FASD: A Fault-tolerant, Adaptive, Scalable, Distributed Search Engine

Amr Z. Kronfol
Princeton University

May 6, 2002

Abstract

This paper introduces FASD, a fault-tolerant, adaptive, scalable, and distributed search layer designed to augment existing peer-to-peer applications. The FASD layer operates as a network of identical nodes that collectively pool their storage space to cache "metadata keys" and cooperatively route queries to the nodes most likely to satisfy them. A "metadata key" is a list of weighted terms that describe the information content of a document in the underlying network. Although completely decentralized, FASD's approach is able to efficiently match the recall and precision of a centralized search engine. Simulation results indicate that latency and bandwidth consumption scale logarithmically with the size of a FASD network.

_____

Amr Z. Kronfol

Contents

List of Figures

Chapter 1

Introduction

The evolution from the modem-based world to one of "always-on" broadband connectivity has seen a surge in the popularity of peer-to-peer[1] systems (see [32] for a good overview). This increase in popularity has spawned interesting technical and social issues. For example, Gnutella [15, 18, 21, 36] has experienced scalability problems and the courts shut down Napster [30] due to alleged copyright violation. Freenet[8–10, 14] is an adaptive peer-to-peer network that addresses some of these issues. Its developer, Ian Clarke, envisioned an anonymous and decentralized paradigm that avoided a single point of vulnerability for authoritarian (or benevolent) regimes to target.

1.1 Freenet: Strengths and Weakness

Notwithstanding the soundness[2] and legal viability of Freenet's political aspirations, its unique performance and security characteristics are important on their own merit. The underlying algorithm adopts a meritocracy-based routing and storage scheme whereby increased popularity of a data item translates to more aggressive caching and lower retrieval latency. In light of

---

[1]Peer-to-peer is a contentious term encompassing any form of distributed computation. A stricter interpretation might require a large network of clients that transmit and receive information from one another without a mitigating server. The model is certainly not novel, dating back to at least 1979 with the introduction of the Usenet system [29].

[2]In a recent article O'Reilly's CTO Jon Orwant estimated that 15.6% of Freenet content was pornographic and 53.8% fell into the broad category of sex, drugs, and rock and roll [33]. In recent e-mail correspondence, Ian Clarke pointed out, "the research is very out-of-date, the nature of content on Freenet has changed dramatically since those early days, of course there is still 'sex, drugs, and rock n' roll', but the percentage given bears little relationship to reality now."

the load problems experienced by world wide web servers during a denial of service attack or a major news event (i.e. SlashDot effect [1]), this property becomes very significant. Moreover, it is difficult for a malicious party to remove data from the network because the process of locating the data serves to further replicate it. Andrew Shapiro [47] and Lawrence Lessig [27] have argued that the Internet is evolving into an architecture of complete control. To the extent that it uses cryptography and strictly limited upstream and downstream communication, the completely decentralized Freenet system helps combat this disturbing trend. Although not completely immune to eavesdropping, there is "good-enough" producer, consumer, and server anonymity.

Freenet's searching capability is not nearly as laudable. All data on the network is identified by arbitrary GUIDs[3] that users must discover through out-of-band means. An internal mechanism that supports search[4] does not exist. Freenet's inability to capture critical mass[5] coupled with the fact that 85% [26] of web users depend on search engines suggests the restrictiveness of this shortcoming. Integrated search capability is critical to the utility of the network as a whole and imperative if Freenet's unique qualities are to become truly accessible.

1.2   A Distributed Search Engine

This paper presents FASD, a completely distributed search engine, designed to enrich Freenet with search while not compromising scalability, anonymity, or fault-tolerance. Although framed as an augmentation to the Freenet model, FASD is applicable to other peer-to-peer systems and it could also form the basis of a new, search-optimized architecture. Since Freenet's strict anonymity requirements prohibit a number of performance optimizations, the results in this paper are an upper bound on latency and bandwidth consumption. Section 3.8 discusses FASD's broader applicability to any application that could benefit from a search capability that is more scalable, distributed, adaptive, or fault-tolerant.

---

[3]GUID is an acronym for globally unique identifier.

[4]Search is the ability to identify which documents match a given query string. For example, in Freenet, it is impossible to identify which documents in the networks most likely contain information about "apples `AND` oranges `NOT` bananas".

[5]Ian Clarke estimates the number of downloads per day in Freenet at 5,000-10,000. Although impressive for a system as young as Freenet, search capability is certainly necessary if the system is to become more widely accessible.

FASD's approach involves automatically generating a metadata key[6]—a list of terms that describe the information content of a given document—whenever an author inserts a document[7] into the system. A closeness operator derived from classic information retrieval allows a FASD node to determine how closely associated the information content of two metadata keys is (and thus how closely associated the information content of the underlying documents is). Similarly, the operator allows a FASD node to determine which metadata keys are closest to a given query (and thus which underlying documents are closest to the given query). When storing a metadata key k, a node also stores an associated "reference"—the address of a node likely to specialize in requests for metadata key k. A node is said to specialize in k if it contains documents whose metadata keys are close to k and if it has references to other nodes likely to specialize in k. References define the connectivity of a FASD network. Node i is connected to node j if node i contains a metadata key whose associated reference is node j. A node forwards a request for query q or an insert of key k to the downstream neighbor most likely to contain more metadata keys that are close to query q or key k. This process continues until the hops-to-live[8] of the message expires. The nodes contacted in this process are said to form an insert or request chain. Each node in an insert chain saves a local copy of the inserted metadata key. Each node in a request chain caches a local copy of the keys closest to the requested query before passing them upstream. Over time, FASD's consistent routing algorithm and aggressive caching scheme permit nodes to "learn" how to better service requests for keys they are deemed likely to specialize in.

The remainder of the paper is as follows: chapter 2 provides an overview of the state of the art (including an in-depth discussion of the Freenet system); chapter 3 presents the algorithms and protocol that comprise the FASD distributed search engine; chapter 4 describes the simulation and experimental results; chapter 5 suggests directions for future work; concluding remarks are in chapter 6.

---

[6]Ian Clarke first coined the term "metadata key" in an e-mail thread discussing future directions for Freenet.

[7]Although only text data is currently supported, the approach is easily extended to other media if authors are willing to manually generate metadata. In the popular MP3 format, album, genre, title, and artist information are already directly encoded into the file using CDDB.

[8]The hops-to-live is the number of times a request or insert may be forwarded before it expires.

Chapter 2

Related Work

The various search schemes in the current peer-to-peer landscape fall into one of three broad categories—centralized search, hash-based index distribution, and query broadcast. This chapter discusses each of these schemes to conclude they are *ad hoc*, failing to match one or more of FASD's primary characteristics—fault-tolerance, adaptability, scalability, and distribution. Section 2.4 provides an in-depth summary of the Freenet peer-to-peer system which serves as the inspiration for FASD's metadata key routing algorithm.

2.1 Centralized Search

In its simplest form, centralized search is a three step process. The search engine first identifies and aggregates all the document in the collection via an automated process (e.g. web crawler) or manual entry. It then inserts the documents into a data structure optimized for efficient search. Typically, this consists of an inverted index: a list of terms in the collection and pointers to documents that contain those terms. Finally, when processing a query, the search engine retrieves from the index the set of documents containing each query term. Depending on the Boolean operators (i.e. `AND`, `OR`, and `NOT`) the result set is taken to be the union, intersection, or difference of the individual sets.

Peer-to-peer systems may use a centralized index for document discovery. In this model, nodes update the centralized index as they store or delete documents. A user then sends queries to the central index and receives a list of "hits"[1] to the query. Each hit includes a pointer that permits the

---

[1]A document is a "hit" if its information content is close to the query. Often, a hit is also assigned a rank describing how closely it matches the query.
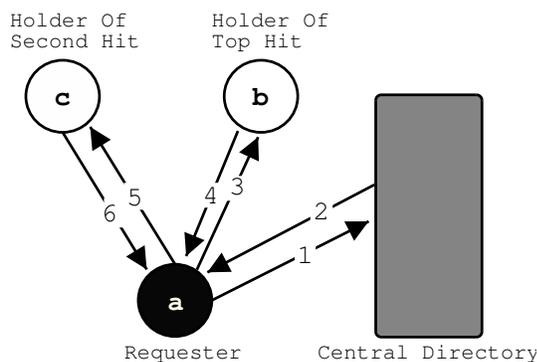
Figure 2.1: Search and retrieval under the Napster model

user to retrieve the corresponding document. Ideally, the top hits will meet the user's information need.

2.1.1 Napster

In the peer-to-peer world, Napster's was the most prominent use of a centralized index. Whenever a user stored an mp3 music file, his node contacted the central directory and inserted the song title, artist name, genre, and so on. Due to the limited amount of information describing a given song, Napster did not utilize the sophisticated[2] weighting and matching techniques necessary in more general information retrieval systems where documents may be long and complex [44, 46].

Figure 2.1 depicts a typical search and retrieval scenario. (1) Node a sends a query q to the central server. (2) The central server processes q and returns to a a list of nodes containing hits. (3) Node a contacts the node associated with the top hit, node b. (4) Node b transfers the song to node a via a direct TCP/IP connection. However, the returned song does not satisfy the user's information need (e.g. it is truncated prematurely or erroneously labeled). (5) Node a then contacts the node associated with the second-best hit, node c. (6) Node c transfers the data to node a. The user is satisfied with this hit and the search/retrieval process is complete.

As the approach of choice for web search engines, the centralized paradigm

---

[2]At the very least, these techniques need to account for the importance of a given term in identifying a given document. For example, the term "the" is less useful in identifying a document about "apples" than the term "fruit" is.

has benefited from substantial research into its scalability[3] with respect to storage and retrieval latency (see [6, 19, 24] for an overview). However, Napster's dependence on this model also had serious disadvantages. It introduced a single point of failure: once the central server was shut down, the still intact network was unable to function because data location and peer discovery became impossible. Furthermore, this approach was prone to the same load issues facing the traditional client/server model. Finally, a user's identity was exposed during queries and during download because of the direct connection necessary for node-to-node and node-to-central-server communication.

2.1.2 Freegle

There is an *ad hoc* searching mechanism for Freenet known as Freegle [13] that utilizes a centralized search approach. Authors of Freenet documents manually enter a description of their document and its Freenet GUID into a centralized search index. Users then execute queries on the index and retrieve the matching documents from the Freenet network using the associated GUID. Although Freegle preserves download anonymity (GUID requests in Freenet are anonymous), it exposes the user's anonymity during the query process. More significantly, should the central search server become disabled[4], Freegle would render Freenet's fault-tolerance, scalability, and decentralized properties a moot point to users dependent on search.

2.2 Hash-based Index Distribution

The fundamental primitive in a hash-based peer-to-peer systems is LOOKUP, a function that returns the node responsible for storing data with a given GUID [25, 37, 50]. Chord [50] implements this function using a consistent hash [22] with some extra[5] routing information for improved efficiency. A consistent hash's ability to change minimally as the range of the function changes is critical in the transient world of peer-to-peer. The Chord system guarantees $O(\log N)$ running time for its LOOKUP function where N is the

---

[3]For example, the Google search engine conducts more than 150 million searches per day on an index of over 2 billion web pages [17].

[4]Targeted attack, court injunction, or overload might disable the central server.

[5]For minimal performance, each Chord node need only be aware of its successor node on the identifier circle [50].

number of nodes in the network[6].

2.2.1 Index Distribution using Chord

There has been some discussion in [50] of a distributed search application of the Chord framework. In this model, the node given by LOOKUP(t) maintains a list of all documents in the network containing the term t. When processing the the query $Q = t_0$, the requesting node i contacts node j where $j = $ LOOKUP(t_0). Node j then transfers a list of all documents containing the search term t_0 to node i. In this particular scenario, Chord seems to offer an $O(\log N)$ distributed search. An immediate issue is term popularity—some index terms may be very popular and, as such, could drive an unsupportable amount of traffic to a single node (or small set of nodes). Developing a load balancing mechanism that breaks up and distributes the indices for popular terms is non-trivial. However, there are more fundamental flaws in this search model. Insertion of a document $D = t_0, t_1, \ldots, t_m$ with m unique terms requires m LOOKUP calls, resulting in a debilitating amount of network traffic ($m \log N$ messages). Complex Boolean queries are also prohibitively expensive. To satisfy a query in the form $q = t_0$ `AND` t_1 `AND` $\ldots$ `AND` t_m the requester must execute m LOOKUP calls, download m potentially large indices, and determine the intersection of m potentially large result sets. Chord's architecture does not effectively distribute the computation necessary for document insertion and query execution—the onus is on the initiating node to coordinate the process using a prohibitive amount of bandwidth in LOOKUP calls and file transfers.

2.3 Broadcast

In the broadcast approach, each node searches its local index for a given query and then forwards the query to all of its neighbors. The HTL (hops-to-live) of the query indicates its depth or search horizon. Once the HTL has expired, nodes no longer broadcast the query further downstream. Each node passes its hits back to the initial requester who then selects which documents to download. The broadcast system amounts to breadth-first search with cutoff.

---

[6]To achieve $O(\log N)$ LOOKUP performance, each node maintains routing information for $O(\log N)$ other nodes. Additionally, $O(\log^2 N)$ messages are required to preserve the network's consistency on node entry and exit [50].
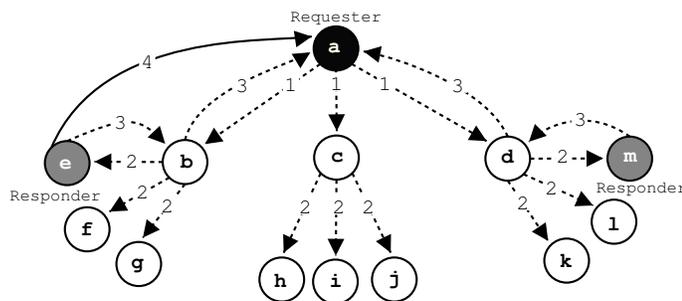
Figure 2.2: Search and retrieval under the Gnutella model

2.3.1 Gnutella

Gnutella is a network of equal peers that selectively respond to queries. Each node attempts to establish a number of simultaneous connections to other nodes that constitute its neighbor set.

Figure 2.2 depicts a typical search and retrieval scenario. (1) Node a broadcasts a query q to all nodes in its neighbor set N. (2) Each node in N forwards q to its neighbors. (3) Any node able to satisfy q (in this case nodes e and m) returns via the request chain a brief description of its hits to node a. (4) After reading the descriptions, the user chooses to download one of the hits returned by e (via direct TCP/IP connection). The user is satisfied with this hit and the search/retrieval process is complete. Had the user been unsatisfied, he could retrieve any of the other hits in e's result set or m's result set.

The Gnutella protocol does not specify how nodes must process incoming queries. Thus, it is well-suited for search across heterogeneous data sources. For example, when evaluating the query string "1+1", one node might recognize a mathematical expression and return "2", another might return results from a news database, and so on. To the extent that it is difficult to determine whether an upstream node is forwarding or initiating a query, there is query anonymity. However, due to the direct connection between peers during file transfer, there is no download anonymity.

Studies [21, 36] indicate that Gnutella's search mechanism does not scale. Every query reaches an exponentially increasing number of nodes and generates a prohibitive amount of network bandwidth. There has been some work incorporating hub nodes called reflectors or super peers into the network [23]. These super peers are broadband servers that cache incoming

queries and prevent them from propagating further. This is a flawed solution because it introduces identifiable points of failure into the network, requires a central managing authority, and is vulnerable to the SlashDot effect.

2.4   Freenet Described

Freenet is an adaptable network of nodes that agree to maintain a local data store of information. Files in the system are identified by arbitrarily generated GUIDs[7]. There is no semantic correspondence between a file and its GUID. Nodes forward GUID requests to the downstream neighbor deemed most likely to contain close GUIDs. Closeness, in this case, is not semantic but mathematical with the closeness of GUID i to GUID j simply defined as $|i - j|$. When caching a document with GUID i, a node also stores an associated "reference"—the address of a node likely to specialize in requests for GUID i. A node is said to specialize in i if it contains documents whose GUIDs are close to i and if it has references to other nodes likely to specialize in i. References define the connectivity of the Freenet network. Node i is connected to node j if node i contains a GUID whose reference is node j.

The Freenet routing algorithm can efficiently locate documents in time proportional to $O(\log N)$ where N is the size of the network[8]. Furthermore, Freenet's aggressive caching scheme avoids the SlashDot effect. Since popular data is widely replicated, it does not introduce bottlenecks into the network. Finally, the strictly limited upstream and downstream communication coupled with data encryption protects producer, consumer, and server anonymity.

2.4.1   Requests

If a node has the data corresponding to the requested GUID i, it returns the file to the upstream requester. Otherwise, it identifies the closest GUID to i in its data store and routes the request to the node listed as the reference. When a document is found, it is passed up the requesting chain to the original requester. Each intermediary node caches a local copy and lists the source node as reference. To preserve server anonymity, any node along

---

[7]Currently, an SHA-1 hash is used to calculate the GUID.

[8]Freenet has average case performance of $O(\log N)$. Unlike Chord, it does not make any guarantee as to worst-case performance. In the worst-case, some requests might need to visit every node in the network before locating the requested document [18].
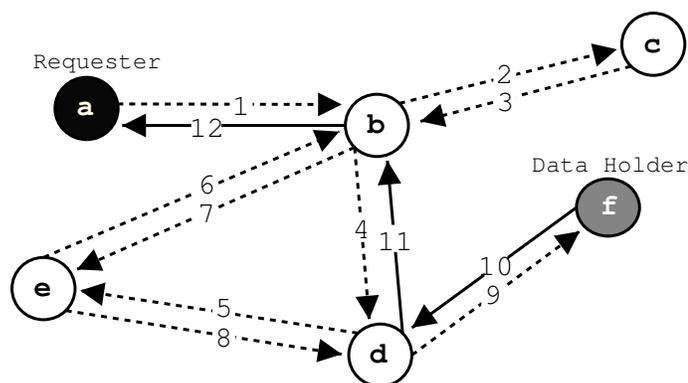
Figure 2.3: Retrieval under the Freenet model

the chain can arbitrarily declare itself as the reference, preventing upstream clients from learning the original location. Note that the more popular a document is, the more widely replicated it will be in the network. Freenet's routing is equivalent to hill-climbing local search.

A unique message identifier allows nodes already in the request chain to recognize loops and inform their upstream requester[9]. In this case, the upstream requester looks up its next closest key and contacts the associated reference node. Should a node exhaust its candidate list, it sends a failure reply to its predecessor. The predecessor then contacts the reference node of its next closest GUID. Nodes continue to forward messages until the document is found or the message's HTL (hops-to-live) expires.

Figure 2.3 describes document retrieval in this model. (1) Node a issues a request for GUID i to node b. (2) Node b does not hold the data, searches its data store for the closest GUID to i and routes to the associated reference, node c. (3) Node c does not hold the data, has no further candidates to route to, and backtracks the request to node b. (4) Node b searches its data store for next closest GUID to i and routes to the associated reference, node d. (5)(6) The request passes from node d to e and back to b without success. (7) Node b detects a loop and backtracks the request to node e. (8) Node e has no further candidates to try and backtracks the request to node d. (9) Node d routes the request to its next closest neighbor, node f, and the data is found. (10)(11)(12) The data is passed up the requesting chain with nodes a, b, and d caching a local copy and listing node f as the

---

[9]Nodes may forget about pending requests after a period of time to keep message memory free.

reference.

2.4.2 Inserts

Inserts in Freenet are analogous to requests. First, the author computes a proposed GUID for the new document by taking its content hash[10]. The author's node then issues a request for the proposed GUID to determine if it is already in use. If there are no collisions (i.e. a duplicate document does not exist), the authoring node sends an insert message which is routed in precisely the same way as the request (including backtracking). Each intermediary node caches a local copy of the new document and lists the authoring node as the reference. Nodes continue to forward an insert message until its hops-to-live expires. To preserve producer anonymity any node in the chain may arbitrarily declare itself as the reference, preventing downstream clients from learning the author's identity. Note that, in the event of a collision when trying to insert a new document, the original file is passed upstream just as in a request (i.e. caching along the chain). Thus, a failed insert still serves to propagate the original file.

The insert mechanism provides a supplementary method by which new nodes can announce themselves to the network. It also improves the network's adaptability by placing documents in nodes expected to store them. Finally, it is difficult for malicious parties to displace popular files with dummy files under the same GUID. The insertion of the dummy will likely fail (because of collision) and serve only to propagate the original file.

2.4.3 Node Announcements

A new node i forms connections into an existing network by starting with a small set of initial GUIDs and associated references obtained through out-of-band means[11]. A node announcement protocol is necessary so that existing nodes become aware of i and form connections to it. Specifically, existing nodes must consistently list i as the reference for some specialization GUID. For node adaptation to occur, existing nodes must list i under the same specialization GUID (see section 2.4.3). For security reasons, node i should not unilaterally decide which GUID it will specialize in. To reconcile the goals of consistency and security Freenet utilizes a collaborative random number generator that any one node cannot influence.

---

[10]This is a simplification of the current Freenet implementation in which there are different types of GUID. See [9] for details.

[11]For example, the Freenet home page or e-mail lists.

A new node i generates a random seed and hashes it to create a commitment. It then forwards an announcement message containing its address and commitment to a node in the network. This node generates a random seed, `XOR`s it with the commitment it received, and then forwards the new commitment to a random reference in its data store. This continues until the HTL of the message expires, at which point each node reveals its random seed. Node i's specialization GUID is taken to be the `XOR` of all the random seeds. Each node adds an entry in its data store for node i under this specialization GUID. Since the final result is an `XOR` of random seeds that can be verified using the final commitment, it is impossible to influence the specialization GUID without control of a large part of the network.

Adaptability

Over time, the network organizes itself to improve its retrieval accuracy. Consider a new node i expected to specialize in GUID j. Since node i will be listed as the reference for j in routing tables, it will mostly receive requests for GUIDs close to j. Initially, i's data store will consist of arbitrary GUIDs and references obtained via out-of-band means. However, with each successful request passing through it, node i will cache another document close to j and store a reference to a downstream neighbor storing at least one document close to j. Thus, adaptability occurs on two levels: nodes cache documents close to their specialization GUID and form connections to other nodes specializing in similar GUIDs.

2.4.4 Data Stores

Aggressive caching in a world of finite disk space necessitates a culling mechanism. Accordingly, Freenet implements its data store structure as a limited stack. Documents that are not requested move to the bottom of the stack and have their data culled. If a document continues to be unpopular, even the GUID and reference are purged. This least-recently-used policy is an explicit trade-off that sacrifices unpopular (and unimportant?) data to ensure the most popular (and important?) data is widely replicated. Freenet makes no guarantee as to how long a file will remain in the system.

All Freenet files are encrypted. Coupled with the fact that GUIDs have no semantic meaning, file encryption permits node operators in a censoring regime to "plausibly deny"[12] any knowledge of their data store contents. Decryption keys are distributed to end users at the same time as the document GUID.

---

[12]The legal strength of this argument is questionable. Node operators could still be found to have vicarious or conspirator liability.

Chapter 3

Protocol and Architecture

FASD draws heavily on Freenet's routing algorithm because of its fully distributed, fault-tolerant, and scalable characteristics. The only other peer-to-peer offering that delivers this combination—Chord's hashtable model—is ill-suited for distributed search (see section 2.2 for details). FASD's approach is to introduce a new type of automatically generated primitive—a metadata key—into the network[1]. Modifying Freenet's routing, caching, and data store algorithms permits FASD to support search on metadata keys while integrating seamlessly into the existing Freenet network. Section 3.8 discusses FASD's extensibility beyond Freenet to other peer-to-peer architectures.

3.1 Metadata Keys

In classic information retrieval, documents are modeled as vectors in the dimensionality of the lexicon[2]. Each entry in the term vector corresponds to the weight of the term in the document. For example, consider a limited lexicon with only three terms—"apples", "bananas", and "oranges". A document's term vector might be $\vec{d} = (13.7, 43, 0)$. $\vec{d}$ suggests that the document is mostly about "bananas", mentions "apples", and does not discuss "oranges". To determine $w_{d,t}$, the weight of term t in document d, FASD adopts a standard measurement known as "TFIDF"[3] [20]:

$$w_{d,t} = \log_2(N/f_t) \log_2(1 + f_{d,t}) \qquad (3.1)$$

---

[1] As section 3.3.1 indicates, metadata keys must be generated before the document is encrypted.

[2] A lexicon is a list of all the unique terms that appear in a document collection.

[3] TFIDF is an acronym for term frequency × inverse document frequency.

where N is the number of documents in the collection, f_t is the number of documents in the collection that contain term t, and $f_{d,t}$ is the frequency of term t in document d. The expression $\log_2(N/f_t)$ in (3.1) captures the "resolving power"[4] of term t [46]. It ensures that common terms will have a lower resolving power than rarer terms (e.g. "the" should have a lower resolving power than "aardvark"). The expression $\log_2(1 + f_{d,t})$ in (3.1) describes the importance of term t in document d. It ensures that a term occurring very frequently in the document will not overpower a less frequent one with higher resolving power.

FASD's metadata keys are based on the vector representation of documents. A metadata key for document d is a list of the nonzero entries in d's term vector. Additionally, each metadata key includes a pointer to the actual document. After receiving metadata keys in response to his query, the user may fetch the actual document from the underlying network using the associated pointer.

Figure 3.1 provides a metadata key for this paper[5]. Based on (3.1), it is not surprising that "fasd", "metadata", "htl", etc. are weighted so heavily—these terms occur frequently in this paper and have a high resolving power (because they occur infrequently in the English language). The sample metadata key in the figure is designed for a FASD network operating on top of Freenet. As such, the pointer component consists of a GUID and a decryption key. If a user wished to retrieve the actual text of this paper, he would issue a Freenet request for GUID 89456 and decrypt the data with the cipher key 18234.

3.2 Requests

3.2.1 Metadata Closeness

Clarke's initial design of Freenet abstracts itself from any particular key implementation [8]. As long as "closeness" can be defined for a given key type, Freenet's adaptive routing approach should function. In Freenet, a concept of closeness is necessary to determine which of GUID a or b is closer (or equal) to the requested GUID c. Similarly, in FASD, a concept of closeness is necessary to compare metadata keys with one another and with queries. A consistent closeness operator should send queries to nodes

---

[4]The resolving power of a term is its usefulness in helping distinguish one document from another. It is also referred to as the term's discrimination value.

[5]The figure contains only the top ten entries of the metadata key. This paper contains over 14,000 words and 13,000 unique words.

GUID : 89456
Decryption Key : 18234

fasd	122.44
metadata	67.63
htl	67.21
napster	54.49
gnutella	50.99
freenet	48.14
peer-to-peer	42.58
small-world	42.17
freegle	36.32
$\vdots$	$\vdots$

Figure 3.1: A metadata key for this paper

that contain metadata keys matching the query. It should also route new metadata keys to nodes expected to store them.

Freenet mathematically defines closeness of two GUIDs as their absolute difference. Given the multi-dimensional nature of a term vector, a more sophisticated closeness operator is necessary for metadata keys and queries. FASD adopts a standard measure known as the "cosine correlation value" [44]:

$$\cos \vec{d_i}, \vec{d_j} = \frac{\vec{d_i} \cdot \vec{d_j}}{\|\vec{d_i}\|\|\vec{d_j}\|} = \frac{\sum_{k=1}^{t}(t_{i,k} \cdot t_{j,k})}{\sqrt{\sum_{k=1}^{t}(t_{i,k})^2 \cdot \sum_{k=1}^{t}(t_{j,k})^2}} \qquad (3.2)$$

where $\vec{d_i}, \vec{d_j}$ are the term vectors being compared and $t_{i,k}, t_{j,k}$ describe the weight of term k in the respective vector. The cosine correlation describes the angle between document vectors in a t-dimensional space where t is the number of entries in the lexicon (i.e. the size of the vocabulary). Figure 3.2 depicts document vectors in a space of 3 terms. The weights of the terms in the vector determine its magnitude. In the figure, $\vec{d_2}$ is more similar to $\vec{d_1}$ than $\vec{d_3}$ because $\theta(\vec{d_1}, \vec{d_2}) < \theta(\vec{d_1}, \vec{d_3})$. The cosine correlation utilizes the angle between documents so as to normalize against longer documents being incorrectly favored.
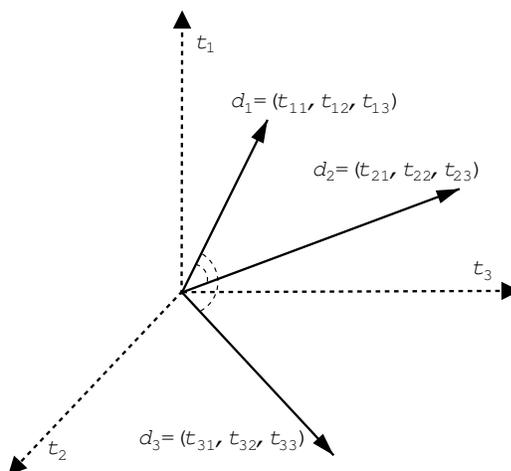
Figure 3.2: A document space in 3 terms

Vector representation of queries

Simple queries in the form $q = t_0$ `AND` t_1 `AND` ... `AND` t_m may also be abstracted into term vectors. q's term vector is obtained by setting the weight of q's terms to be 1. Thus, given the sample lexicon: "apples", "bananas", and "oranges", the vector representation of the query "apples `AND` oranges" would be $\vec{q} = (1, 0, 1)$. By abstracting queries into vectors, FASD uses the closeness operator in (3.2) for comparing metadata keys to one another and for comparing metadata keys to queries. Note that substituting q's term vector $\vec{q}$ for $\vec{d_j}$ in (3.2) gives $\vec{d_i}$'s "score" or "similarity" with respect to q. Section 3.2.2 discusses how FASD processes complex queries.

Thus, (3.1) provides FASD with the framework to route queries and metadata keys. Given two metadata keys i and j, FASD determines which is closer to a given query (or metadata key) q by comparing the cosine correlation measure of i's term vector and q's term vector and of j's term vector and q's term vector. The larger the cosine correlation value, the more closely associated is the information content of the given term vectors. Equality occurs when the cosine correlation of two metadata keys is 1 and the pointers are identical. The stricter definition of equality is necessary in the event that two documents are very similar but not identical[6].

---

[6]This might occur due to the use of stemming algorithms [44], stop-word removal [44], and culling of the least distinguishing terms.

3.2.2 Query Routing

In responding to user queries, FASD needs to return a ranked list of the top n hits (i.e. the n metadata keys whose vector representation is closest to the query's vector representation). Freenet's routing protocol has minimal support for the sophisticated caching and node-to-node communication necessary to collaboratively locate hits and transfer them to the end user. FASD utilizes a routing algorithm, PROCESS-QUERY, that forwards queries downstream and returns multiple hits upstream while addressing bandwidth and security concerns.

PROCESS-QUERY(q, $bestScores$)
  1: **if** SEEN($thisMessage$) **then**
  2:     **return** NIL {Notify upstream node of loop}
  3: **end if**
  4: $results \leftarrow$ GET-TOP-HITS(q)
  5: $oldScores \leftarrow bestScores$
  6: UPDATE-SCORES($bestScores$)
  7: $i \leftarrow 0$
  8: $candidateNode \leftarrow$ NIL
  9: $addedResults \leftarrow$ NIL
 10: **while** $HTL \neq 0$ **do**
 11:     **if** $i \geq results.length$ **then**
 12:         **break** {Ran out of candidates, backtrack upstream}
 13:     **end if**
 14:     $candidateNode \leftarrow results[i].refNode$
 15:     $addedResults \leftarrow candidateNode.$PROCESS-QUERY($q$, $bestScores$)
 16:     UPDATE-RESULTS($results$, $addedResults$)
 17:     UPDATE-SCORES($bestScores$)
 18:     $i \leftarrow i + 1$
 19: **end while**
 20: CACHE(RESULTS)
 21: **return** FILTER-RESULTS($oldScores$, $results$)
*Note that $candidateNode.$PROCESS-QUERY(q, $bestScores$) is a remote procedure call.

The update functions used in PROCESS-QUERY preserve two invariants. UPDATE-RESULTS guarantees that $results$ is a list of the top n metadata keys encountered at, or deeper than, the current depth in the request chain. UPDATE-SCORES ensures that $bestScores$ is a list of the top n scores encountered at any point in the request chain. Thresholding the result set

is necessary because a short query might generate thousands of hits. It is justified to the extent that 85% of web users are unwilling to examine more than the first few tens of results [6].

On receiving a query q a node i invokes the GET-TOP-HITS function which returns a set *results* of the n metadata keys in i's store closest to the vector representation of q. *results* is sorted in descending order such that the top hit will appear at the top of the list. i will forward q based on the references[7] of the metadata keys in *results*. The reference of the top hit is the best candidate, the reference of the second hit is the second-best candidate, and so on. After calling its update functions, node i passes the query and the current list of best scores to the best downstream candidate. When the downstream node returns the request to node i, i re-invokes its update functions and forwards q along with the the current list of best scores to the second-best candidate. As with regular Freenet requests, forwarding continues until the HTL expires. Unlike a Freenet request, a FASD request will never complete before the HTL expires because it is impossible to determine whether the current set of results is globally optimal.

Prior to returning upstream, node i invokes FILTER-RESULTS to ensure that only metadata keys with scores greater than those in *oldScores* are transferred. FASD's use of FILTER-RESULTS reduces bandwidth consumption significantly and prevents upstream nodes from caching metadata keys that only weakly satisfy the query. Node i may also arbitrarily declare itself as the reference for any of the keys it passes upstream so as to protect the server anonymity of downstream nodes.

A unique message identifier allows nodes to detect loops in the request chain and notify the upstream requester that it should contact a different candidate[8]. Should a node run out of candidate nodes, it backtracks the request up to its predecessor. Nodes store a local copy of all metadata keys passed upstream in a least-recently-used cache. Since a node cannot determine whether an upstream requester is initiating or forwarding a query, the routing mechanism preserves query anonymity. The retrieval of the actual document occurs via the standard Freenet request mechanism using the GUID and decryption key given in the metadata key. Thus, to the extent that Freenet does so, FASD also preserves download anonymity.

---

[7]Recall that reference for metadata key i is the address of a a node likely to specialize in requests for i.

[8]Nodes may forget about pending requests after a period of time to keep message memory free.

Complex Queries

FASD's algorithm is easily extended to complex queries. When processing a complex query, FASD breaks it up into a disjunction of conjunctions. For example, converting the query $q = ((t_1 \text{ OR } t_2) \text{ AND } t_3)$ into a disjunction of conjunctions yields $q = ((t_1 \text{ AND } t_3) \text{ OR } (t_2 \text{ AND } t_3))$. FASD draws on standard techniques from first-order logic to convert any complex query into a disjunction of conjunctions [39].

To understand how FASD routes queries in this form, consider the query $q = (q_1 \text{ OR } q_2)$ where

$$
\begin{aligned}
q_1 &= (t_1 \text{ AND } t_2 \text{ NOT } t_3) \\
q_2 &= (t_4 \text{ AND } t_5)
\end{aligned}
$$

To process q the user's node would issue separate FASD queries corresponding to the two conjunctive clauses q_1 and q_2. Let R_1 and R_2 represent the result sets for q_1 and q_2 respectively. The final result set for q is simply $(R_1 \cup R_2)$. FASD routes q_1 and q_2 independently because there there is no guarantee that a node specializing in q_1 will also specialize in q_2. In processing q_1, GET-TOP-HITS retrieves all metadata keys that match the query "t_1 AND t_2" and then removes any keys that contain the NOTed term t_3.

Performance Optimization

Latency in distributed applications is proportional to the number of hops a message consumes before returning to the requester. Metadata routing requires an explicit trade-off between performance and recall (section 4.1.2 defines these evaluation criteria). Setting a larger HTL helps ensure that the closest metadata keys are located while a lower HTL decreases response time. Although this paper concerns itself with an upper bound on FASD's latency, a number of routing optimizations are possible. For example, iterative deepening [40] could reduce response time without sacrificing quality. In this model, the initiating node first sends out a shallow (small HTL) query and presents the results to the user. As the user peruses these results, his node sends out a deeper request to extract hits that might be further downstream. To save bandwidth and ensure that only better metadata keys are returned, the deeper request includes the scores from the shallow request. A production level roll-out of FASD should certainly concern itself with further optimizations to reduce latency and bandwidth consumption.

3.3 Inserts

3.3.1 Key Generation

When an author inserts a new document d into the Freenet system, his node must automatically generate a corresponding metadata key. Key generation occurs on document insert because this is the only time the document exists in unencrypted form. Note that the author does not manually enter any information during the key generation process. Automatic generation improves usability and ensures that all documents are described consistently across the FASD layer.

Calculating the weight $w_{d,t}$ of each term t in document d as per (3.1) is prohibitively expensive. It would require retrieving and processing all the (encrypted) documents in the Freenet system to determine the discrimination value of each term t in d. FASD circumvents this issue by precomputing the resolving power of each term in the lexicon of a representative[9] document collection[10]. When a FASD network is first created, the standard Freenet insert mechanism is used to distribute the precomputed lexicon. Given the lexicon's expected popularity and the nature of Freenet's insert mechanism (see section 2.4.2), it is highly unlikely that a malicious party could successfully insert a "dummy" document under the same GUID as the lexicon. Before generating a metadata key, an authoring node will request the lexicon file[11]. It then parses document d and generates its metadata key. The precomputed resolving powers from the lexicon are substituted into (3.1).

In order to avoid artificial differences between documents that would diminish the accuracy of (3.2), a given term's resolving power must be consistent across the network and over time. Thus, once a lexicon with precomputed discrimination values has been distributed into the system, it can no longer be altered. A static lexicon is an acceptable restriction because the slow evolutionary rate of language suggests the resolving power of a given term is unlikely to change dramatically over time. If a document d contains a term t that is not in the lexicon (e.g. an obscure name or a new acronym), t is arbitrarily assigned the largest possible resolving power[12] be-

---

[9]Given a sufficiently large representative document collection, the estimated discrimination values should be accurate.

[10]The FASD simulation uses the Online Book Initiative's 600 MB document collection consisting of everything from web pages to recipes to news articles to complete texts [31].

[11]Since the lexicon is requested at every document insert, it will be widely replicated. Thus, it should be stored at the author's node or within a minimal number of hops.

[12]Specifically, the resolving power of term t not in the lexicon is taken to be $\log_2 N$ where N is the size of the representative document collection. This value is based on (3.1) and the assumption that f_t, the number of documents in the collection containing t, is 1.

cause it is probably very useful in differentiating d from other documents in the collection.

There is a risk that the lexicon may consume too large a portion of nodes' data stores. In specific, a multilingual document collection such as Freenet's contains a far greater number of unique terms than a unilingual one. FASD's solution is to segment the document space by language. In lieu of one large multilingual lexicon, FASD utilizes smaller unilingual ones based on representative document collections in the corresponding language. Authors retrieve the appropriate language's lexicon when generating a metadata key. In this model, requesters indicate the language of their query to ensure that only keys generated from the relevant lexicon are compared and returned. The GUID for each language's lexicon is explicitly specified in the protocol to ensure universal access. If further compression on a lexicon is necessary, the terms with the lowest discrimination values may be dropped. Although preliminary tests suggest that dropping the least discriminating terms from the lexicon does not result in any significant degradation of result quality, further research is necessary.

3.3.2   Insertion Routing

After generating the metadata for document d, an author invokes a Freenet insert (see section 2.4.2) of d. If the insert is successful (i.e. another document with d's GUID key does not exist), then d's metadata key is inserted. Routing is provided by the PROCESS-QUERY function using the query $q = t_1, t_2, \ldots, t_n$ where t_i is the ith entry in d's metadata key and n is the total number of terms in d's metadata key. Each node in the request chain caches a local copy and lists the authoring node as the reference. To preserve producer anonymity any node in the chain may unilaterally declare itself as the reference. The insertion scheme has three important benefits. Most importantly, Freenet metadata keys are inserted at different locations from the documents they describe, mitigating the high security risk that unencrypted metadata keys pose. An attack targeting metadata keys will, at worst, cripple FASD and reduce the augmented Freenet network to *status quo ante* (i.e. the integrity of GUID requests and inserts is maintained). Furthermore, using PROCESS-QUERY accelerates the network's adaptability by placing a new metadata key k in nodes expected to specialize in queries close to k. Finally, the insertion mechanism provides an additional means for authoring nodes to announce their presence to existing nodes.

3.4 Metadata Stores

Metadata keys and their references are stored in a stack data structure. Keys that are frequently accessed (i.e. regularly passed upstream in response to queries) remain at the top of the stack while less popular keys move to the bottom and are ultimately culled.

3.4.1 Inverted Indices for Efficient Search

In order for a node j to route a query (or metadata key) q appropriately it must determine the cosine correlation value of q and each metadata key k_i in its store. Directly computing (3.2) would require time proportional to the cumulative length of all metadata keys in the data store. Accordingly, each FASD node maintains a supplementary inverted index to permit efficient query processing. An entry in the index for a given term t has the form:

$$< t; [(k_1, w_{k_1,t}), (k_2, w_{k_2,t}), \ldots, (k_m, w_{k_m,t})] > \tag{3.3}$$

where k_i is a pointer to the ith metadata key containing t, $w_{k_i,t}$ is the weight of t in k_i's vector representation, and m is the total number of metadata keys in j's store containing t. Standard algorithms for search on an inverted index that execute in time proportional to the query length are given in [45].

Since nodes cache and cull metadata keys very frequently, the inverted index must support efficient add/delete operations. Although efficient dynamic update is not possible in large search engines [19], it is possible in FASD because of the relatively small size of any one node's metadata store. FASD maintains each inverted index entry as a sorted list. Thus, insertion and deletion of a metadata key k with $\|k\|$ terms occurs in $O(\|k\| \log(\|k\|))$ time. Further optimization is possible using batch processing that only executes when a node is idle.

3.4.2 Maintaining Consistency

The dynamic nature of metadata and data storage may give rise to two possible inconsistencies in the network vis-à-vis a document d and its metadata key k:

1. k may be purged while d remains

2. d may be purged while k remains

The first inconsistency is remedied by requiring any direct requester of d to periodically re-generate k and re-insert it into the metadata layer. A direct requester is a user who accesses d without going through the FASD engine (i.e. learns d's GUID key through out-of-band-means or follows a link to d). Periodic re-generation and re-insertion offers a reliable guarantee that the metadata keys of frequently accessed documents will exist in the FASD network.

FASD remedies the second inconsistency (essentially a "HTTP 404: File Not Found" error) using "cull requests". If a node n suspects that d no longer exists (i.e. d was not retrievable within a reasonable number of hops), it notifies downstream nodes to cull k. The request is routed using the query $q = t_1, t_2, \ldots, t_m$ where t_i is the ith term in k and m is the total number of terms in k. On receiving a cull request, a node first issues a request for d to verify the legitimacy of the request. If d is found, the downstream node returns an "illegitimate cull" error upstream. Otherwise, the cull-request is considered legitimate and routed downstream. If all nodes in the chain agree that the cull is legitimate, an "all clear" is sounded and k is removed from each node's metadata store. The metadata cull mechanism enhances FASD's security and adaptability. An illegitimate cull by a malicious node attempting to censor k not only fails to remove k from any downstream data stores but also causes further replication of d. An illegitimate cull issued by a trustworthy node n will help bring the desired document d within n's search horizon.

3.5 Metadata Security

FASD's lack of encryption coupled with its unsophisticated ranking techniques gives rise to a number of difficult security issues. This section identifies these issues, provides *ad hoc* solutions, and suggests directions for future research.

3.5.1 Avoiding Censorship

PROCESS-QUERY provides some protection against query censorship. Assume that a malicious node m wishes to censor a certain query q. Since queries are unencrypted, node m can detect when it receives a request containing q. At this point, the most damage node m can inflict is to allow q to propagate downstream until its HTL expires and then refuse to pass any results upstream. m cannot dilute the results already found by upstream nodes because UPDATE-SCORES ensures that the final result set consists of

the top n results encountered at *any* point in the request chain. Furthermore, should the end-user suspect censorship, he could attempt to circumvent node m altogether. In lieu of routing q to the reference associated with its best hit, his node would route q to the reference associated with its second-best hit. The user's node could continue to iterate through this process (i.e. routing to the reference associated with the third-best hit and so on) until the user's information need was satisfied. In bypassing the censoring node and passing relevant results back to the user, this circumvention process leads to wider replication of keys close to the censored query. Thus, in attempting to censor q, node m might inadvertently cause metadata keys satisfying q to become more widely replicated.

Further investigation into the applicability of cryptographic search algorithms [25, 49] to a more robust metadata routing framework is necessary. These techniques might enable a scenario in which an untrusted node computes the similarity of an encrypted query to an encrypted metadata key without ever gaining knowledge of the semantic meaning. An encrypted approach would curtail censorship and protect node owners with a plausible deniability clause *vis-à-vis* the contents of their metadata stores. Unfortunately, it is not obvious how to update an encrypted inverted index as encrypted metadata keys are added and removed [49]. Currently, the techniques in [49] only work in situations where Alice is searching through data that she herself has encrypted and stored with Bob. In a distributed environment, this paradigm poses a significant "chicken and egg" problem: the querying user must somehow gain knowledge of the keys used to encrypt the hits to his query.

3.5.2 Closeness versus Quality

There is a distinction between the quality of a result and its closeness to the query q in document space [11, 16, 42]. Closeness is objectively defined by (3.2) whereas quality is a subjective measure of a result's ultimate utility to the end user. Consider a document d comprised entirely of n instances of the term t. d will be extremely close to the query $q = t$ but of minimal utility to the end-user. A search technique based entirely on (3.2) does not adequately incorporate other indicators of quality. Such indicators include reputation of the document source, update frequency, popularity or usage, and citation [6, 7, 17, 34]. The Google search engine very successfully adopts these techniques. In particular, its PageRank algorithm examines the hyperlink structure of the web to determine the authoritative value of a given site [6].

From a security perspective, improved quality is critical to protecting against attacks on the distributed search mechanism. Even the strongest encryption techniques cannot prevent a malicious user from censoring query q by flooding the network with illegitimate documents whose metadata keys are highly relevant to q. Insertion of dummy data to dilute the quality of returned results is essentially a denial of service attack on the search engine.

Unfortunately, the anonymous and encrypted nature of Freenet makes it difficult to incorporate non-query dependent factors into the search mechanism. Although there is a form of linking between documents, it is not obvious how to apply a PageRank heuristic. Doing so would require a crawling agent that could discover the hyperlink structure of encrypted data. A possible approach could separately encrypt a document's link structure with a public key whose private pair was known only to the crawler. Further research is necessary into reconciling the goals of decentralization, fault-tolerance, and anonymity with the requirements of a network crawling agent capable of analyzing document link structure and/or other factors suggestive of quality. Only then can FASD make informed decisions that will ensure results are of high utility to the end-user.

3.6   Node Announcements

In order for a new node i to gain connectivity to the metadata layer, existing nodes must form connections to i and i must form connections to existing nodes. FASD's node announcement protocol achieves both of these objectives.

Existing nodes form connections to node i by consistently listing i as the reference for some specialization metadata key. For node adaptation to occur, existing nodes must list i under the same specialization metadata key (see section 3.7). For security reasons, node i should not unilaterally decide which metadata key it will specialize in. To reconcile the goals of consistency and security, FASD leverages Freenet's existing announcement mechanism. After a specialization GUID key has been selected for node i, an existing node j is randomly selected by taking the specialization GUID modulo the number of nodes in the announcement chain[13]. Node j then randomly selects a metadata key from its data store. All the nodes in the announcement chain store this metadata key and list node i as the reference. It is impossible for a malicious party to influence which node donates the

---

[13]The nodes involved in the announcement process are said to form an "announcement chain".

specialization metadata key because the selection process is based on a GUID guaranteed to be random (see section 2.4.3).

Node i forms connections into the existing network by storing a reference to each of the nodes in the announcement chain. Specifically, each existing node sends i a metadata key randomly selected from its store. i caches these metadata keys and lists the corresponding donor node as a reference.

3.7 Adaptability

An argument analogous to that in section 2.4.3 demonstrates that adaptability also holds for metadata routing. Consider a new node i expected to specialize in metadata key k. As per the announcement protocol, node i will be listed as the reference for k in routing tables. Accordingly, it will receive many queries to which k is a good hit (i.e. the cosine correlation between k's term vector and the query's term vector is large). Initially, i's data store will consist of arbitrary metadata keys and references obtained in the announcement process. However, with each query passing through it, node i will cache any top hits passed back by downstream nodes and list the corresponding downstream nodes as references. Thus, adaptability occurs on two levels: nodes cache metadata keys close to their specialization metadata key and form connections to other nodes specializing in similar metadata keys.

Also note a second level of adaptability. Numerous nodes will spawn requests for popular metadata keys causing them to be widely cached throughout the network. Thus, FASD organizes itself to reduce the latency of popular queries by ensuring that the most relevant hits to popular queries will be within a minimal number of hops. As in Freenet, there is an explicit trade-off whereby unpopular (and unimportant?) metadata key may be culled to ensure that the most popular (and important?) metadata keys are widely replicated.

3.8 Extensibility Beyond Freenet

Although designed for the Freenet system, FASD's framework has broader applicability. FASD can be viewed as a search layer that integrates into a wide array of distributed applications. This section briefly explores two possibilities that hopefully suggest FASD's value to the greater peer-to-peer community.

3.8.1 Applying FASD to Chord

Although extremely impressive, Chord's $O(\log N)$ LOOKUP primitive does not support search. Augmenting a Chord network with FASD would significantly enhance the network's utility without sacrificing distribution, fault-tolerance, or scalability. Each node in this hybrid system would maintain an auxiliary metadata store and route searches as per PROCESS-QUERY. On receiving hits to his query, the user would call LOOKUP(*pointer*) to retrieve the actual documents.

Since Chord does not focus on anonymity, nodes could maintain knowledge of the entire request chain. In addition to passing results upstream, node i could also transfer directly to the initial requester. Doing so would dramatically reduce latency while retaining the adaptive benefits of caching along the request chain. Furthermore, node i could detect loops *a priori*, eliminating the need for the additional hops necessary to do so in Freenet's opaque environment.

3.8.2 FASD as a Stand-alone Application

More generally, FASD could leverage TCP/IP to become a stand-alone search application that sat on top of an Intranet or the Internet. In this model, every participant maintains a dynamic metadata store on his node. When an author decides to share a document d, his node automatically generates a FASD metadata key describing d and inserts it into the FASD search network. Instead of identifying d by a Freenet GUID and decryption key, FASD would do so by IP address and filename. Search proceeds as normal with nodes collaboratively locating and returning a list of metadata keys close to the query. Users would retrieve the actual documents directly from the author's node via TCP/IP. In the event that multiple authors decide to share an identical file[14], metadata keys could contain a list of pointers describing all the file's locations.

---

[14]This is frequently the case with music and video files.

Chapter 4

Simulation and Experimental Results

Simulations of a FASD network indicate promising results across all axes: performance, scalability, recall, and fault-tolerance. This chapter describes the discrete event simulator used to model the network and presents an analysis of the results.

4.1 Simulator Details

The FASD simulator[1] is implemented as a single-threaded, event-driven application. Test scripts spawn external events—metadata inserts/requests and node creation/announcement—that generate any number of internal events—adding/culling metadata keys to/from stores, search of inverted indices, and node communication. Recursive calls to PROCESS-QUERY route messages as per the FASD protocol.

On startup, the simulator randomly[2] selects 2500 documents from the OBI's (Online Book Initiative) collection of documents and generates their metadata key. It then inserts each of these metadata keys into a global inverted index. For storage and computational efficiency, nodes do not construct individual inverted indices. Instead, queries are processed against the global inverted index with each node filtering out hits not currently in

---

[1] Parts of the simulation are inspired by Ian Clarke's JAVA adaptive network client released under the GNU public license [14].

[2] Due to limited memory resources documents larger than 4 KB are not used. The OBI's collection consists of everything from web pages to recipes to news articles to complete texts [31].

its metadata store. In order to emulate normal usage of a FASD network, the simulator generates queries in the form $q = t_1$ `AND` ... `AND` t_5 where term t_i is selected at random from the inserted metadata keys. Somewhat arbitrarily[3], each node's data store has a capacity of 200 metadata keys.

4.1.1 Bootstrapping

In any network simulation, there is a question of how to bootstrap connectivity. Abstractly, a FASD network is a directed graph with connectivity given by the contents of nodes' metadata stores. Specifically, if node i contains a key whose reference is node j, then the graph contains the directed edge (i, j). FASD's aggressive caching leads to a dynamic topology that constantly changes with metadata key inserts, queries, and node announcements. To permit a new network to adapt and grow, the bootstrapping procedure must ensure connectivity—there must be a directed path connecting every pair of vertices in the network's graph representation[4]. Furthermore, the bootstrapping procedure must emulate the node announcement protocol whereby nodes are initially referred to under a consistent specialization metadata key selected at random. This initial consistency in how nodes are referenced is imperative to node adaptation (see section 3.7).

As figure 4.1 describes, the simulator uses a regular ring-lattice topology and a hash function to achieve the goals of connectivity and consistency. It arbitrarily assigns each node i a specialization metadata key given by $k_{hash(i)}$. In order to create the directed edge (j,i) the simulator places $k_{hash(i)}$ in node j's metadata store and lists i as the reference. A regular ring-lattice topology is achieved by creating the directed edges $(i, i-1)$ and $(i, i + 1)$ for every vertex i. Vertex 0 is connected to vertex $n - 1$ and vertex $n - 1$ to vertex 0 where n is the number of nodes in the network.

4.1.2 Primary Metrics

Efficiency

Consider a given metadata key request q that travels h hops. Assume q is routed downstream without any backtracking and that the hth node provides

---

[3]Whenever possible, simulation runs attempt to emulate the parameters in [18].

[4]In the course of metadata key caching and culling, there is a risk that the network will fragment into disconnected components. As a scale-free network, (see section 4.6) FASD is very resistant to fragmentation unless exposed to targeted attack [2]. Nevertheless, finding theoretical guarantees against fragmentation is an important avenue for future research.
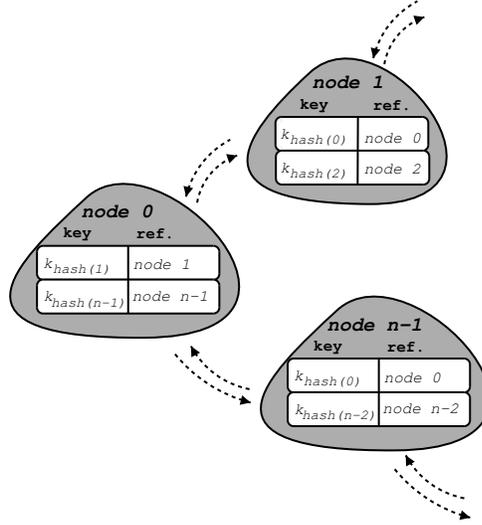
Figure 4.1: Bootstrapping in a lattice topology

all the n top hits. The approximate bandwidth consumption is:

$$h \cdot (\text{SIZE}(q_{down}) + \text{SIZE}(q_{up})) \qquad (4.1)$$

where $\text{SIZE}(q_{down})$ and $\text{SIZE}(q_{up})$ denote the average size of q as it passes downstream and upstream respectively. The approximate latency is:

$$h \cdot (\text{TURNOVER}(q_{down}) + \text{TURNOVER}(q_{up})) \qquad (4.2)$$

where $\text{TURNOVER}(q_{down})$ and $\text{TURNOVER}(q_{up})$ represent the average amount of time a node spends processing q as it passes downstream and upstream respectively. Bandwidth consumption and latency are indicators of network performance: larger bandwidth consumption and latency leads to poorer performance while smaller bandwidth consumption and latency leads to improved performance. As (4.1) and (4.2) indicate, these metrics are proportional to the number of hops a query takes (referred to as the query's pathlength). Thus, by noting the pathlength of queries in a FASD network the simulator can measure network performance.

Effectiveness

There are two primary measures used in the evaluation of information retrieval systems [42]. Recall measures the ability of the system to present

all "relevant" items, while precision measures the ability to identify and reject "irrelevant" items. A classic dilemma in information retrieval is the definition of "relevance" in the context of subjective user feedback. For the purposes of the simulation, relevance of a metadata key vis-à-vis a given query is naïvely taken as the cosine correlation value. Although an important direction for future research, a more sophisticated definition of relevance is outside this paper's focus on developing a fault-tolerant, scalable, and completely decentralized search framework.

The exclusive use of cosine correlation value as the measure of relevance removes any need for subjective intervention. Given a query q, spawn a search for q from a node in the FASD network and label the resulting set R_F. Then determine the results that a centralized search engine would return in response to q (i.e. use the global inverted index). Label this result set R_G and remove from R_G any metadata keys not yet inserted into the FASD network. Finally, ensure $\|R_G\| \leq n$ by culling the items with the lowest cosine correlation value. Recall that n (set at 10 for the simulation) is the number of hits that PROCESS-QUERY will pass upstream. Within this framework, recall and precision are as follows:

$$\text{Recall} \quad = \quad \frac{\|R_F \cap R_G\|}{\|R_G\|} \tag{4.3}$$

$$\text{Precision} \quad = \quad \frac{\|R_F \cap R_G\|}{\|R_F\|} \tag{4.4}$$

If the top n hits returned from FASD are precisely the top ten hits returned from the centralized search then $\|R_F \cap R_G\| = \|R_G\| = \|R_F\| = n$. In this case, recall and precision as given by (4.3) and (4.4) will both be 100%. Thus, 100% recall and precision indicate that FASD's distributed approach was as effective as its centralized counterpart (search on an inverted index containing the entire collection of inserted documents).

Reconciling efficiency and effectiveness

In FASD, as in any other information retrieval system, there is a trade-off between effectiveness (recall and precision) and efficiency (latency and bandwidth consumption). Utilizing a larger HTL for queries will ensure higher recall and relevance at the expense of decreased performance. The simulator concerns itself with an upper bound on latency and bandwidth consumption by examining the number of hops necessary for a request to retrieve the same top n results that centralized search would return.

Every 200 time-steps, the simulator spawns 300 random queries from randomly selected nodes. For each query, it records the number of hops necessary to retrieve the same top n results that centralized search would return. During this poll period, the network's topology is frozen—no data is cached or culled. The HTL of poll queries is set at 500. If a poll query still fails to retrieve the same top n results as centralized search, it is treated as taking 500 hops.

4.2   Growth

Simulating growth incorporates all the components of the FASD protocol—node announcements, query requests, and metadata key inserts. Initially, there is a connected core of 20 nodes. At every time-step, the script selects a random node i in the network. Node i randomly decides to issue an insert of a randomly selected metadata key or a request for a randomly generated query. Every 5 time-steps, a new node enters the network and announces itself to a randomly selected existing node. This process continues until the network reaches a size of 1,000 nodes. The HTL for announcements and inserts/requests is set at 10 and 20 respectively.

As described in section 4.1.2, the simulator polls the network every 200 time-steps to determine network performance. During this poll period the network is frozen and 300 requests are spawned from random nodes. The simulator observes how many hops each of these requests takes before it retrieves the same top n results as centralized search.

Figure 4.2 plots request pathlength as a function of network size. FASD exhibits excellent average-case performance—in under 20 hops it retrieves the same top n results as centralized search engine. The top quarter of requests locate the top hits with only 4 hops. As the plot of the third quartile indicates, there is poor worst-case retrieval with some requests taking as many as 100 hops to locate all the results. The histogram in figure 4.3 depicts the distribution of all request pathlengths at the end of the simulation. Using iterative deepening (see section 4.5) can help mitigate the worst-case performance by retrieving most of the top hits in a shallow request (hops-to-live of 20) with the flexibility to deepen the search if desired.

One might object that the simulation does not realistically model the rate of change of growth. In [48], Shapiro and Varian argue that networks are subject to positive externalities. As a network's size increases, the rate of growth will also increase because the value of the network is proportional
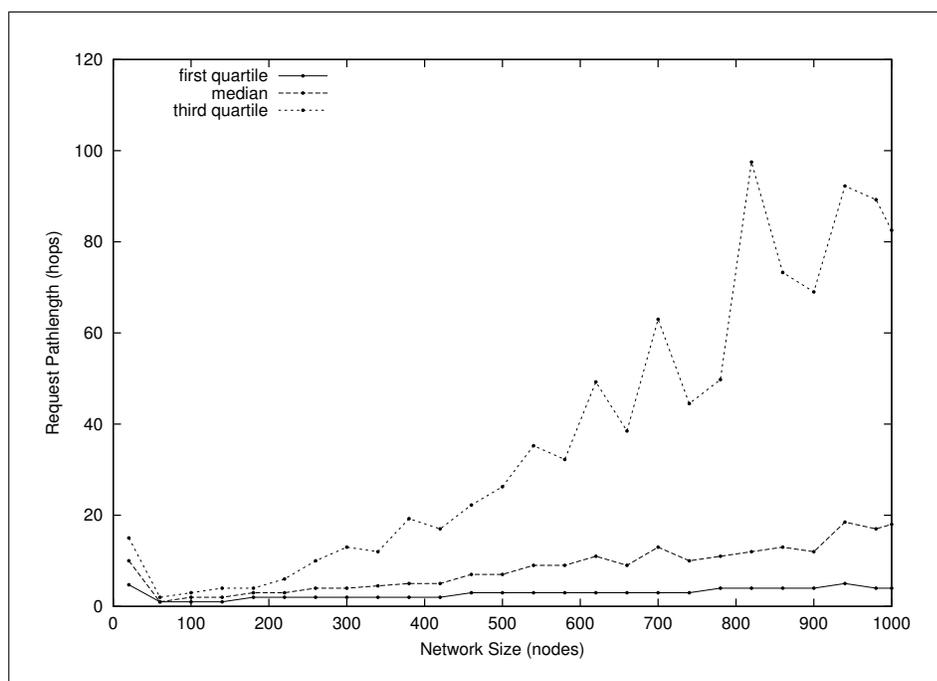
Figure 4.2: Request pathlength versus network size. The request pathlength is the number of hops necessary to retrieve the same top n results as centralized search. See section 4.2 for details.

to its size[5]. However, as [18] points out, the number of requests in the network will also be proportional to size. Since the number of requests issued is also held constant, it is valid to assume a constant rate of growth. If anything, the model may be overly demanding in that it does not allow for the period of steady state convergence that network economics predicts. The adoption cycle for a new network typically includes a slower rate of growth at launch, followed by an increase rate during takeoff, and then a leveling off as saturation occurs [48]. As the results in section 4.4 suggest, a steady state convergence period would permit the network to further adapt and, as such, would further reduce the number of hops necessary for requests to retrieve the same top n results as centralized search. To the extent that the simulation focuses on an upper bound for latency and bandwidth, the lack of a steady state convergence period is justifiable.

---

[5]Metcalfe's Law suggests that the value of a network increases as the square of the number of users.
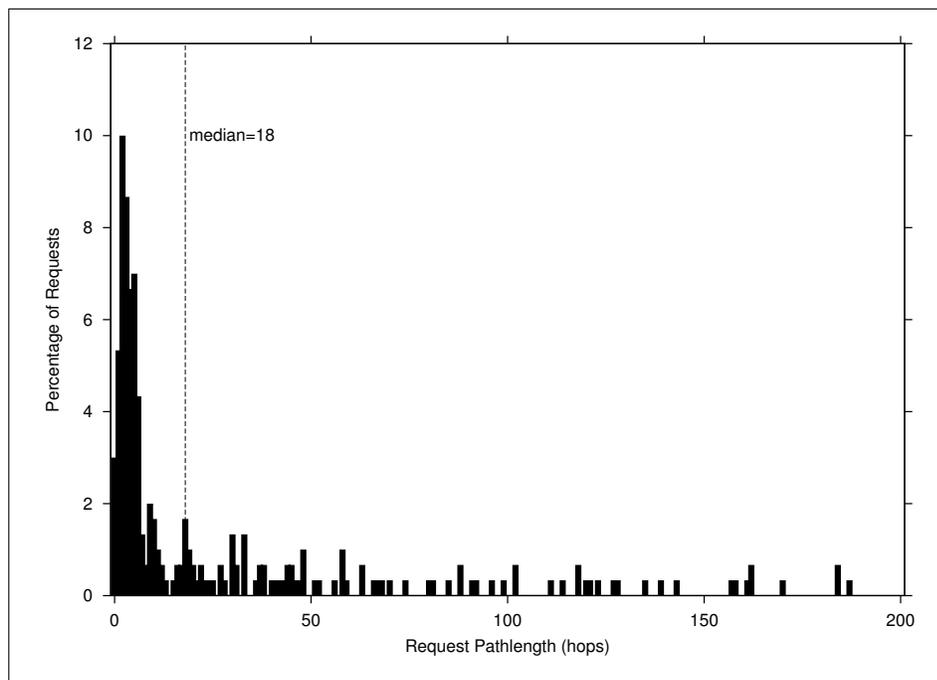
Figure 4.3: Distribution of all request pathlengths at the end of the simulation. The median number of hops necessary to retrieve the same top n results as a centralized search is 18. See section 4.2 for details.

Another plausible objection might question the random usage pattern in the simulation whereby the probability of issuing a given query is uniformly distributed. Random usage is a questionable model given several studies' conclusion that web access follows a Zipf-like distribution [3, 5, 12]. In order to model a Zipf-like usage pattern, the simulator should arbitrarily assign a popularity to each possible query. The relative probability of issuing the ith most popular query is proportional to $1/i^\alpha$ with α taking on some value less than unity. FASD's bias towards wider replication of popular metadata keys suggests that performance would only improve under Zipf-like usage. Thus, a limited time-frame and this paper's focus on upper bound latency and bandwidth consumption justify a uniform usage pattern. Future simulation should certainly incorporate more realistic usage scenarios to confirm FASD's ability to very efficiently service the most popular queries.
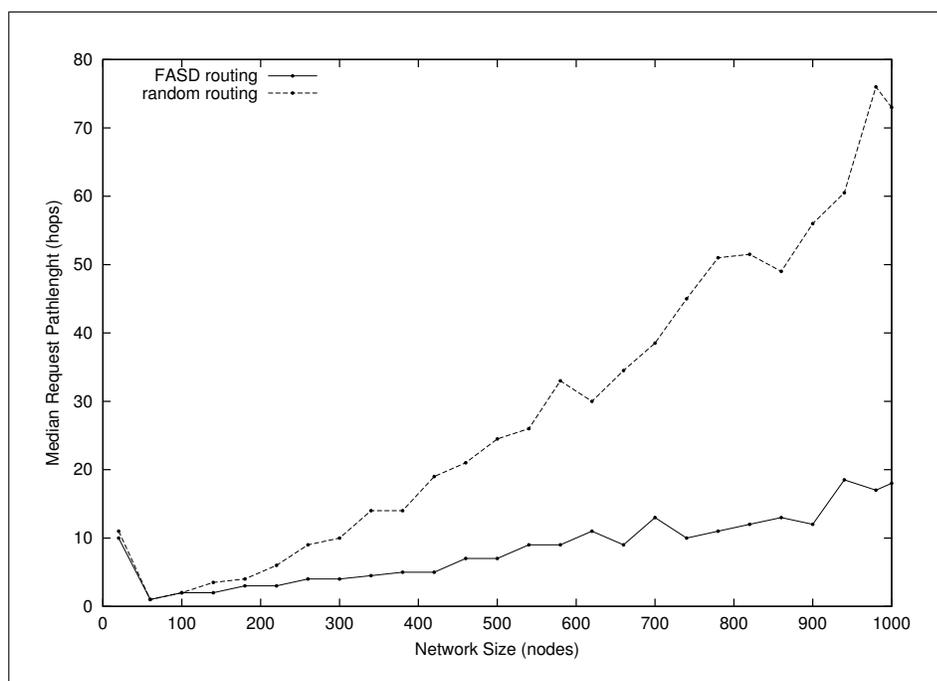
Figure 4.4: Median request pathlength under FASD and random routing. FASD is more efficient (retrieving the same top n results as centralized search in 18 hops versus 73 hops). See section 4.3 for details.

4.3 Routing

To test the routing heuristic used in Process-Query, the simulator compares its performance to a random router. Instead of forwarding to the reference associated with the closest metadata key, a random router contacts a neighbor selected at random. Figure 4.4 compares the evolution of median request path length under the two routing models. All other parameters are the same as given in section 4.2.

By the time the networks have grown to a size of 1000 nodes, the number of hops necessary to retrieve the same top n results as a centralized search is significantly higher under random routing. It seems that FASD's closeness based routing is critical to its performance.

4.3.1 The Cluster Hypothesis

Rijsbergen's clustering hypothesis asserts that "closely associated documents tend to be relevant to the same queries" [51]. His hypothesis offers deeper insight into why routing via PROCESS-QUERY is so effective. Unlike a random router, FASD routing distributes metadata keys in a consistent fashion. Specifically, the use of the cosine correlation value pushes the network towards a clustered hierarchy in which each node's data store contains only closely associated metadata keys. Thus, when FASD routes a query q to node i because it was the closest reference, there is a high likelihood that other metadata keys stored at node i will also be close to q.

A number of *ad hoc* probes failed to confirm increased key clustering in a FASD network versus a random network. The approach involved computing the centroid vector C for node j's data store. By observing the mean closeness of all metadata keys in j's data store to C, a measurement of metadata key clustering is obtained (a larger mean closeness indicates high key clustering). The centroid C is defined as:

$$C_k = (1/m) \sum_{i=1}^{m} t_{i,k} \tag{4.5}$$

where m is the number of metadata keys in node j's data store, C_k is the kth entry in C, and $t_{i,k}$ is the weight of term k in metadata key i.

The failure of these probes to detect a statistically significant difference suggests that clustering is not occurring at such a readily observable level. For any given node j in a FASD network there likely exists some subset of metadata keys K in j's store or immediate neighborhood that is tightly clustered. Keys that do not belong to K are likely due to to irrelevant queries spawned in the vicinity of or from node j. Additionally, these anomalous keys might be artifacts stored at j before adaptation was complete. Developing heuristics to discover these pockets of clustered metadata keys is critical to verifying the interesting claim that FASD realizes the goals of centralized document classification systems in a distributed and dynamic fashion.

4.4 Adaptability

To observe the adaptive characteristics of a FASD network, the simulator bootstraps a 1000 node network in a regular ring topology. At every timestep, the script selects a random node i in the network. Node i randomly decides to issue an insert of a randomly selected metadata key or a request
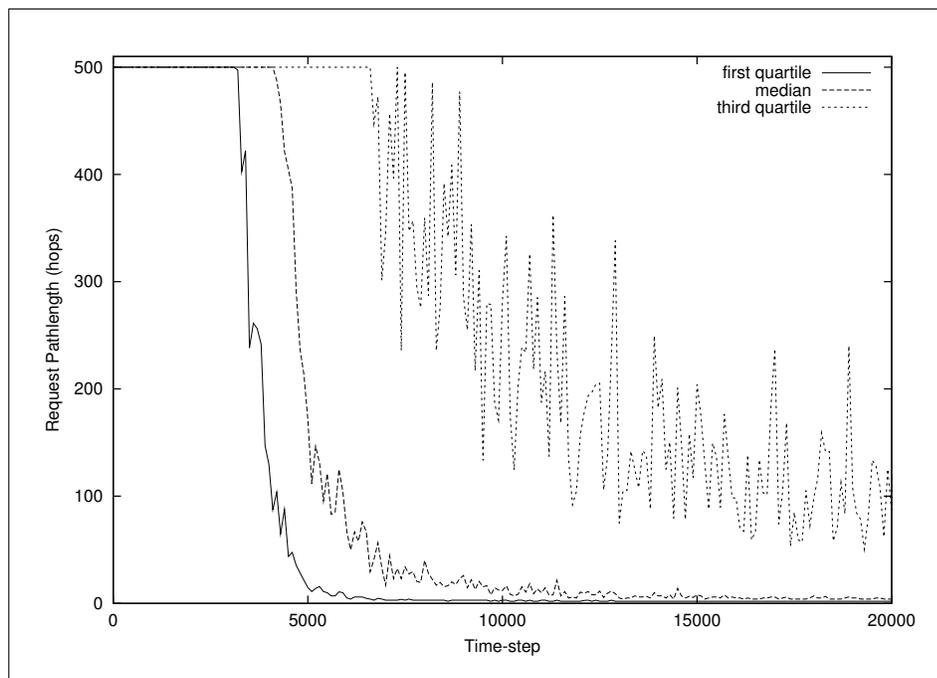
Figure 4.5: Request pathlength versus time. After 20,000 time-steps, the 1000 node network can retrieve the same top n results as a centralized search in a median number of 5 hops. See section 4.4 for details.

for a randomly generated query. Figure 4.5 describes the evolution of request pathlength over time. Within 20,000 time-steps, FASD requires only 5 hops to retrieve the same top ten results as centralized search. It seems that query requests and metadata inserts enable the network to increase its connectivity and improve routing efficiency.

A ring topology is selected for the reasons given in section 4.1.1 and because it is the most difficult state from which to converge. Network convergence is facilitated significantly by long-range edges that permit requests and inserts to travel to the specializing node. In a ring topology, there are no long-range edges and the mean number of hops necessary to reach a given node is maximized: it should be approximately $n/2$ (justified intuitively as half the number of hops necessary to get to the opposite side of the ring). Since FASD can converge from this state, it should be able to converge from any topology. Although tests using other topologies (e.g. random graphs

and k-regular graphs[6] with $k > 1$) suggest that this is true, an important goal for future research is theoretical support for the claim that convergence will occur from any connected topology.

4.4.1 Small-world

In [53] Watts and Stogatz identified so-called "small-world" networks. First, consider two structural properties of a graph $G = (V, E)$: its characteristic path length L and clustering coefficient C. L is the average shortest distance between all $u, v : u, v \in V$. C is the average cliquishness for all $u \in V$. To define the cliquishness of a vertex $u \in V$, consider the subgraph G' of G induced by the neighbors of u. That is, $G' = (V', E')$ where

$$V' = v \in V : (u, v) \in E$$
$$E' = (u, v) \in E : u, v \in V'$$

A measure of the cliquishness of u can now be given as:

$$\frac{\|E'\|}{(\|V'\|)(\|V'\| - 1)} \tag{4.6}$$

Starting from a ring lattice with n vertices and k edges per vertex, Watts and Stogatz examined the effect of randomly re-wiring each edge with probability p. When $p = 0$, L is approximately equal to $(n/2k)$ and C is approximately equal to 0.75. Conversely, when $p = 1$, L is approximately equal to $(\lg n / \lg k)$ and C is approximately equal to (k/n). As [53] proves, a large C does not necessarily lead to a large L nor does a small C necessarily lead to a small L. Small-world networks belong to a large range of $0 < p < 1$ over which the characteristic path length is close to that of a random graph while the clustering remain high. Apparently, the introduction of a long-range edge between u and v has a highly non-linear effect on L, shortening not only the distance between u and v but between their immediate neighbors, neighborhoods of neighbors, and so on. FASD's discovery of these long rang-edges in the network of figure 4.5 is likely responsible for the steep drop in the request pathlength between 4000 and 5000 time-steps.

Tests indicate that Freenet networks have a low characteristic path length and a relatively high clustering value [18]. Reassuringly the characteristic pathlength and clustering given in figure 4.6 confirm that the FASD meta-data layer also exhibits small-world characteristics. The clustering coeffi-

---

[6]A k-regular graph is a ring of of n vertices each of which is connected to its nearest k neighbors.
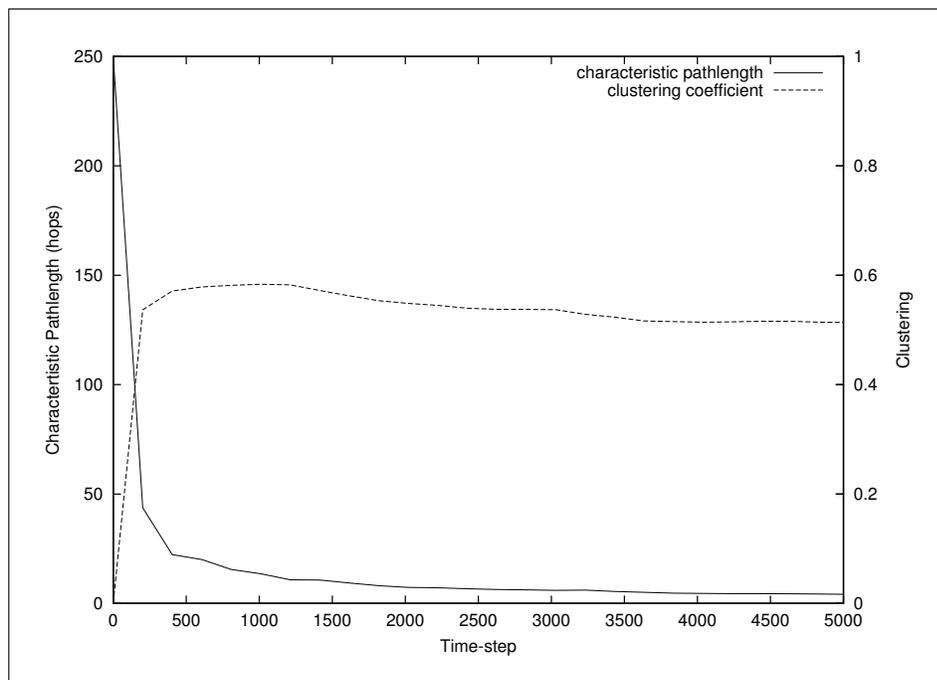
Figure 4.6: Evolution of characteristic pathlength and clustering over time. See section 4.4.1 for details.

cient in the FASD network is orders of magnitude larger than what would be expected in a random graph. However, the characteristic pathlength quickly converges to 2.5 which is consistent with a random graph.

Comparing figures 4.5 and 4.6 yields a disparity between the convergence of characteristic pathlength and the number of hops necessary to retrieve the same top n results as centralized search. Since FASD is equivalent to local search it will not necessarily yield the globally optimal decisions necessary to attain the lower bound on hops given by the characteristic pathlength. Moreover, there is no reason to expect that the shortest distance between any two nodes is equivalent to the number of hops necessary to retrieve the same top n results as centralized search. Unless one node contains all the top n hits, the latter will always be larger than the former. The two metrics are related to the extent that FASD's adaptation process drives the number of hops necessary to retrieve the same top n results as centralized search towards the lower bound given by the characteristic pathlength.
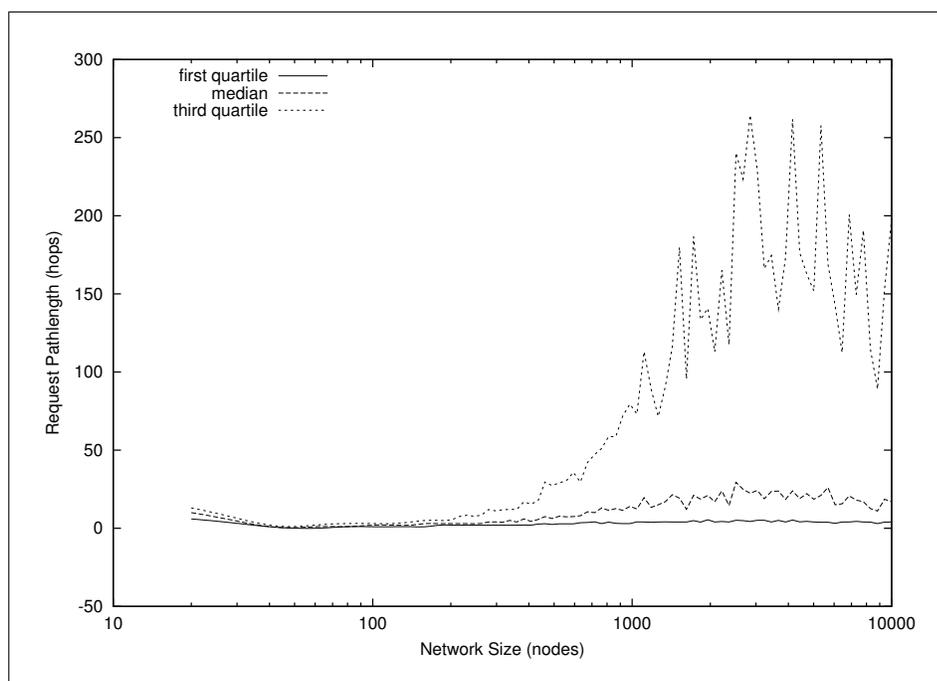
Figure 4.7: Request pathlength versus network size (logarithmic scale). A 10,000 node network can retrieve the same top n results as a centralized search with a median number of 17 hops. See section 4.5 for details.

4.5 Scalability

In order to examine FASD's scalability, extend the scenario in section 4.2 to a network of 10,000 nodes. Figure 4.7 indicates strong scalability with the 10,000 node network retrieving the same top n results as a centralized search in under 20 hops. Note that the worst-case request pathlength peaks at an inhibitive 250 hops. To determine whether or not the iterative deepening approach of section 3.2.2 might work to mitigate this worst-case performance, the simulator issues 300 shallow requests (hops-to-live of 20) at the end of the simulation. The histogram in figure 4.8 describes the ability of these shallow requests to retrieve the same top n results as centralized search. In the 10,000 node network, 83% of the shallow requests retrieve 50% or more of the top n hits centralized search would retrieve. The ability of shallow requests to retrieve most of the top n results makes a strong case for itera-
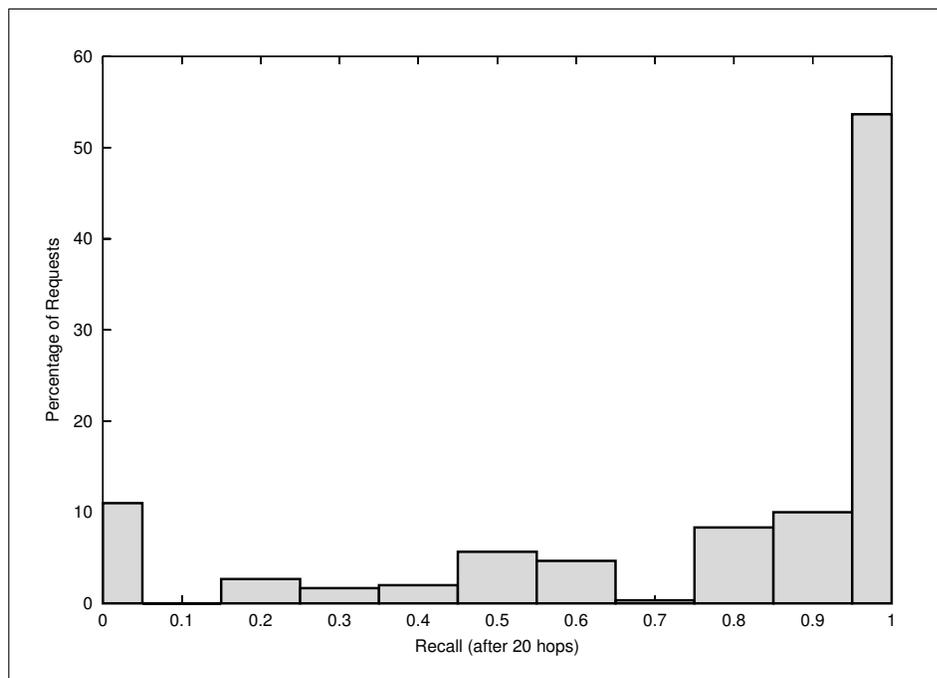
Figure 4.8: Distribution of recall achieved by 300 shallow requests (hops-to-live of 20) at the end of the simulation. In the 10,000 node network, 83% of the shallow requests retrieve 50% or more of the top n hits centralized search would retrieve. See section 4.5 for details.

tive deepening whereby the user's shallow request would retrieve most hits and a deeper request would locate the outstanding ones.

As section 4.4.1 discusses, a FASD network is a small world. Accordingly, its characteristic pathlength L is similar to that of a random graph and, as such, will scale logarithmically with the size of the network. Recall that L provides a lower bound on the number of hops necessary to retrieve the same top n results as centralized search. As the network organizes itself by clustering metadata keys and improving local routing decisions, the request pathlength converges towards L. Thus, the request pathlength is constantly driving down towards a lower bound that grows logarithmically in the number of nodes. The semi-log plot in figure 4.7 confirms that an exponential increase in the number of nodes precipitates only a linear increase in the median request pathlength.
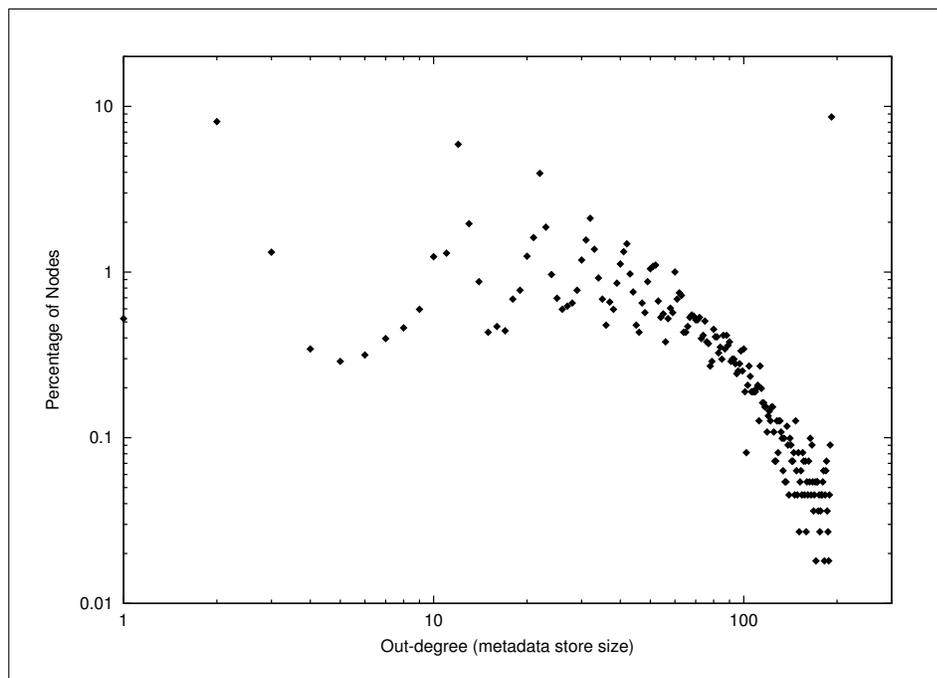
Figure 4.9: Out-degree distribution at the end of the simulation. A large percentage of nodes have low out-degree (poorly connected) while a small percentage have large out-degree (well connected). See section 4.6 for details.

4.6 Fault-tolerance

[2] examines the correlation between a network's connectivity distribution $P(k)$ and its attack tolerance. $P(k)$ is the probability that a node in the network has out-degree k. If $P(k)$ decays as a power law (i.e. $P(k) \approx k^{-\lambda}$), the corresponding network is said to be "scale-free". In these networks, a small portion of nodes have high connectivity while a large portion are weakly connected. Figure 4.9 describes the out-degree distribution in the FASD network from section 4.2. The anomalous point at an out-degree of 200 is due to data store saturation. Increasing the meta data store size would distribute the anomalous point in a scale-free distribution. The linearity of the degree distribution in logarithmic space demonstrates that FASD, like Freenet, is a scale-free network.

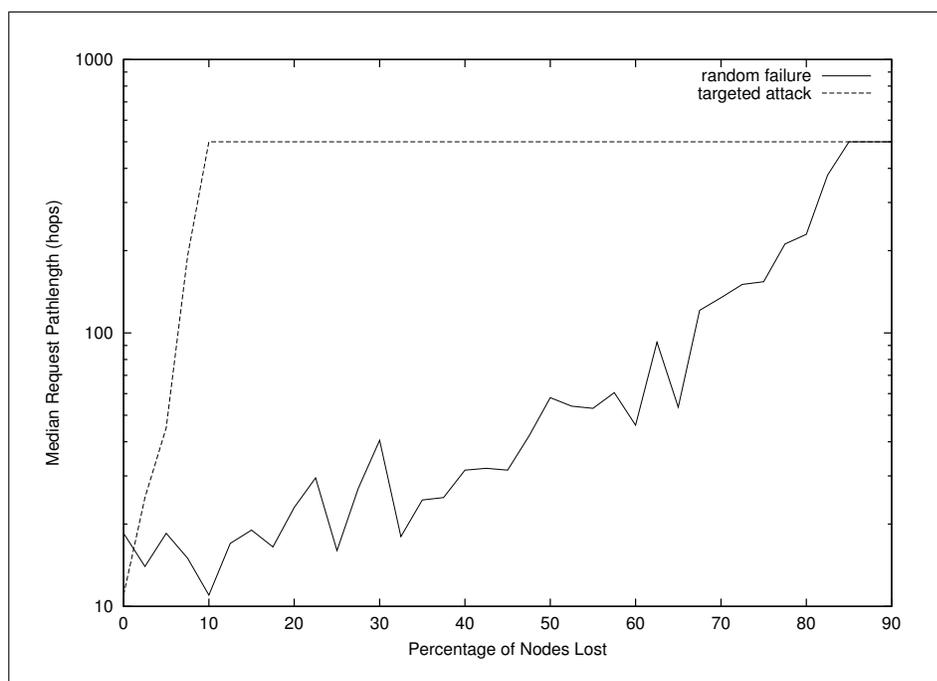To understand why such a distribution arises consider a new node i that

Figure 4.10: Comparing the impact of random failure and targeted attack on median request pathlength. See section 4.6 for details.

enters the network and a richly connected node j already in the network. Since the announcement process follows random links and since j is better connected than most nodes, there is a strong likelihood i will form a link to j. Subsequent queries/inserts through i will further enrich j's connectivity. Also note that there is a strong probability that node i will never become as connected as node j because new nodes will "prefer" to attach to j and existing nodes will consistently route more requests/inserts through j. Thus, the node announcement protocol coupled with growth generates and maintains a scale-free link distribution.

Scale-free networks are extremely tolerant of random failure. Since the majority of nodes have few links, nodes with small connectivity will be selected with much higher probability. Removal of these nodes has a minimal impact on overall network performance. However, if a malicious agent gains knowledge of the network's topology and targets the small portion of richly connected hub nodes, performance degradation is far more rapid. The dif-

ficulty of discovering a FASD network's topology[7] helps mitigate this risk.

Figure 4.10 describes fault-tolerance under random failure and targeted attack in the network of section 4.5. Fault-tolerance is modeled by progressively removing nodes from the network and observing the impact on the median request pathlength. For every 2.5% of nodes lost, the simulator issues 300 requests and observes how many hops are necessary to retrieve the same top n results as a centralized search. As described in section 4.1.2, requests requiring more than 500 hops are treated as taking 500 hops. When modeling random failure, the simulator removes nodes at random. Conversely, when modeling targeted attack, it removes the most-connected nodes first.

As expected, FASD is highly resistant to targeted attack with the median number of hops necessary to match centralized search remaining below 50 even when up to 47.5% of nodes fail. This resistance to random failure is due to FASD's scale-free link distribution and the replication of metadata keys across the network. When exposed to a targeted attack, the network becomes unusable almost immediately with the number of hops necessary to match centralized search passing 50 after only 5% of nodes are lost.

---

[7]It is not even obvious how to evaluate the size of a FASD network.

Chapter 5

Future Work

This chapter organizes the various threads for future research identified throughout the paper into three broad categories: improved result quality, enhanced security, and further simulation.

5.1 Improved Result Quality

Currently, FASD utilizes a primitive closeness function that fails to incorporate non-query factors such as a document's authority value or popularity. It is critical to security and usability that the successful algorithms in [6, 34] be incorporated into FASD. How to collect this information in a decentralized and encrypted system such as Freenet remains an open question.

5.2 Enhanced Security

The techniques for search on encrypted data in [49] offer great potential for dramatically improving FASD's current level of security and anonymity. Unfortunately, there is the difficult problem of distributing decryption keys to searchers without compromising them to untrusted node operators. A solution may lie in a trusted middle layer that mediates encryption of searches from consumers and decryption of results passed backed from nodes. The challenge lies in implementing such a layer without any intermediary central authority. Furthermore, as [49] points out, efficient update schemes for encrypted indices are required.

5.3   Further Simulation

More sophisticated simulation is necessary to verify the claim that FASD is a distributed implementation of the clustering techniques described in [43]. Efforts should also be made to model the trade-off between result quality and compression schemes for metadata key and lexicons. Finally, simulation models involving complex queries are necessary to confirm the efficacy of the approach suggested in section 3.2.2.

Chapter 6

Conclusion

The fundamental problem facing any distributed application is the location of data in a transient network without centralized control or hierarchical organization. A cursory examination of the peer-to-peer landscape might suggest the problem has been solved with the advent of Freenet's adaptive routing and Chord's consistent hash paradigm. Although these networks can fetch a requested file with laudable efficiency ($O(\log N)$ hops), they are incapable of identifying *which* files will be the most pertinent to a given user's query. In a world of information overload, the seriousness of this shortcoming cannot be understated. Not surprisingly, the most successful peer-to-peer networks such as Napster and Gnutella include(d) an integrated search mechanism. Unfortunately, these are best described as *ad hoc*, lacking scalability, decentralization, or fault-tolerance.

FASD is the first search architecture that is completely decentralized, self-maintaining, and scalable. A FASD metadata layer sits on top of an existing distributed system to offer search capability comparable to that of a centralized inverted index. At the crux of the system are automatically generated metadata keys and a powerful primitive that organizes, replicates, and retrieves these keys without any mediating authority.

Simulations of a FASD layer for Freenet indicate excellent scalability and performance. The simulation networks display a high degree of fault-tolerance against random failure at the expense of decreased resistance to targeted attack. Fortunately, the adherence to most of Freenet's stringent security requirements mitigates this risk. Currently, the most pressing issue is FASD's naïve use of the cosine correlation value. A more sophisticated metric that incorporates non-query factors such as a document's "authority value" and popularity is necessary from a usability and security standpoint.

Despite its rudimentary closeness operator, FASD offers a powerful augmentation to the Freenet network that this community will hopefully consider adopting[1]. More broadly, the system has promising potential to any distributed application that can benefit from a fault-tolerant, adaptive, scalable, distributed search engine.

---

[1]In e-mail correspondence, Ian Clarke remarks, "[FASD] is extremely impressive, both in concept and execution, and I definitely think that this mechanism, or a derivative of this mechanism, should go on the [Freenet] v0.6 to do list"

Bibliography

[1] ADLER, S. The slashdot effect, an analysis of three internet publications. *Linux Gazette* (March 1999).

[2] ALBERT, R., JEONG, H., ET AL. Error and attack tolerance of complex networks. *Nature*, 406 (July 2000), 378–382.

[3] ALMEIDA, V., BESTAVROS, A., ET AL. Characterizing reference locality in the WWW. Tech. Rep. 1996-011, 21, 1996.

[4] BORODIN, A., ROBERTS, G. O., ET AL. Finding authorities and hubs from link structures on the world wide web. In *World Wide Web* (2001), pp. 415–429.

[5] BRESLAU, L., CAO, P., ET AL. Web caching and zipf-like distributions: Evidence and implications. In *INFOCOM (1)* (1999), pp. 126–134.

[6] BRIN, S., AND PAGE, L. The anatomy of a large-scale hypertextual Web search engine. *Computer Networks and ISDN Systems 30*, 1–7 (1998), 107–117.

[7] CHO, J., GARCÍA-MOLINA, H., ET AL. Efficient crawling through URL ordering. *Computer Networks and ISDN Systems 30*, 1–7 (1998), 161–172.

[8] CLARKE, I. A distributed decentralised information storage and retrieval system. http://citeseer.nj.nec.com/clarke99distributed.html.

[9] CLARKE, I., MILLER, S. G., ET AL. Protecting free expression online with freenet. *IEEE 6*, 1 (January/February 2002), 40–49.

[10] CLARKE, I., SANDBERG, O., ET AL. Freenet: A distributed anonymous information storage and retrieval system. In *Workshop on Design Issues in Anonymity and Unobservability* (2000), pp. 46–66.

[11] COOPER, W. A definition of relevance for information retrieval. *Information Storage and Retrieval 7*, 1 (June 1971), 19–37.

[12] CUNHA, C., BESTAVROS, A., ET AL. Characteristics of WWW client-based traces. Tech. Rep. 1995-010, 1, 1995.

[13] Freegle. http://www.freegle.com. Site was down as of March 31, 2002.

[14] Freenet. http://freenetproject.org.

[15] Gnutella. http://www.gnutella.com.

[16] GOFFMAN, W. On relevance as a measure. *Information Storage and Retrieval 2*, 3 (December 1964), 201–203.

[17] Google. http://www.google.com.

[18] HONG, T. *Performance*. In Oram [32], 2001, ch. 14.

[19] HUANG, L. A survey on web information retrieval technologies. Tech. rep., ECSL, 2000.

[20] JONES, K. S. A statistical interpretation of term specificty and its application in retrieval. *Journal of Documentation 28*, 1 (March 1972), 11–20.

[21] JOVANOVIC, M. A., ANNEXSTEIN, F. S., ET AL. Scalability issues in large peer-to-peer networks—a case study of gnutella. Tech. rep., University of Cincinnati, 2001.

[22] KARGER, D. R., LEHMAN, E., ET AL. Consistent hashing and random trees: Distributed caching protocols for relieving hot spots on the world wide web. In *ACM Symposium on Theory of Computing* (1997), pp. 654–663.

[23] Kazaa. http://www.kazaa.com.

[24] KOBAYASHI, M., AND TAKEDA, K. Information retrieval on the web. *ACM Computing Surveys 32*, 2 (2000), 144–173.

[25] KUBIATOWICZ, J., BINDEL, D., ET AL. Oceanstore: An architecture for global-scale persistent storage. In *Proceedings of ACM ASPLOS* (November 2000), ACM.

[26] LAWRENCE, S., AND GILES, C. L. Accessibility of information on the web. *Nature 400* (1999), 107–109.

[27] LESSIG, L. *Code and other laws of cyberspace.* Basic Books, New York, NY, 1999.

[28] MICROSOFT. Cache array routing protocol. White Paper, 1997.

[29] MINAR, N., AND HEDLUND, M. *A Network of Peers: Peer-to-Peer Throughout the History of the Internet.* In Oram [32], 2001, ch. 1.

[30] Napster. http://www.napster.com.

[31] Online book initiative. http://gopher.std.com/obi/.

[32] ORAM, A., Ed. *Peer-To-Peer: Harnessing the Power of Disruptive Technologies.* O'Reilly, Sebastopol, California, 2001.

[33] ORWANT, J. What's on freenet. http://www.openp2p.com, November 2000.

[34] PAGE, L., BRIN, S., MOTWANI, R., ET AL. The pagerank citation ranking: Bringing order to the web. Tech. rep., 1998.

[35] Project gutenberg. http://promo.net/pg/.

[36] RITTER, J. Why gnutella can't scale. no really. http://www.darkridge.com/ jpr5/doc/gnutella.html.

[37] ROWSTRON, A., AND DRUSCHEL, P. Pastry: Scalable, decentralized object location and routing for large-scale peer-to-peer systems.

[38] RUSSELL, S., AND NORVIG, P. *Artificial Intelligence: A Modern Approach.* Prentice-Hall, 1995.

[39] RUSSELL, S., AND NORVIG, P. *Inference in First-Order Logic.* In [38], 1995, ch. 9, pp. 265–296.

[40] RUSSELL, S., AND NORVIG, P. *Search Strategies.* In [38], 1995, ch. 3, pp. 55–91.

[41] SALTON, G., AND MCGILL, M. J. *Introduction to Modern Information Retrieval.* McGraw-Hill, New York, 1983.

[42] SALTON, G., AND MCGILL, M. J. *Retrieval Evaluation.* In [41], 1983, ch. 5, pp. 118–156.

[43] SALTON, G., AND MCGILL, M. J. *Retrieval Refinements.* In [41], 1983, ch. 6, pp. 199–256.

[44] SALTON, G., AND MCGILL, M. J. *The SMART and SIRE Experimental Retrieval System.* In [41], 1983, ch. 4, pp. 118–156.

[45] SALTON, G., AND MCGILL, M. J. *Systems Based on Inverted Files.* In [41], 1983, ch. 2, pp. 24–51.

[46] SALTON, G., AND MCGILL, M. J. *Text Analysis and Automatic Indexing.* In [41], 1983, ch. 3, pp. 52–117.

[47] SHAPIRO, A. L. *The Control Revolution.* PublicAffairs™, New York, NY, 1999.

[48] SHAPIRO, C., AND VARIAN, H. R. *Information rules: a strategic guide to the network economy.* Harvard Business School Press, Boston, Massachusetts, 1998.

[49] SONG, D. X., WAGNER, D., ET AL. Practical techniques for searches on encrypted data. In *IEEE Symposium on Security and Privacy* (2000), pp. 44–55.

[50] STOICA, I., MORRIS, R., ET AL. Chord: A scalable peer-to-peer lookup service for internet applications. Tech. Rep. TR-819, MIT, March 2001.

[51] VAN RIJSBERGEN, C. *Information Retrieval*, 2 ed. In [52], 1979, ch. Automatic Classification.

[52] VAN RIJSBERGEN, C. *Information Retrieval*, 2 ed. Butterworths, 1979.

[53] WATTS, D. J., AND STROGATZ, S. H. Collective dynamics of 'small-world' networks. *Nature 440–442*, 393 (1998).