REDUCING BANDWIDTH UTILIZATION IN PEER-TO-PEER NETWORKS

Dushyant Bansal and Paul A.S. Ward
{dbansal,pasward}@ccng.uwaterloo.ca

E&CE Technical Report Number: 2002-17
Shoshin Distributed Systems Group
Department of Electrical and Computer Engineering
University of Waterloo

December 2002

ABSTRACT

In this document we study the problem of reducing bandwidth utilization in file sharing peer-to-peer (P2P) networks. First we describe the various aspects of peer-to-peer computing that make it attractive over client-server computing. Then, we examine the environmental characteristics of a P2P system in deployment. We explore some orthogonal factors that need to be considered when constructing a bandwidth-efficient P2P framework. These include the distribution of control in the system, routing, and caching and replication mechanisms. We follow this with a description of a number of P2P architectures, including indexing-based schemes, flooding-based schemes, hashing-based schemes, and a publish/subscribe-based scheme, and their advantages and disadvantages are qualitatively examined. Finally, we put forth a number of proposals on the topics of multidimensional P2P routing, wireless P2P networks, web services and P2P, and what Internet Service Providers (ISP) can do to reduce bandwidth consumption by poor-quality P2P file sharing systems.

CONTENTS

1 INTRODUCTION

Peer-to-peer (P2P) computing is a recently popular model of network computing. The distinguishing characteristic of P2P systems is logical node symmetry. That is, every node can function as a client, as a server, or both. Allowing peer nodes to share unused disk storage, CPU cycles and network resources, P2P has found applications in various areas, such as file distribution, distributed applications, online collaboration, home networking, online games, and instant messaging. Nodes connect to the P2P network on an ad hoc and dynamic basis, use services that are available at the time, offer any services if they so desire, and then leave. Although a central location in client-server computing allows for faster resource look up, data consistency and control, the server also becomes a single point of failure and bandwidth bottleneck. The distributed approach of P2P on the other hand means that there is no single point of failure and more work can be accomplished by aggregating the unused resources of all participating nodes.

Despite these advantages, this distributed approach and intermittent presence of nodes on a P2P network make it difficult to locate and use resources on the network efficiently. The service resources that peers make available (*e.g.* files in a file-sharing service) are not mirrored in a deterministic manner, so nodes all over the world may end up communicating with each other unnecessarily, increasing the consumption of network resources. The focus of this document is dealing with these problems in the context of sharing static data. First, the environmental conditions of a deployed P2P network are examined. Then the various factors that need to be considered when architecting a resource-efficient P2P system are analyzed. Various architectures are examined qualitatively. The techniques that these systems use to improve network resource utilization are assessed. Finally, proposals are made on multidimensional routing, P2P in a wireless environment, P2P in conjunction with Web services and what ISP's can do to save bandwidth consumed by P2P file-sharing applications.

2 ENVIRONMENTAL CHARACTERISTICS

We first describe the characteristics of the Napster and Gnutella P2P protocols, as presented by Saroiu, Gummadi, and Gribble [20]. They reveal a significant amount of peer heterogeneity in terms of bandwidth, latency, availability, and the degree of sharing, which vary over three to five orders of magnitude. Their study also observed characteristics of the Gnutella P2P network topology and found that peers tend to deliberately misreport information.

2.1 HETEROGENEITY

A significant fraction of the participating hosts tend to behave more client-like or server-like. Server-like hosts have low latency and high availability, good upload bandwidth, and a large number of files available for sharing. Client-like hosts, also known as "free-riders," exhibit high downloads and low availability, poor upload bandwidth, and a small number of files available for sharing.

2.1.1 BANDWIDTH

On average, peers have higher download than upload bandwidth, because of the asymmetric nature of their Internet connection links. Thus, in theory, the download capacity of the system exceeds its upload capacity. Only about a tenth of the Gnutella users are connected with modems (of 64 kbps or less), possibly because the flooding-based search mechanism of the Gnutella protocol places too high of a burden on low-bandwidth connections.

2.1.2 LATENCY

There were two significant points of note with regard to latency. First, the closest (in terms of latency) 20% of the peers are four times closer than the furthest 20%. Second, a peer interacts with three large classes of peers: (1) peers on the same part of the continent, (2) peers on the opposite part of a continent and (3) trans-oceanic peers. Within these classes, bandwidth varies significantly.

2.1.3 AVAILABILITY

Uptime was measured as the percentage of time that the peer is available and responding to traffic. The IP-level uptime is when peers are connected to the Internet but may or may not be connected to the P2P system. The application-level uptime

is when peers are connected to the P2P system as well. Only a fifth of the peers have an IP-level uptime of 93% or more. The best one-fifth of the peers in Gnutella have a P2P uptime of only 45% or more, while in Napster the best one fifth were connected to the Napster network 83% or more of the time. Most sessions are quite short. The median session duration is approximately an hour, which is how long it typically takes to download a small number of music files from the service.

2.1.4 THE DEGREE OF SHARING

Less than one-tenth of the peers share more files than the rest combined, showing that Gnutella contains a large percentage of free-riders. There are more zero-download high-speed peers than zero-download low-speed peers, possibly because higher bandwidth peers spend less time downloading due to their higher connection speeds. Conversely, there are more zero-upload peers for modems than for cable modems, possibly because modem users cannot (or will not) spare their limited bandwidth for uploads and because users prefer high-bandwidth peers to download content from. There is a strong positive correlation between the numbers of files shared and the number of shared megabytes of data. The average size of a shared file is 3.7MB, corresponding to the average size of an audio file encoded in the Moving Picture Experts Group 3 (MPEG3) format.

2.2 TOPOLOGY

In Gnutella, peers preferentially form connections with highly available and high out-degree nodes in the overlay, leading to a vertex connectivity power-law distribution with an index less than 3. Such a network is very robust in the face of random node breakdowns. For example, the overlay fragments only when more than 60% of the nodes breakdown randomly. However, an orchestrated attack against the highest degree nodes in the overlay “shatters” the overlay into a large number of disconnected components. Additionally, firewalls and NAT’s (Network Access Translation) create partitioned domains, adding to the complexity of network topologies [21].

2.3 MISREPORTING

The extent to which peers deliberately misreport their bandwidths was used as a first-order approximation to quantify a peer’s willingness to cooperate. A larger fraction of users with low bandwidth Internet connections tend to misreport their Internet connection speeds than than do users with high bandwidth Internet connections. The likely motive for misreporting is to discourage download requests

from other peers, which would otherwise consume valuable network resources. This is particularly important to low bandwidth users, who have smaller amounts of network resources to start off with. However, it should be noted that an alternate explanation is that users are concerned about the legality of the files that they are sharing [1]. Future systems may be able to directly measure or verify reported information.

3 CONSERVING BANDWIDTH

The distributed nature of P2P-style content sharing means that the closest peer-node sharing a file may not be the closest node in the network that offers that file. Indeed, the closest peer-node may be several transit links away. Downloading files from such sources would unnecessarily consume more bandwidth, using expensive links, such as transcontinental links, driving up transit charges on the carrier side. Besides the actual file download, protocol chatter (messaging required to maintain the P2P network) consumes a large percentage of its total traffic. One study has shown that in a flooding-based query protocol, querying constitutes over half the message traffic [15].

According to network measurements performed in a study between April 2000 and November 2001, the percentage of network bandwidth consumed by Napster traffic increased from 23% to 30% while the percentage consumed by web-related traffic actually decreased from 20% to 19% [3]. By 2002 it was observed that file-sharing traffic consumed up to 60% of an ISPs bandwidth, with a non-trivial fraction of that being consumed by protocol chatter [19].

Mechanisms are required to reduce bandwidth consumption of P2P content sharing. To save network resources and charges for both the user and the service providers, the files could be replicated at various sites. This document describes ways to pick these sites. Caching the information on the user's local hard disk would also enable the user to consume a smaller portion of the network bandwidth and can produce significant improvements in traffic reduction [12].

To reduce the traffic generated by querying, one option is to use better search algorithms. Rather than querying machines individually for a desired resource, different mechanisms should be used to organize content and route messages. The frequency and locality of queries can also be exploited by caching query results on the user's machine to reduce the query overhead, reducing bandwidth consumption [3]. Cached query results can also be used to answer other users' queries. Due to the intermittent nature of node connectivity in P2P networks, query results should stay in the cache for a limited time only or the cache would need to be invalidated somehow.

4 ARCHITECTURAL CONSIDERATIONS

As with most distributed systems, it has been found that using caching, replication, and directories boosts the performance of P2P systems greatly by cutting down on search time and bandwidth consumption [9]. The specific design choices by which these mechanisms are implemented varies in the different P2P architectures. We therefore now consider these choices.

4.1 LEVEL OF CONTROL

We can identify some ways of allocating control to the nodes [13]. These are centralized, decentralized and structured, and decentralized and unstructured architectures.

4.1.1 CENTRALIZED

In this model, a central directory is maintained that is constantly updated. Nodes in the P2P network issue queries to the central directory server to find which other nodes hold the desired files, and the files are subsequently downloaded directly from those peers. This boils down to the client-server model as far as searching is concerned. Although this model offers a comprehensive search, a fast directory update, and a minimal total number of messages exchanged, this model also suffers from scalability issues and has a single point of failure, unless the directory is replicated a sufficient number of times to handle the required service load. An example of such a centralized approach in the P2P context is Napster [13]. Other examples, not usually thought of in a P2P context, are the various web search engines, such as Google.

4.1.2 DECENTRALIZED AND STRUCTURED

These systems have no central directory server, but the P2P overlay topology is tightly controlled. Files are placed at specific locations to make subsequent queries easier to satisfy. Examples include Pastry/PAST [18, 17] and Chord [4].

4.1.3 DECENTRALIZED AND UNSTRUCTURED

In these systems, there is neither a centralized directory nor any precise control over the network topology or file placement. Nodes join the P2P network and files are placed without knowledge of the underlying network topology. An example would

be Gnutella [13]. Not dependent on a central indexing node, there is no single point of failure, which results in increased fault tolerance and robustness. A node only stores the addresses of its immediate P2P neighbours without concern for the P2P-network topology beyond its vicinity. This avoids the problem of convergence due to slow propagation of routing information and the node can deal with the dynamic nature of peer-to-peer networks more easily [3].

4.2 Caching and Replication

To improve bandwidth utilization and access latency in P2P file sharing, data should be brought closer to the nodes that request it [9]. This is typically accomplished by caching and/or replicating data at various points in the network. Caching query results may also avoid repeating the same work multiple times. Both caching and replication act as load-balancing mechanisms. We briefly distinguish between these two mechanisms.

4.2.1 Caching

A cache is a store of files, or portions of files, that gets populated with a copy of the downloaded content on an on-demand basis and is expungeable. The purpose of caching is to allow a client (possible a different client from that which originally downloaded the file) to access the same information faster the next time. This inadvertently also results in reducing the load on the server. In cooperative caches, nodes make use of data that may have been cached by other nodes for their use. It is coincidental that they find information that another node had cached. A cache may be expunged to make room for more currently in-demand data objects.

4.2.2 Replication

Replication refers to the process of a node deliberately placing information in multiple locations. In the case of replication, the goal is to reduce the load on the server by providing mirror sites to acquire the same content. The client may experience a lower access time as a result, primarily by virtue of the lower load on the server, rather than due to proximity. In the case of replication, it is not coincidental that information is found in multiple places, since it was deliberately placed there. A replica-store is generally not expunged to make room for more currently in-demand data objects, although objects that go out of demand may be evicted. Replication can be performed using different strategies [13]:

Owner replication: When a search is successful, the object is stored at the requester node only, which is the same thing as caching. Owner replication is used in systems such as Gnutella.

Pro-Active replication: A node may choose to pro-actively replicate certain files on other nodes. These nodes may be chosen in a deterministic manner such as based on node identifier or in a random manner. In random replication, either a random number of neighbouring nodes or a random number of nodes along the download path are used to store replicas of the desired files.

Path replication: When a search succeeds, the object is stored at all nodes along the path from the requester node to the provider node. Path replication is used in systems such as Freenet.

4.3 ROUTING

Once a data item is created on the P2P network, another peer can locate it either by a process of discovery, as would happen in Gnutella, or in a deterministic way using a directory. The process of discovery, or message routing, can take place either in a content-agnostic way or in a content-based way [2]. Content-agnostic routing does not take the content of the file into account and content-based routing does. A directory works by having each entry map the hash key of a data item to the address of the node storing/administering the data item or its replica. A distributed directory can help speed up the process of discovery by providing short cuts to the node storing a copy of the data item.

4.3.1 CONTENT-AGNOSTIC ROUTING

In this approach the content of the file is not considered when routing a query or determining a replication node for the file. The mechanisms used in content-agnostic routing include querying (e.g. Gnutella), indexing, (e.g. FastTrack), and publish/subscribe (e.g. JXTA Search).

Querying means that every search a peer attempts results in the peer querying other peers for the requested file. The manner in which this is done varies. For example, Gnutella does it by broadcasting the request to all P2P neighbours, which in turn do the same, and so on until the hop-limit is reached. This results in a flood of messages, as described in Section 5.2.1. Other algorithms that can be used to propagate the query include the expanding ring protocol, which uses successive floods with increasing Time-To-Live (TTL) in terms of number of hops until the object is found [13]. The random-walks algorithm forwards a received query to a random number of neighbouring nodes simultaneously. To avoid loops, it remembers which nodes it forwarded the query to previously and does not pick them again [13].

Indexing is a way of maintaining a database of various attributes of files, such as filename and author, and some identification of nodes that may have that file.

Napster followed a centralized approach and FastTrack follows a decentralized approach.

In the centralized approach, a single logical indexing server keeps track of all available files. Such centralized indexing may have scalability and reliability issues since the indexing node becomes the bandwidth bottleneck. While it should be noted that centralized systems can be architected to handle the load and deal with the reliability issues (*e.g.* Google handles over 150 million queries daily without appreciable difficulty [7]), P2P systems that are based on the voluntary offering of resources are unlikely to be able to replicate that performance of such carefully architected systems. Further, for social reasons the centralized indexed may not be considered acceptable (*e.g.* possible search monitoring, vulnerability to legal shutdown of the central index, *etc.*).

In the decentralized approach, various nodes maintain an index of files hosted by a set of peers in their vicinity. These index nodes respond to queries sent by their respective peer set, either by locating other peers in the same set that are willing to share the file, or by redirecting the queries to other indexing nodes.

In the publish/subscribe paradigm, the producer nodes publish filters for their files and the consumer nodes subscribe to these filters to be notified when the desired files are available. These roles can be reversed. In the centralized mechanism, a group of coordinating nodes maintains the producer and consumer filters and is responsible for matching up subscriptions with publications. In the decentralized approach, the consumers propagate their filters to the producer nodes, which compare the consumer filters with their own filters, and respond if they can fulfill the request and propagate the filter [8]. A middle approach is to dynamically select coordinating nodes, as is done in JXTA search [2]. To improve routing efficiency, only the most general filters should be maintained in all cases [2].

4.3.2 CONTENT-BASED ROUTING

Content-based routing is the alternative scheme. It takes some or all of the file content into account. One way of accounting for file content is to hash that content to map the file to a certain key. This is the approach taken by Content-Addressable Network (CAN). Clearly a client cannot use this content-hashing technique when doing a search, as it cannot know the content. However, it is useful in locating other copies of the same file. In CAN, each node itself has an identifier and is responsible for a certain range of keys (zone) and the corresponding files. A request for a file is routed toward the destination node by forwarding the request to a node with an increasingly similar identifier. We present more details on the CAN network in Section 5.3.1.

The Pastry routing overlay assigns random identifiers to nodes [17] and the

PAST storage utility, built on top of Pastry, hashes the file's name and owner as a file identifier [18]. In this Pastry-PAST combination, nodes are responsible for files with identifiers within a certain range of their own. Routing is performed by forwarding the request toward the node with a numerical identifier closest to the file identifier.

Other content-based routing mechanisms include routing based on the metadata of a file, such as name and author, without hashing, however. By organizing the metadata in a hierarchical fashion and aggregating files by metadata, the authors hope to create an infrastructure that would require much fewer requests to satisfy any query [6].

5 EXISTING ARCHITECTURES

We now present a qualitative examination of various architectures that have been designed for P2P file sharing. We point out disadvantages of these approaches, where applicable.

5.1 INDEX-BASED

Files can be indexed for easier query routing and file location. We evaluate three different index-based P2P schemes: Napster, FastTrack and Content-Based Aggregation.

5.1.1 NAPSTER

As explained by Brain [3], every time a user logged into the Napster network, the desktop software updated a central indexing server with the files being shared on the user's machine. Users directed their queries to this central server, which would return the identity of the machines containing matching files. Thus the method of message routing was content-agnostic. The user directly connected to one of these machines and downloaded the desired file. This distributed the storage and bandwidth requirements throughout the Napster network, except for the index owner.

Documentation on Napster's caching and replication strategy are not available. Hence, it can only be speculated that Napster cached files on the user's local machine and that the central indexing server kept track of recent users who downloaded certain files as an implicit replication strategy.

5.1.1.1 DISADVANTAGES

This scheme has the obvious disadvantage of a centralized index, which formed a single point of failure and a bandwidth bottleneck. Shutting down the central indexing server in Napster brought down the entire network [3].

5.1.2 FASTTRACK

FastTrack follows a decentralized and structured approach, with message routing taking place in a content-agnostic way. In the FastTrack P2P network files are first downloaded from the peers with the highest bandwidth at the time, using nodes with lower ping times to increase download speed and reduce the load on ISP backbones [11]. A cluster-based approach is followed where each clustering node

("super-node") contains a partial index of the files available and their location. A query is directed to a super-node, which propagates the query to other super-nodes. Grokster, a client for FastTrack, extracts information about the content of a file when it is published. For audio files this would include data such as the bit rate and the genre. Such metadata is associated with the respective file and can be searched, which is better than enforcing a fixed directory structure on any node. Since FastTrack is a proprietary undisclosed protocol, its details on caching and replication are not available.

5.1.2.1 DISADVANTAGES

The clustering-based approach greatly reduces the number of messages required to execute a query. However, when a super-node exits the network, it could take some time and message traffic before the super-node's child nodes are re-distributed among the existing super-nodes, files are re-indexed, and a new super-node is elected to make up for the previous one. Maintaining a replica node for every super-node's index could help speed up the process.

5.1.3 CONTENT-BASED AGGREGATION

The content-based aggregation architecture uses a content-based distributed index to locate content without using broadcast-style techniques [6]. The aim is to reduce the number of nodes looked up to satisfy any query. This architecture associates a metadata file with each file, where the metadata corresponds to information such as genre in the case of multimedia files or author in the case of any file type. The extension of the file forms the root of the aggregation tree that determines what fields will be contained in the metadata for that file. As an example of classification, a music composition by Bach encoded in mp3 format would be classified as "mp3" as file type, "classic" for genre, "J.S. Bach" for composer, "organ" for medium and "Toccata and Fugue in D minor" for title. Aggregation points would exist for each level in this sample hierarchy. Thus, when a user searches for music of the aforementioned qualifications, his/her query would be routed from one aggregation point to another in a tree-like manner, without having to search the entire file-space. Any empty metadata fields are treated as wildcards and the query is forwarded to all aggregation points at that level. Hence, the more elaborate the metadata is in the search context, the more efficient the search becomes in terms of using content aggregation. The query percolates down the metadata hierarchy until it receives the IP address of one or more leaf nodes containing the required file. The hierarchy of the aggregation networks ensures a relatively small search overhead in the order of $O(n)$, where it is not stated what n refers to, but it may be the number of search fields [6]. The overall result is a decentralized and structured organization.

When a node wishes to the join the network, it contacts a "helper node," the IP address of which is obtained through external means, such as a website or magazine. The helper node returns the addresses of aggregation points it knows of. Using a series of measurements such as delay and latency, the joining node decides to join the group of a particular aggregation node.

To provide fault tolerance and load balancing, the index of every aggregation point is mirrored on other nodes called "replicants," which take over in the case of failure. If no replicas are present, any node in the group can take over based on node characteristics and a suitable election algorithm.

5.1.3.1 DISADVANTAGES

A query would only be routable in a tree-like manner if the query matches the directory hierarchy. This is a matter of schema specification and it would be required to enforce it on both the user's end as well as the whole organization of files.

Using metadata to locate files may require the exact spelling of the various attribute values. Although approximate matching can mitigate this problem, there still remains the issue of having the same file named differently or having different metadata due to a different classification.

5.2 FLOODING-BASED

Another approach to looking for files is the brute-force approach. In the P2P context, this refers to broadcasting a query to all surrounding nodes, which in turn do the same thing until an instance of the file is found. The Gnutella protocol follows this strategy to execute queries.

5.2.1 GNUTELLA

As with the Napster network, a user would log into the Gnutella network using some Gnutella-compatible desktop software [3]. However, rather than referring to a central indexing server, the software uses a flooding-based application-level broadcast to satisfy queries. A user's query would be executed by first contacting known Gnutella machines with the query. If the contacted machines have, or known of, the file, they send back the file name and the IP addresses of the nodes where the file can be found. They also propagate the query to the machines they are connected to and the process repeats. By virtue of a hop-limit, a request propagates only for a limited number of steps, thus limiting the message explosion. The result is a decentralized and unstructured organization with content-agnostic routing. No caching or replication is performed.

5.2.1.1 DISADVANTAGES

Gnutella suffers from a number of serious drawbacks.

1. It does not perform any caching or replication optimizations on content or query results. It only saves the downloaded file on the local hard drive, which may or may not be added to the pool of shared files based on the user's request.

2. The exponential growth in the number of query and reply messages consumes a large bandwidth and time overhead, which prevents this flooding approach from being scalable.

3. The hop-limit limits how far the query propagates, which means that search may not result in success even if the file does exist on some node. Studies find that only half of the peers are reached by Gnutella's flooding mechanism [3].

4. The Gnutella network topology does not match the underlying Internet topology very well, leading to inefficiencies in bandwidth utilization [3].

5. When a node leaves the network and if it was the only neighbour of another node, this would lead to some degree of network fragmentation for the second node. A re-broadcast to discover other neighbours or being contacted by another node would be the only ways to help the second node rejoin the network.

Apparently in spite of these disadvantages, Gnutella is quite popular, presumably due to the nature of the files being shared, and the difficulty implied by the architecture in shutting down such a P2P network.

5.3 HASHING-BASED

A better approach to distributing information about available files is by using hashing techniques. Three architectures that exploit this approach are CAN, Pastry/PAST and Chord.

5.3.1 CONTENT-ADDRESSABLE NETWORK

Content-Addressable Network (CAN) is a scalable indexing mechanism to manage data items in a P2P network. However, as opposed to the metadata approach, CAN does not impose any form of naming structure to achieve scalability [16]. Instead, files are placed in the P2P network in a more structured but still decentralized

way. A hash of the contents of the file is used to determine which node in the network would take on the responsibility of serving that file, making the CAN routing mechanism a content-based one. Many files will map to any given node. Each node's set of files is called a zone. This results in a distributed hash table with the whole file space divided up into zones. It is assumed that when a node comes online, it publishes its files to the P2P network. Publication means adding an entry for its files in the hash-tables of the node with the closest hash identifier. The CAN system performs caching and replication, as described in the following paragraphs.

By virtue of the keys that every node is responsible for in its zone, the entire node space is divided up into a dynamically organized d-dimensional virtual coordinate space [16]. When sending a message, the destination coordinates are specified in the message. The message is then routed incrementally toward the destination node by forwarding the request to the node with common edges along the most number of dimensions, as is done in hypercube routing. With this concept of dimensionality, the CAN scheme is able to strike a balance between per-state node in terms of number of neighbours ($O(d)$ for a d-dimensional space) and routing path length in terms of application level hops ($O(dn1/d)$ for d dimensions and n nodes). However, this does not address the possibility of a large latency per hop, since adjacent nodes may still be many miles and many hops apart. A reduction in latency is achieved by varying certain parameters, as described below.

Increasing the number of dimensions in the coordinate space of nodes reduces the routing path length at the expense of a small increase in the size of the routing table [16]. This is because increasing the number of dimensions increases the number of neighbours each node has. Renumbering the coordinates creates multiple coordinate spaces, each being called a reality. This changes the zone assigned to each node, allowing each node to be aware of more nodes and files in the network, which allows for faster routing and resource look up. Increasing the number of nodes per zone also helps, as it results in creating a number of replicas of the index for that zone. Using multiple hash functions allows files to be assigned to multiple nodes, allowing for a greater distribution of information. Since every CAN node has a coordinate along each dimension, the aim of routing queries can be thought of as trying to reach the destination node by traveling the least possible Cartesian distance. However, because adjacent nodes in a CAN can be located many miles and hops apart, every path option at every routing point should be evaluated by dividing the Cartesian distance to be traveled by the network-level round-trip-time (RTT) to get a better estimate of the fastest route. Note that this is a greedy algorithm, and thus not necessarily optimal.

The last strategy used by CAN to improve the performance of the system is Uniform Partitioning. In this strategy, when a new node joins a network, it contacts a random node in the CAN network (an optimization is suggested below). Instead

of adding the new node as a replica in that zone or sharing part of the random node's file-space, the random node compares the volume of its index with that of its neighbours and the node with the largest volume splits its load with the new node.

For efficient resource utilization, the overlay network should map well to the underlying network as closely as possible. When trying to joining the CAN network, a node first measures and orders its network delay to a set of m landmark nodes in an effort to determine its relative position in the Internet. The entire coordinate space is then divided up into m! zones, where each zone corresponds to a certain ordering of the m landmark nodes. Then, instead of joining a random point in any zone in the entire coordinate space (as described above), the node joins a random point in the zone corresponding to its landmark ordering. This allows for an network-topology-aware overlay.

Data keys are cached as they are accessed, where a data key refers to the hash of the file's contents. This allows a node to check its own cache for a data key before issuing a query for that file the next time. A node should replicate high-demand data on its neighbours [16]. Thus, when a query is received for the same file again, the neighbouring node may serve the file directly or forward the query on to the owning node with a less than 100% probability. This would achieve some degree of load balancing.

5.3.1.1 DISADVANTAGES

The CAN architecture assumes knowledge of the entire coordinate space. It would take a significant amount of messaging to keep nodes updated on the status of other nodes. Due to the large amount of time this requires, up to date knowledge of this entire space may not be available to any node at any given point in time.

Although hash functions can help locate content deterministically, they lack the flexibility of partial-match or keyword searching, which are useful operations to find content without prior knowledge of exact object names [9].

CAN assumes that peers have similar capabilities and resources and that they are equally reliable [2]. This does not always reflect reality [3].

CAN assumes a flat content-delivery mesh, where every node is connected to other nodes in a purely peer-to-peer fashion. In reality, nodes on the Internet are connected in more complex and dynamic topologies. Local Area Networks (LAN) and Wide Area Networks (WAN) have protected access to the Internet through a firewall or NAT. ISP's have their own clusters of users. Within each of these LAN's, WAN's and ISP clusters, machines go online and offline randomly. Further study would be required to assess the cost and effectiveness of routing using network locality properties such as RTT [4].

5.3.2 PASTRY AND PAST

Pastry is a generic P2P routing substrate. Various P2P applications can be built on top of Pastry, including those for content-sharing or collaboration. PAST is a persistent file storage system built on top of the Pastry routing substrate.

5.3.2.1 PASTRY

Pastry is a scalable, distributed, object-location and routing substrate for wide-area P2P applications running over a potentially very large overlay network of nodes connected via the Internet [17]. When a node wishes to join the Pastry network, it runs an expanding ring multicast to locate the closest Pastry node. The new node is assigned an uniformly distributed random 128-bit identifier. Given the large space that 128 bits implies, those few nodes that have identifiers close together are likely to be diverse in geography, ownership, and other parameters. No mechanism to maintain uniqueness is specified [17], and thus, based on the birthday paradox, it is a fair assumption that, as the Pastry network grows, there will be some duplication of identifiers. The implication of this is unclear.

The routing mechanism is a form of hypercube routing. Each message carries an application-dependent key. A message is routed to the node with an identifier closest to that key. This is achieved by comparing prefixes of increasing length, leading to a decentralized and structured organization overall. The key may be a hash of the file name and author, as is done in PAST, or it can be a hash of the topic, as is done in Scribe, thus making Pastry's routing mechanism content-based [17].

Pastry's routing scheme takes into account network locality. It minimizes the distance messages travel according to relevant metrics (*e.g.* the number of IP routing hops) and keeps track of immediate neighbours in the node identifier space [17]. To route messages each node maintains a routing table, a neighborhood set and a leaf set. The neighbours are the k closest nodes and the leaves are the k nodes with closest identifiers to the current node, where k is a programmable system parameter that stays fixed once the Pastry system is deployed.

A node can go offline silently. This affects the node's neighbours' routing tables. Although there is some degree of support for repairing the routing tables in Pastry, a silent departure or failure of any node can lead to network partitioning. The situation is dealt with by running an expanding-ring multicast to locate other nodes running Pastry and exchanging routing information with them.

Pastry's routing is more scalable than algorithms such as link-state and distance vector based methods because there is no global propagation of information about routes to each destination. Because of its key-based routing, Pastry has replica management built into it. However, more functionality for caching and replication is built into PAST, the file-sharing layer built on top of Pastry.

Disadvantages Pastry has a number of problems and potential problems.

1. Pastry assumes that all nodes have identical capabilities and responsibilities and all communication is symmetric [17]. This is not always realistic [3]. In particular, it is unlikely to be true in grid computing, cycle stealing opportunistic parallel systems, or voluntary file sharing systems.

2. All experimental results are based only one set of parameters [17]. This provides insufficient experimental data to prove the robustness of the Pastry system.

3. Recovery from node failure requires 57 remote procedure calls. This seems like a very high number of messages [14], especially given the expected lifetime of a node, as discussed in Section 2.1.3.

4. By forcing specific locations for items, it is unclear that it provides a good match for file sharing systems, though it could be used to create a distributed index for such systems.

Pastry's primary limitation is that it is not a widely deployed system, and thus we have no good basis for comparison with systems such as Gnutella and FastTrack.

5.3.2.2 PAST

PAST [18], as noted above, builds a file system on top of Pastry's routing substrate. It hashes a file's name and owner to create a key to associate with the file. An algorithm that assigns the key value uniformly is used. Files are then statistically assigned to storage nodes to maintain a balanced load. The relevance of PAST to this research proposal is in its caching and replication techniques.

PAST stores replicas on r nodes whose node ID is numerically closest to the file identifier. The number r is chosen in proportion to the demand for the file. This way, when a file is in high demand, a node has more replicas to choose from, allowing it to locate a replica that is the best based on a network proximity property such as the number of IP level hops. When a node becomes overloaded with files and replicas, load diversion can be used to offload some files or replicas to other nodes.

PAST nodes can also use unused portions of their disk space to cache files. Files in the cache can be evicted automatically to make room for replicas. The authors suggest caching files near clusters of clients as an additional mechanism to save on bandwidth [18].

Disadvantages PAST does not include file content when generating a key for it. This means that a file by another name, although identical in content, will be assigned a different key. This can have detrimental effects when injecting files into the PAST network and locating replicas.

PAST does not support directory look up and keyword searching to retrieve a file. This limits the system to searching for content without prior knowledge of exact object names or the file identifier [9].

To improve bandwidth, the authors [18] have suggested a Gnougat-like approach of striping a file over several distributed disks. However, more research needs to be done to analyze the tradeoffs between the latency and bandwidth savings against the cost of aggregate query, network load, and availability.

5.3.3 CHORD

The Chord protocol is essentially the same as Pastry for routing. The fundamental difference lies in the fact that Chord routes messages based on numerical difference between the next node's identifier and the destination ID, while Pastry attempts to route messages toward nodes with successively longer address prefixes in common with the destination identifier [4]. Hence, we omit a detailed analysis of this protocol.

Chord has a decentralized and structured organization. Its routing mechanism uses the node identifier, which itself may or may not be content-based. No details are provided on caching and replication in the Chord specification, primarily because the focus of Chord is its routing mechanism and not replica management. A file-sharing application layer built on top of Chord can deal with caching and replication.

5.3.3.1 DISADVANTAGES

In addition to the disadvantages suffered by Pastry, the Chord protocol suffers from the fact that it does not take advantage of network proximity. As a result, its protocol for maintaining the overlay network is very lightweight but messages may travel arbitrarily long distances in the Internet in each routing hop [17]. However, this is not an intrinsic problem, as Chord can pick the next routing hop from among many nodes using something like the network-level latency rather than picking the first node it finds according to its formula.

A suggested optimization is maintaining a location table of all other peers encountered and their network distance, so that a node can look up this table to determine the next node to route a message to, rather than running the Chord algorithm to determine an appropriate node identifier [2]. A load-balancing optimization is

to assign multiple identifiers to every node based on its capabilities and how much load it can handle [2].

5.4 PUBLISH/SUBSCRIBE-BASED

The final system that we examined uses the publish/subscribe paradigm. This approach is used by JXTA (Juxtapose).

5.4.1 JXTA

JXTA technology is a set of P2P protocols composed of 3 layers [4]:

1. The core layer, at the bottom, deals with routing and peer establishment.
2. A service layer, in the middle, deals with the higher-level concepts of indexing, searching and file sharing.
3. An application layer, at the top, deals with applications such as email, auctioning, and storage systems.

JXTA uses a publish/subscribe-based architecture at the service layer to advertise and locate resources such as files and services [4]. Peer-discovery mechanisms currently supported include LAN-based broadcast, peer invitation, cascaded discovery between peers, and rendezvous points which are nodes that maintain and share knowledge of other peers in the network [4].

5.4.1.1 JXTA SEARCH

JXTA search is used to index content in P2P systems to allow easier searching in a decentralized and structured way. It consists of consumers, providers and search hubs. Consumers are peers that are looking for information. Providers are peers that provide information. Search hubs are the peers that register resource advertisements in XML form from the providers and help the consumers find these resources. Any node can act in any/all of these capacities.

The search hubs serve two functions: registering advertisements and answering queries. All search hubs are divided up into a number of groups, either based on content or in a content-agnostic manner. Each hub in such a group maintains an inverted index of registrations it receives. All hubs in a group exchange knowledge of each other's registrations in summary form using a Bloom Filter [2], which is a way of hashing keys (in this case, filters for available information) into a condensed bit array [5].[1] One or more members of each such advertisement group serve the

[1]The Bloom Filter approach is described in more detail in Appendix A.

role of responding to queries directed toward their group. Having knowledge of its own inverted index as well as a summary of the indices of the other hubs in its group, the query hub can redirect queries it receives. If the query cannot be answered within that group, it is broadcast to other groups. The hubs that are picked to represent each group may be picked based on metrics such as physical distance. Such an arrangement helps reduce the query response time since nodes only need to contact one hub, which has knowledge of other hubs in its advertisement group. The small degree of centralization introduced by the search hubs allows policies to be implemented easily. JXTA Search itself does not deal with caching and replication issues. Storage management systems built on top of JXTA Search would deal with these issues.

Disadvantages Search hubs need to forward information about subscription changes to all search hubs in their advertisement group. Depending on the size of the registrations and the network-level distance between the search hubs, this may lead to scalability problems.

A publish-subscribe architecture like JXTA's requires a match-making process. This approach itself can suffer from scalability problems.

5.4.1.2 GNOUGAT

A part of the JXTA initiative, Gnougat is a technique to increase download speed and reduce bandwidth consumption when downloading files in a P2P network. Files are identified by a hash of the contents. This allows the system to automatically cache both content and queries based on the hash value. To facilitate downloading, files are broken down into chunks, where a chunk is defined as a "fixed-size portion of file content" [10]. Chunking can be used as a means to resynchronize file downloads when abruptly interrupted, and increase the parallelism of the transfer process by downloading different chunks from different sources. Chunks themselves are hashed to create a key for each chunk. The keys of all the chunks are included as part of the descriptor of the original files. Gnougat's communication scheme is similar to Gnutella's. To reduce the broadcast overhead of Query messages, Gnougat caches descriptors of files. State information called crumbs are left behind in the routing tables of intermediate nodes so that responses to broadcast messages can be routed back following the trail of these crumbs. Chunks are also cached. Gnougat assumes that searches are based on hash values, which is possible since once the desired file is located using JXTA Search, its descriptor can be looked up to determine the chunk keys.

Disadvantages If the chunk size adapts to network conditions, then the chunk identifiers will change. Since this would substantially complicate search, it is doubtful that this approach would be taken. Given a fixed chunk size, it is not clear what the optimal size would be.

Since Gnougat's communications are similar to Gnutella's, it suffers from the same message overhead as Gnutella except that caching descriptors helps alleviate this overhead. Gnougat does not address the issue of routing to the node containing the desired chunk, or locating a suitable node to store a replica of a chunk.

As with Pastry, JXTA is not widely deployed, and so there is no good basis for comparison with systems such as FastTrack.

6 PROPOSALS

Based on this study we plan to explore a number of research possibilities. These include network-carrier performance in the face of mismatching overlay peer-to-peer protocols, providing web services over peer-to-peer networks, the relationship between *ad hoc* networking protocols and peer-to-peer protocols, efficient peer-to-peer networking protocols, security and anonymity issues, and alternate uses for P2P computing.

6.1 NETWORK CARRIER PERFORMANCE

P2P traffic represents something of a dilemma for Internet Service Providers (ISPs). On the one hand, the presence of file-sharing content attracts customers to their services. However, the large amount of protocol chatter and the mismatch between the P2P overlay network and the underlying ISPs network can cause substantial network traffic. This is especially true for the poorly-designed but popular file-sharing schemes such as FastTrack, Gnutella, and Kazaa.

Worse, an ISP pays transit charges whenever traffic crosses its network boundary. Since the overlay is a poor match for the underlying network, a copy of the content may reside within the ISP's network, and yet not be reached by the P2P client search. Rather, a distant copy may be reached first. This can significantly increase an ISP's transit changes, with no good reason.

There are essentially two problems to be addressed here: protocol chatter and P2P network structure. While these problems can be addressed by creating more efficient P2P networks (see Section 6.4), ISPs will still have to deal with a large base of customers who are happy with their current (inefficient) P2P systems.

There are several possible solutions to one or both of these problems, though many of these ideas have deficiencies of one form or another. Perhaps the most obvious solution would be to have customer cost reflect carrier cost. Thus, if a customer is using a poor-quality P2P overlay system, s/he will pay more to the ISP, while a customer with a good-quality P2P system would pay less. There are several points in this solution that make it unattractive. The most notable problems are that few customers will be aware of the cause of the higher billing, and, in a highly competitive ISP market, most will simply move to a different ISP that provides flat-rate billing.

A second solution would be for the ISP to place peers within its network, and then manipulate the P2P traffic such that those peers are preferred. For example, a FastTrack super-node could perform such actions. In conjunction with this, the

ISP's peer node could replicate popular files, so that queries would be either satisfied directly from the ISP's node, or redirected from that node to other nodes within the ISP's network. The primary problem with this approach is not technical, but legal. Much of the file-sharing content is copyright, and thus such a peer would violate copyright law, opening the ISP to suite for infringement.[1]

The third solution is to simply cache files requested by P2P clients. While this is worth examining, at least some ISPs are uncomfortable with this, for the same reason that they reject the second solution.

There is an alternate solution that we plan to explore. If the ISP is able to monitor the various P2P traffic, it can locate P2P files that shared by its various customers. Rather than caching the file content, the ISP may cache knowledge of the file's location. It then becomes possible to redirect the request for files that users within the ISP's network share from a distant node to the node within the ISP's network. In effect the users' machines on the ISP's networks act as the cache, rather than the ISP itself.

There are several issues involved with this approach. First, the technical details of how this might be accomplished have to be worked out. Specifically, we need to resolve how files are identified, how this knowledge is accumulated and where it is stored. Some form of content hash is likely required for the first of these problems. The storage can be at the edge of the ISP's network, on the presumption that the major cost of relevance is the transit cost. Knowledges accumulation, however, would have to come from throughout the network, and be forwarded to the edges.

Second, the implications of redirecting a file get request need to be thought through. In particular, it is possible that acquiring the distant file is legal, while getting a copy of the local file is not. There are variations on this solution that may make is acceptable to ISPs. In particular, rather than redirecting the get file request, the search might be redirected, as follows.

We start by building an index of all hosts connected to the P2P network. Note that this is not a supernode in the P2P network. It is simply caching knowledge of nodes that are part of the P2P network. The first time a P2P search query is noticed by the ISP, it is redirected to other P2P nodes within the network. Insofar as those peers can satisfy the request, the ISP will experience a reduction in transit traffic.

Peers within a provider's domain will not always be able to satisfy a query. The issue is then one of how to ensure that the first query goes to the P2P nodes within the network, and the repeat query is permitted to exit the network. A simple solution is for the edge routers to selectively redirect requests that would exit the provider's network. If we set the probability of redirection to p, then three attempts

[1] Note that the fact that the content traverses the ISP's network is not a copyright violation, as the ISP would satisfy the "common carrier" rules.

to query for a file would have probability $(1 - p^3)$ of exiting the ISP's network. Given the 50% success rate of systems such a Gnutella, it is doubtful if a user would observe the behavioral difference for a carefully selected value of p. The benefit to the ISP would be a factor of p reduction in transit traffic when the file is local. We plan to investigate more sophisticated solutions, though keeping in mind that the return path is not guaranteed to be the same as the forward path, nor is the path of the repeat query necessarily the same as that of the original query.

The ISP may also monitors data transfer outward from its domain and, based on latency measurements, maintain a sorted inverted-index of resources and which peers have them. This index can be used to help redirect the query to a preferred node. Note again, the query will be redirected, not the file request itself.

The index of peers inside the ISP's domain and the inverted index of resources that have been downloaded by outside peers must be maintained using some user session statistics. Specifically, entries that are older than a certain amount of time should be purged on a periodic basis. This procedure effectively allows the ISP to "comb" through the network to untangle expensive P2P connections and readjust the P2P graph so that the graph better matches the underlying IP infrastructure.

This approach is clearly predicated on a particular type of P2P system. In particular, it presumes non-deterministic and/or expanding-ring protocols. The behaviour of deterministic P2P systems will depend heavily on how they treat perceived dead peers. If is also predicated on the notion that there are alternate peers that can be contacted besides distant ones. If the query has to start at a distant node, then alternate means of bandwidth reduction would have to be investigated. Finally, these solutions only work if the P2P communication is not encrypted. If communication is encrypted it would not be possible to monitor P2P traffic and know when an inside host connects to a P2P network and when an outside host downloads a data item from an inside host.

6.2 WEB SERVICES

Web Services, in the broad sense of the term, is a standards-based model for applications and web sites to be interoperable with each other. A more narrow, technical definition is that Web Services is remote method invocation (RMI) where all parties to the standard agree on the various common formats required (*e.g.* binder (or name server) query format, presentation of data format, *etc.*). The set of infrastructure services required for Web Services are the following:

Discovery: Client programs or programmers need a way to discover the desired Web Service. In tradition RPC/RMI systems, this is the function of the binder or name server. Finding such a server has been the domain of distributed name services research.

Description: Once a Web Service is located, the client needs to know a description of the Web Service functionality and how to use it. This includes method names, parameter types, and return types. This is the problem of dynamic invocation in distributed object systems. In a system such a CORBA it was resolved by querying the interface repository. This presumes that syntactic description is sufficient. In many instances this is likely not enough. What is desired is a semantic model of what a service does, which can be discovered by its semantic model. Invocation would then match the particular foibles of the syntactic interface for that semantic service.

Wire Format: When a client invokes a Web Service method, the method call must be serialized into a format recognizable by the Web Service. This is the presentation-layer problem (OSI layer 6). Several methods have been used in the past, including ASN.1, XDR, and NDR. Web Services uses XML.

The components of Web Services are as follows [22]:

HyperText Transfer Protocol (HTTP): This acts as the transport protocol of Web Services.

XML: The eXtensible Markup Language provides a platform-independent way to mark up data, providing (in effect) a common OSI Presentation Layer.

Simple Object Access Protocol (SOAP): This provides the common format for messaging in Web Services. It amounts to an asynchronous remote method invocation.

Universal Description, Discovery, and Integration (UDDI): This is an XML-based grammar that describes the format of advertisements to be used by Web Service providers when advertising their Web services. UDDI is what allows client programs and programmers to discover Web Services. It thus takes on the role of the interface to the naming service.

Web Service Description Language (WSDL): This is an XML-based grammar that describes the layout of the SOAP messages accepted by a particular Web Service. When a client program or programmer discovers a Web Service, the client downloads a WSDL document that describes the method names, parameter types, and return types supported by the Web Service. This thus represents the interface repository in CORBA terminology.

Even with this very brief view of Web Services it is apparent that there is a distinct similarity in purpose between the Web Services and peer-to-peer technologies. A question that we ask ourselves is, given this similarity of purpose, and assuming

a layered architecture, is it of value to place Web Services over peer-to-peer networks. For example, JXTA can be likened to the P2P version of Web Services. JXTA is essentially a system in which the registration hubs act as mini-UDDI registries with descriptions of the services registered with them in the form of XML advertisements. These registration hubs communicate with each other using SOAP or arbitrary well-formed XML to help peers locate resources. However, while Web Services are based on a centralized model, JXTA depends on absolute decentralization. We would like to explore the possibility of combining the two architectures, bearing in mind that a centralized architecture exhibits faster search and directory update and lesser messages exchanged, but also potentially suffers from a single point of failure and performance bottleneck. We believe that exploring this possibility of Web Services over P2P can lead to a highly scalable, decentralized Web Service environment.

6.3 *Ad Hoc* AND P2P

Wireless *ad hoc* networks are similar to P2P networks in various ways. In particular, in both cases nodes can only communicate directly with their neighbours. Thus, in both cases the key issues are resource (or service) discovery and resource acquisition. The primary distinction between the two technologies are that nodes are considered mobile in the *ad hoc* environment, while there are generally fixed within P2P networks. That said, given the relatively short life expectancy of a peer node, P2P networks face a similar challenge of "disappearing" resources. We thus wish to explore whether various of the *ad hoc* protocols (routing, discovery, *etc.*) can be exploited to improve P2P capabilities.

6.4 EFFICIENT PROTOCOLS

The use of *ad hoc* protocols to improve P2P efficiencies and capabilities is but one approach in the broader category of creating efficient P2P protocols. Some other ideas we have in this area include the view that we should move toward using multiple resource identifiers, rather than single identifiers. These additional identifiers could correspond to search categories, such as author, filename, last update date and time, file type, file size and possibly a content hash when trying to locate replicas. The search words could be hashed into numbers, resulting in the final resource identifier being a tuple of individual identifiers, though this would seem to preclude partial-match searching. We expect to use multidimensional-database indexing techniques for storing the resulting routing table.

Some of the issues in this scheme that we need to explore are:

1. How do we effectively maintain separate routing information given that partial-

order theory indicates that we cannot flatten identifiers into a single big identifier

2. How do we deal with fields that are specified using wild cards such as "book*"? Is it possible to hash the search words in such a way that, for example, the hash value for "book" would also be the prefix of the hash value for "bookstore"? Or should we numerically encode quantifiable metadata such as the resource type and store the rest of keywords in a tag of text (*e.g.* using Extensible Markup Language (XML)) on another dimension? The complexity of resources probably means that such and approach is necessary, perhaps requiring registration hubs to help locate peers based on key-tag descriptions.

3. To ease wild-card searches, one possible technique may be to encode the entry for that dimension as all zeroes. Re-ordering the individual identifiers to make the wild-card field the least significant field in the tuple may help route the query to a node in the vicinity of the destination node. Would multidimensional-database indexing techniques improve such an operation?

4. Given the heterogeneous and dynamic nature of P2P networks, how frequently do we need to update the routing tables? Should the routing information be updated along each dimension separately or is it possible to flatten the index as described above and then update the table in one go?

5. Although the focus of our survey is sharing immutable files, having a date and time factor as a dimension may assist in querying for documents that do change with time.

6.5 Security and Anonymity

As observed by Biddle, England, Peinado, and Willman [1], many people are reluctant to expose their computers for sharing purposes due to limitations in security and anonymity. We therefore plan to explore the various sandbox approaches (such as the Java sandbox) as well as the use of trusted operating systems such a Palladium [**?**] to see if these could enable the provision of both end-user security and anonymity.

6.6 Alternate Uses

This proposal has focused on the use of P2P networks for file sharing. However, as already noted in our proposals, it is conceivable to also use these systems for

general Web Services. Indeed, we know of no fundamental reason why they cannot be used for other purposes. The Grid Computing community is currently exploring and/or using various forms of peer-to-peer systems for their purposes. We therefore plan to inventory the various uses of P2P systems, wit ha view to determining where they are useful, where thy have limited value, and what, if anything, is the common theme that runs through the various uses. We expect that they will prove to be of substantive benefit in the pervasive computing area.

A BLOOM FILTER SUMMARIES

An ideal summary representation should be small in size and have a low false-hit ratio [5]. The Bloom Filter provides these features. Computationally efficient, it can represent a set of keys with minimal memory requirements, while answering membership queries with zero probability for false negatives and low probability for false positives [5].

The procedure is described as follows [5]. Given a set $A = \{a_1, a_2, \ldots, a_n\}$ of n keys, a Bloom filter translates these n keys to a vector v of m bits, which is the memory requirements. All bits in the vector v are initially set to 0. Then using k independent hash functions, $h_1, h_2, \ldots, h_k$, each with range $\{1, 2, \ldots, m\}$, each element $a \in A$ is hashed and the bits at positions $h_1(a), h_2(a), \ldots, h_k(a)$ in v are set to 1. A particular bit might be set to 1 multiple times and a count may optionally be maintained separately. Given a query for b we check the bits at positions $h_1(b), h_2(b), \ldots, h_k(b)$. If any of them is 0, then $b \notin A$. Otherwise, b may be in the set although there is a certain probability of a false positive. The parameters k and m should be adjusted to bring the probability of a false positive within an acceptable range.

After inserting n keys into a table of size m, the probability that a particular bit is still 0 is exactly $(1 - 1/m)^{kn}$. Hence, the probability of a false positive is:

$$\left(1 - (1 - 1/m)^{kn}\right)^k \approx \left(1 - e^{-kn/m}\right)^k$$

The right hand side is minimized for $k = (m/n)\ln(2)$ or equivalently, $1/2^k = (0.6185)^{m/n}$. Thus, under optimal k, the probability of a false positive reduces exponentially with increasing m. Since k is the number of hash functions, it must be an integer. The probability of a false positive exponential decreases as the number of bits per entry (*i.e.*, m/n) increases. Bloom filters require little storage per key at the slight risk of some false positives.

REFERENCES

[1] Peter Biddle, Paul England, Marcus Peinado, and Bryan Willman. The darknet and the future of content distribution. In *Proceedings of the ACM Workshop on Digital Rights Management*, New York, November 2002. ACM Press.

[2] S. Botros and S. Waterhouse. Search in JXTA and other distributed networks. In *Proceedings of the First International Conference on Peer-to-Peer Computing*, pages 30–35. IEEE Computer Society Press, August 2001.

[3] Marshall Brain. How file sharing works, September 2002. Available at http://www.howstuffworks.com/file-sharing.htm.

[4] Miguel Castro, Peter Druschel, Y. Charlie Hu, and Antony Rowstron. Exploiting network proximity in peer-to-peer overlay networks. Technical Report MSR-TR-2002-82, Microsoft Research, Cambridge, CB3 0FB, UK, 2002. Available at http://www.research.microsoft.com/ antr/PAST/location.pdf.

[5] Li Fan, Pei Cao, Jussara Almeida, and Andrei Z. Broder. Summary cache: A scalable wide-area web cache sharing protocol. In *Proceedings of the ACM SIGCOMM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication (SIGCOMM '98)*, New York, September 1998. ACM Press.

[6] R. Gold and D. Tidhar. Towards a content-based aggregation network. In *Proceedings of the First International Conference on Peer-to-Peer Computing*, pages 62–68. IEEE Computer Society Press, August 2001.

[7] Google. Google corporate information, December 2002. Available at http://www.google.com/corporate/facts.html.

[8] Dennis Heimbigner. Adapting publish/subscribe middleware to achieve gnutella-like functionality. In *Selected Areas in Cryptography*, pages 176–181, New York, 2001. ACM Press.

[9] Hung-Chang Hsiao and Chung-Ta King. Modeling and evaluating peer-to-peer storage architectures. In *Proceedings of the Parallel and Distributed Processing Symposium*, pages 240–245. IEEE Computer Society Press, April 2002.

[10] Gene Kan and Yaroslav Faybishenko. Introduction to gnougat, September 2002. Available at http://www.jxta.org/gnougat.pdf.

[11] Kazaa. New to kazaa? The big 3 ways to better and safer file sharing!, September 2002. Available at http://www.kazaa.com/en/help/big3.htm.

[12] Nathaniel Leibowitz, Aviv Bergman, Roy Ben-Shaul, and Aviv Shavit. Are file swapping networks cacheable? Characterizing p2p traffic. In *Proceedings of the 7th International Workshop Web Content Caching and Distribution*, August 2002. Available at http://2002.iwcw.org/papers/18500253.pdf.

[13] Qin Lv, Pei Cao, Edith Cohen, Kai Li, and Scott Shenker. Search and replication in unstructured peer-to-peer networks. In *Proceedings of 16th ACM International Conference on Supercomputing(ICS'02)*. ACM Press, June 2002.

[14] Sudarshan Murthy. Peer-to-peer (P2P) storage systems, November 2002. Available at http://www.cse.ogi.edu/ smurthy/p2ps/p2ps.zip.

[15] Marius Portmann, Pipat Sookavatna, Sebstien Ardon, and Aruna Seneviratne. The cost of peer discovery and searching in the gnutella peer-to-peer file sharing protocol. In *Proceedings of the 9th IEEE International Conference on Networks*, pages 263–268. IEEE, September 2001.

[16] Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard M. Karp, and Scott Schenker. A scalable content-addressable network. In *Proceedings of the ACM SIGCOMM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication (SIGCOMM 2001)*, pages 161–172, New York, August 2001. ACM Press.

[17] Antony Rowstron and Peter Druschel. Pastry: Scalable, decentralized object location and routing for large-scale peer-to-peer systems. In *Proceedings of the IFIP/ACM International Conference on Distributed Systems Platforms*, pages 329–350. ACM Press, November 2001.

[18] Antony Rowstron and Peter Druschel. Storage management and caching in PAST, A large-scale persistent peer-to-peer storage utility. In *Proceedings of the 18th ACM Symposium on Operating Systmes Principles*, pages 188–201. ACM Press, October 2001.

[19] Sandvine. Peer-to-peer file sharing: The impact of file sharing on service provider networks. Technical report, Sandvine Incorporated, 408 Albert Street, Waterloo, Ontario Canada N2L 3V3, December 2002. Available at http://www.sandvine.com.

[20] Stefan Saroiu, P. Krishna Gummadi, and Steven D. Gribble. A measurement study of peer-to-peer file sharing systems. In *Proceedings of Multimedia Computing and Networking 2002 (MMCN '02)*, San Jose, CA, USA, January 2002.

[21] Bernard Traversat, Mohamed Abdelaziz, Mike Duigou, Jean-Christophe Hugly, Eric Pouyoul, and Bill Yeager. Project jxta virtual network, February 2002. Available at http://www.jxta.org/docs/JXTAprotocols.pdf.

[22] Web Services-Interoperability Organization. Basic profile, October 2002. Available at http://ws-i.org/Profiles/Basic/2002-10/BasicProfile-1.0-WGD.htm.