the shortcut structure sequentially until content is found. If content is found at a peer who is not currently a shortcut, it gets added to the list as a new shortcut. Peers resort to Gnutella only when content cannot be found through the shortcut structure. We believe this could be an efficient way to learn about new shortcuts without needing to excessively flood through Gnutella. In addition, we hope to capitalize on the situation depicted in Figure 5. The peer on the left has previously downloaded file A, and has added the peer in the middle as a shortcut. It now wants to find file C. Its immediate shortcut does not have file C, but its shortcut's shortcut which is the peer on the right, has file C. In this case, the peer on the left can successfully locate content through its shortcut's shortcut.

Figure 4(c) depicts the success rates when using this algorithm for locating content. The vertical axis is success rate and the horizontal axis is the time the observation was made during the simulation. The gray lines, given as reference points, represent the success rates when using the basic algorithm. Again, we limit the shortcut list size to 10 entries. The success rates for discovering new shortcuts through existing shortcuts is higher than the basic algorithm. For segment 2 of the Boeing trace, the success rate increased from 81% to 90% at the end of the hour. And similarly, the success rate increased from 89% to 95%, and 81% to 89% for segment 3 of the Microsoft trace and segment 5 of the CMU-Web trace, respectively. In addition, the load is reduced by half. However, the shortest path length to content increases slightly to 2 hops. The results for the CMU-Kazaa and CMU-Gnutella traces have similar trends.

Our results show that the basic algorithm evaluated in the previous section performs reasonably well. In addition, a few practical refinements to the basic algorithm can yield further



(a) Add as many shortcuts as possible.

(b) Success rate and the number of shortcuts added.

(c) Success rate for asking shortcuts' shortcuts.

Fig. 4. The potential of interest-based shortcuts.

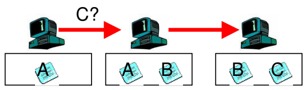


Fig. 5. Discovering new shortcuts through existing shortcuts.

performance gains.

VI. Understanding Interest-Based Locality

In this section, we seek to get a better understanding of the factors that contribute to the degree of interest-based locality observed in our workloads. We target our search to answer three questions in the context of the Web workloads:

- What do interest-based structures look like?
- Is interest-based locality capturing relationships between embedded objects that belong on the same Web page, or relationships across Web pages?
- Is interest-based locality capturing locality in accesses to the same publisher (Web server), or across publishers?

We would like to perform similar analyses on the CMU-Kazaa and CMU-Gnutella workloads to answer questions such as whether interests capture relationships between songs from the same artist, or songs from the same genre. However, it is nearly impossible to extract such information out of anonymized URLs and file names. Therefore, we focus our analysis on the Web workloads.

A. Properties of Interest-Based Structures

In this section, we treat the shortcut structure as a directed graph, and analyze its properties. Vertices in the graph are peers, and edges represent shortcut relationships. For example, if peer P_1 uses peer P_2 as a shortcut, a directed edge from P_1 to P_2 is created in the graph.

We look at two snapshots of the graph taken at 10 minutes and 1 hour into the simulation for segment 2 of the Boeing trace. Due to space limitations, we summarize our key findings below. First, when viewed as an undirected graph, we find that there are a large number of connected components after the first 10 minutes. Each connected component has only a few peers. At an hour into the trace, there are a few connected components, each composed of several hundred peers. In addition, each connected component is very well connected. Specifically, the graph has the highly-clustered characteristics of "small world" networks [18] with a small minimum distance between any two nodes. These characteristics are different than random graphs which are not clustered. The *clustering coefficient* of a vertex v is defined to be the fraction of edges that exist between its k_v neighbors over the possible total number of edges, $k_v * (k_v - 1)$. The clustering coefficient is 0.6 and 0.7 at 10 minutes and an hour into the trace. Clusters in the shortcut graph correspond to clusters of interests. Clusters have several implications for content location. First, if peers are looking for content within their usual areas of interest, it is likely that they will find it through the shortcut structure. However, if a peer wants to find content that does not lie within its usual areas of interest, then shortcuts are not useful and peers need to be able to escape from their current interest clusters. Retaining Gnutella, which has *random* connectivity, as an escape route is useful for such purposes.

Another interesting observation is that the clustering coefficient for the Web graph [19] which represents the HTML link structure in Web documents is much lower than those observed for the shortcut graph which represents Web accesses. We believe that interest-based locality is capturing relationships that are different from following links through Web pages. The Web access graph has interesting properties that are different from previously studied Web graphs. We also looked at clustering the document graph, where vertices in the graph are documents, and edges represent documents that are accessed by the same peer. For example, if a peer downloads files A and B, an edge is created between the two files. Edges are weighted by the number of distinct peers who also download those files. Running off-the-shelf graph partitioning algorithms [20] on the graph produces partitions of files. Future work includes comparing these partitions to the clusters created by shortcuts, and studying properties of the Web access graph.

B. Web Pages or Web Objects?

In this section, we explore the effect of the structure of Web pages on interest-based locality. Each Web page consists of multiple embedded objects. When a client requests a Web page, it usually gets several objects back from the server in parallel. In our evaluation in the previous sections, we assumed that Web content sharing systems share this same fundamental characteristic. That is, each object is a component of a larger entity (Web page). However, if objects in peer-to-peer systems have the same granularity as Web pages, would interest-based

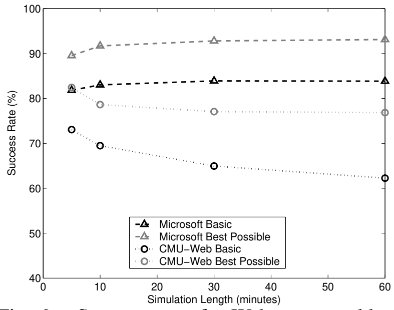


Fig. 6. Success rates for Web page workload.

locality still be useful?

In order to answer this question, we adapt the traces to approximate requests at the granularity of Web pages. We filter out all dynamic content using the methodology described in Section III-C. We then use requests for documents that have the text/html MIME type from the Microsoft and CMU-Web traces for the simulations. The Boeing trace did not contain content type information and were not used in these experiments. Due to incomplete information, our methodology has several limitations. For example, we do not explore the effect of multiple frames on one Web page. We count each frame as one distinct Web page. In addition, when examining only text/html requests, the number of participating peers drops by 30% because not all peers requested text/html (static) content.

We run the same set of simulations on the Web page workload as those discussed in Sections IV. We present results for one metric: success rates. Results for other metrics have the same trend as the results reported earlier. The black lines in Figure 6 depict the average success rate for the Web page workloads when using the basic algorithm of adding one shortcut at a time. The success rates are 85% and 63% at the end of the one-hour period for the Microsoft and the CMU-Web workloads. This is a performance drop of 5% and 15% compared to the success rates of locating Web objects in Section IV. We suspect that the performance difference is caused by locality between Web objects on the same page. We verify our hypothesis by looking at the best possible success rate when peers are allowed to add all possible peers returned from Gnutella's flooding as shortcuts, depicted as the gray lines in Figure 6. The performance drop of 5% to 15% is consistent with the basic algorithm. Improving the shortcut discovery algorithm as explored in Section V should also improve the success rate performance for Web pages as well.

In summary, the performance gains observed in Section IV are contributed to by the structure of Web pages (multiple objects per Web page) and the interest-based relationship between Web pages. Interest-based shortcuts are capable of exploiting both properties to further improve performance.

C. Objects From Different Publishers?

Another factor that may contribute to interest-based locality is the locality in accesses to content from the same Web site (or publisher). In this section, we ask does content from the same publisher necessarily belong to the same interest group? In addition, does interest-based locality capture relationships across publishers?

TABLE III

THE EFFECT OF SAME-PUBLISHER SHORTCUTS.

Trace	Successful		Unsuccessful	
	Same	Diff	Same	Diff
Boeing	49%	31%	7%	13%
Microsoft	32%	54%	2%	12%
Microsoft-HTML	10%	73%	2%	15%

Intuitively, content from the same publisher should belong to the same interest group. However, it is not clear whether this granularity of interests is powerful enough to help with content location. Specifically, content popularity at a publisher follows a Zipf-like distribution [21]. A small number of pages are highly popular, whereas a large number pages are rarely accessed. Thus, each peer may access very different sets of pages. To gain a better understanding, we analyze the results reported in Section IV to answer the following questions:

- When shortcuts succeed, is the content from the same publisher?
- When shortcuts do not succeed, do peers have a shortcut for that publisher?

We use the hostname part (or IP address) of the URL as the publisher name. Due to anonymization, we can only match servers that have the same exact name. For example, www.cnn.com and www2.cnn.com are treated as two different publishers although they are semantically the same. As a result, it is likely that we underreport the count for content belonging to the same publisher.

Table III lists the frequency at which content is found successfully through shortcuts, along with whether the shortcut that was used was for the same publisher. Same-publisher shortcuts for a request are shortcuts that were originally created as a result of accessing content from the same publisher as the publisher for the current request. For the Boeing trace, we found that on average, same-publisher shortcuts are successful at locating content 49% of the time across 8 trace segments. Different-publisher shortcuts are successful 31% of time. We find that interest-based locality does capture interests across multiple publishers. In addition, we find that 7% of the time same-publisher shortcuts are not sufficient at locating content. The results for the Microsoft trace are similar, but less content can be found through same-publisher shortcuts. We believe the majority of these same-publisher shortcut successes are for Web objects that belong to the same Web page because same-publisher shortcuts are much less effective at locating content for the HTML-only traces. To summarize, interest-based locality is different than same-publisher locality. Interest-based shortcuts can exploit relationships that span across multiple publishers to locate content.

We find that shortcuts are effective at capturing interest-based locality at many levels of granularity ranging from locality in accessing objects on the same Web page, accessing Web pages from the same publisher, and accessing Web pages that span across publishers. In addition, interest-based structures have different properties than the HTML link structures in

TABLE IV

LOAD AT EACH PEER IN QUERY PACKETS/SECOND.

Trace	Protocol	5	6	7	8
Boeing	Chord	0.0352	0.0414	0.0473	0.0397
	Chord w/ Shortcuts	0.0113	0.0132	0.0165	0.0145
Microsoft	Chord	0.0677	0.0985	0.1179	0.1462
	Chord w/ Shortcuts	0.0228	0.0334	0.0365	0.0471

Web documents.

VII. SENSITIVITY TO UNDERLYING CONTENT LOCATION MECHANISM

In Section II, we claimed that shortcuts are modular components that can be used with any underlying content location protocol. In this section, we explore the performance of shortcuts with Chord [5], a DHT-based protocol.

Chord provides efficient and scalable distributed lookups that resolves content IDs to locations in $1/2 \log N$ overlay hops, where N is the number of participating peers. To facilitate lookups, each node maintains $O(\log N)$ state about peers in the system.

For our evaluation, we assume that Chord provides the following simple interface: when a peer sends a query for a piece of content, Chord returns a list of IP addresses of all peers that store a copy of that content. In addition, we assume that content placement is based on peers' individual interests. That is, peers only store content that they have requested and downloaded. Following the architecture presented in Section II, we implement shortcuts as a separate performance-enhancement layer. Queries are resolved through Chord only when peers do not have any shortcuts to use, or shortcuts are unsuccessful at locating content.

The performance metrics are the same as Gnutella's. Because we are using the same trace and the same shortcuts algorithm, the results for success rate and state overhead remain the same as those reported in Section IV. When shortcuts are successful, the path length for locating content is also the same at 1.5 hops. By comparison, Chord locates content within 7 hops. The key differences between using Chord and Gnutella as the underlying content location protocol are load and query scope. Table IV lists the number of queries observed at each peer in packet/second for Chord and Chord with shortcuts. Due to space limitations, we only present results for the last 4 segments of the Boeing and Microsoft workloads. Chord is already efficient at limiting the load compared to Gnutella (see Table II). For example, for segment 5 of the Microsoft trace, Chord limits the number of queries to 0.07 packets/second compared to Gnutella's 479 packets/second. Shortcuts, when used with Chord, reduce the load even more down to 0.02 packets/second. Shortcuts reduce the load by a factor of 2-4 across all traces.

Query scope, which is the fraction of peers in the system involved in a query, for Chord is analogous to query path length. The scope is $7/N$ peers for all trace segments. Shortcuts, when successful, reduce the query scope down to $1.5/N$. However, when shortcuts are not successful, query scope is the sum of the number of all shortcuts that were tried (shortcut list size) and the number of Chord hops over the total number of peers in the system. The scope for unsuccessful shortcuts is increased to approximately $12/N$ hops. Fortunately, peers do not have to pay the penalty for unsuccessful shortcuts very often. To summarize, interest-based shortcuts can improve the performance of Chord, a DHT-based protocol.

VIII. CONCLUSION AND DISCUSSION

In this paper, we propose a technique to create shortcuts in content location overlays. We believe that this is a promising approach to introducing performance enhancements to overlay construction algorithms. In our architecture, shortcuts are modular building blocks that are constructed on top of generic large-scale overlays. Because shortcuts are designed to exploit locality, they can significantly improve performance. Furthermore, layering enables higher performance without degrading the scalability or the correctness of the underlying overlay construction algorithm.

Interest-based locality is a powerful principle for content distribution applications. We show that interest-based locality is present in Web content sharing and two popular peer-to-peer file-sharing applications. Applications can construct shortcuts to exploit their locality characteristics and thereby improve performance.

In our study, we find that interacting with a small group of peers, often smaller than ten, is sufficient for achieving high hit rates. Our results differ from previous Web caching studies [22] that report that hit rates only start to saturate with population sizes of over thousands of clients. The difference is that in our approach, peers are grouped based on interests, whereas in Web caching, all clients are grouped together. Cooperation with small groups of peers who share interests provides the same benefits as cooperation with a large group of clients with random interests.

In addition to improving content location performance, interest-based shortcuts can be used as a primitive for a rich class of higher-level services. For instance, keyword or string matching searches for content and performance-based content retrieval are two examples of such services. Distributed hash tables [3] [4] [5] [6] do not support keyword searches. Interest-based shortcuts can be used to implement searches on top of those schemes in the following way. Peers forward searches along shortcuts. Then, each peer that receives a search performs a keyword match with the content it stores locally. There is a likely chance that content will be found through shortcuts because of interest-based locality.

Performance-based content retrieval can also be implemented using interest-based shortcuts. The advantage of such a service is that content can be retrieved from the peer with the best performance. Most peer-to-peer systems assume short-lived interaction on the order of single requests. However, shortcuts provide an opportunity for a longer-term relationship between peers. Given this relationship, peers can afford to carefully test out shortcuts and select the best ones to use based on content retrieval performance. In addition, the amount of state peers need to allocate for interest-based shortcuts is small and bounded. Therefore, peers can store performance history for all of their shortcuts. Peers can even actively probe shortcuts for available bandwidth if needed.

One potential concern about interest-based locality is whether exploiting such relationships infringes on privacy any more so than underlying content location mechanisms. We argue that it does not. First, peers do not gain any more information than they have already obtained from using the underlying content location mechanism. Interest-based shortcuts only allow such information to be used intelligently to improve performance.

IX. Related Work

Improvements to Gnutella's flooding mechanism have been studied along two dimensions. First, query caching [23] exploits the Zipf-like distribution of popularity of content to reduce flooding. Second, approaches based on expanding ring searches, which are designed to limit the scope of queries, and random walks [24], where each peer forwards a query message to a randomly chosen neighbor, in place of flooding are also promising at improving Gnutella's scalability. Such approaches are effective at finding popular content, whereas interest-based shortcuts can find both popular and unpopular content. These proposed improvements to Gnutella can be used along with interest-based shortcuts to further improve the performance of Gnutella. More recently, associative overlays [25], also proposed to improve Gnutella's performance, are based on the same principles as interest-based locality.

Other peer-to-peer applications, such as Kazaa, can also use interest-based locality to improve performance. Although the specifics of Kazaa's protocol is not publicly known, well-connected peers are selected as "supernodes" which serve to index content located at other peers. When locating content, a normal peer contacts its supernode who, in turn, may contact other supernodes. Shortcuts can be used in such environments for efficient query routing between supernodes, or to reduce load at supernodes.

There are several proposals for systems that distribute Web content using peer-to-peer technology. YouServ [12] is a peer-to-peer Web hosting service. Their design stresses ease in publishing. Content location is provided through dynamic DNS lookups on URLs using a centralized architecture. CoopNet [11] proposes to use peer-to-peer content distribution to complement client-server systems during flash crowds. Clients that have downloaded content can turn around and serve the content to other clients. Content location is initiated through the original Web server. BitTorrent [13] delivers Web content using peer-to-peer technology and is designed to integrate seamlessly with existing Web browsers. Content location is provided by asking a well-known server. In Squirrel [10], peers in a local area network share content to emulate a Web proxy. Squirrel uses Pastry [4], a DHT-based algorithm, for content location. Interest-based shortcuts can also be employed in these systems to either avoid centralized content location bottlenecks in YouServ, CoopNet, and BitTorrent, or to improve the performance of Pastry in Squirrel.

Interest-based locality is related to ideas in collaborative filtering [26] which suggests that people make choices based on others' opinions. We are building a decentralized peer recommendation system, where we use heuristics to identify and "recommend" peers who share similar interests.

Acknowledgements

We wish to thank Venkat Padmanabhan for the Microsoft Corporate proxy traces and Matei Ripeanu for the Gnutella connectivity graphs. We also thank Frank Kietzke and other members of CMU Computing Services who helped us tremendously to debug and run our trace collection software. Dan Blandford and Guy Blelloch helped experiment with graph partitioning algorithms on our graphs. Finally, we thank Yang-hua Chu, Andy Myers, and the anonymous reviewers for the constructive criticism of earlier drafts of this work.

References

[1] "Akamai Technologies, Inc." http://www.akamai.com.

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

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

[4] A. Rowstron and P. Druschel, "Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems," in *IFIP/ACM International Conference on Distributed Systems Platforms (Middleware)*, 2001.

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

[6] B. Zhao, J. Kubiatowicz, and A. Joseph, "Tapestry: An infrastructure for wide-area fault-tolerant location and routing," *U. C. Berkeley Technical Report UCB//CSD-01-1141*.

[7] C. Plaxton, R. Rajaraman, and A. W. Richa, "Accessing nearby copies of replicated objects in a distributed environment," in *Proceedings of the 9th Annual ACM Symposium on Parallel Algorithms and Architectures*, 1997.

[8] S. Ratnasamy, S. Shenker, and I. Stoica, "Routing algorithms for DHTs: Some open questions," in *Proc. of International Peer-To-Peer Workshop*, 2002.

[9] "Kazaa," http://www.kazaa.com.

[10] S. Iyer, A. Rowstron, and P. Druschel, "Squirrel: A decentralized peer-to-peer Web cache," in *ACM Symposium on Principles of Distributed Computing, PODC*, 2002.

[11] V. Padmanabhan and K. Sripanidkulchai, "The case for cooperative networking," in *Proc. of International Peer-To-Peer Workshop*, 2002.

[12] R. B. Jr., A. Somani, D. Gruhl, and R. Agrawal, "Youserv: A Web hosting and content sharing tool for the masses," in *Proc. of International WWW Conference*, 2002.

[13] "Bittorrent," Available at http://bitconjurer.org/BitTorrent.

[14] J. Meadows, "Boeing proxy logs," Available at ftp://researchsmp2.cc.vt.edu/pub/boeing/, March 1999.

[15] V. Jacobson, C. Leres, and S. McCanne, "tcpdump," Available at http://www.tcpdump.org/.

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

[17] M. Ripeanu, I. Foster, and A. Iamnitchi, "Mapping the Gnutella network: Properties of large-scale peer-to-peer systems and implications for system design," *IEEE Internet Computing Journal*, vol. 6, no. 1, 2002.

[18] D. Watts and S. Strogatz, "Collective dynamics of 'smallworld ' networks," Nature 393,440–442 (1998).

[19] L. Adamic, "The small world web," in *Proc. of 3rd European Conf. Research and Advanced Technology for Digital Libraries, ECDL*, 1999.

[20] D. Blandford and G. Blelloch, "Index compression through document reordering," in *Data Compression Conference (DCC)*, 2002.

[21] V. Almeida, A. Bestavros, M. Crovella, and A. de Oliveira, "Characterizing reference locality in the WWW," in *Proc. of the IEEE Conference on Parallel and Distributed Information Systems (PDIS)*, 1996.

[22] A. Wolman, G. Voelker, N. Sharma, N. Cardwell, A. Karlin, and H. Levy, "On the scale and performance of cooperative web proxy caching," in *Proc. of ACM SOSP*, 1999.

[23] K. Sripanidkulchai, "The popularity of Gnutella queries and its implications on scalability," http://www.cs.cmu.edu/ kunwadee/research/p2p/gnutella.html, February 2001.

[24] Q. Lv, P. Cao, K. Li, and S. Shenker, "Replication strategies in unstructured peer-to-peer networks," in *Proc. of ACM International Conference on Supercomputing(ICS)*, 2002.

[25] E. Cohen, A. Fiat, and H. Kaplan, "A case for associative peer-to-peer overlays," in *Proc. of Workshop on Hot Topics in Networks*, 2002.

[26] P. Resnick, N. Iacovou, M. Suchak, P. Bergstorm, and J. Riedl, "GroupLens: An Open Architecture for Collaborative Filtering of Netnews," in *Proc. of ACM Conference on Computer Supported Cooperative Work*, 1994.