A Scalable Peer-to-peer Protocol Enabling Efficient and Flexible Search

Reaz Ahmed and Raouf Boutaba
David R. Cheriton School of Computer Science
University of Waterloo
Waterloo, ON, CA N2L 3G1
{*r5ahmed,rboutaba*}*@uwaterloo.ca*

(Technical Report: CS-2006-05)

Abstract

Efficient discovery of information, based on partial knowledge, is a challenging problem faced by many large scale distributed systems. This paper presents a peer-to-peer search protocol that addresses this problem. The proposed system provides an efficient mechanism for advertising a binary pattern, and discovering it using any subset of its 1-bits. A pattern (e.g., Bloom filter) summarizes the properties (e.g., keywords or service description) associated with a shared object (e.g., document or service).

The proposed system has a *partially decentralized* architecture involving superpeers and adopts a novel *structured search* mechanism derived from the theory of *Error Correcting Codes (ECC)*. Better resilience to peer failure is achieved by utilizing replication and redundant routing paths. The number of routing hops and the number of links maintained by each superpeer scales logarithmically with the number of superpeers. The concept presented in this paper is supported with theoretical analysis, and simulation results.

1 Introduction

Locating the peer, responsible for a given data item, is an essential functionality for most peer-to-peer (P2P) systems, featuring frequent joins and leaves of peers. Distributed Hash Table (DHT) techniques, such as Chord[24], CAN[22], Kademlia[21], have been extensively investigated for their efficiency in performing this task. These techniques assign a portion of the key space to each peer and offer a single functionality: given a key, find the ID of the peer responsible for that key. Keys and IDs belong to the same space, usually 160-bit integers. A key is associated with a data item, and is computed by hashing the data item or some of its identifying properties.

A major handicap of DHT-techniques is their inability to resolve inexact queries, i.e., queries that are generated from a partial knowledge of the target data item. Many distributed applications, including content sharing P2P systems and service discovery systems, are required to provide support for inexact queries. For example, in service discovery systems [18] (e.g., Jini, SLP and Salutation), a service is advertised as a list of descriptive attribute-value pairs, called service description. Queries for a service include only a subset of the attribute-value pairs previously advertised in the description of that service. Similarly, for a file-sharing P2P system, the users do not always know the exact name of the advertised file. Instead, queries are based on a subset of the keywords that are present in the filename.

Several research proposals have focused on building an additional layer on top of a DHT protocol for supporting subset matching. For example, Squid[23] and Twine[6] build a middleware on top of Chord to support partial keyword matching (prefixes only) and service discovery, respectively. DHT-techniques cluster keys based on numeric distance. But, for efficient subset matching keys should be clustered based on hamming distance[1]. As a result, these solutions

[1]The Hamming distance between two bit vectors X and Y of same length is calculated as $d(X,Y) = |X \oplus Y|$ = number of bits on which X and Y disagree.

(Squid and Twine) generate many independent DHT-lookups for resolving one query. More recently, the problem of subset matching in distributed environments has been addressed in [3]. Here, a hierarchical architecture, utilizing semi-structured search, has been proposed.

The contribution of this paper is a scalable and efficient P2P search protocol for supporting subset search. Instead of using keys, we have used patterns (like Bloom filters [7]) to summarize the identifying properties associated with a shared object. The proposed system provides an efficient mechanism for advertising a binary pattern, and discovering it by using any subset of its 1-bits. We adopted a *partially decentralized* architecture involving superpeers, and adopts a novel *structured search* mechanism based on the theory of *Error Correcting Codes (ECC)*. Better resilience to peer failure is attained by using replication and redundant routing paths. The number of routing hops and the number of links maintained by each superpeer scales logarithmically with the number of superpeers. The concept presented in this paper is supported with theoretical analysis, and simulation results obtained from the application of these concepts to partial keyword search utilizing the extended Golay code[16].

The rest of this paper is organized as follows. Section 2 compares the proposed system with related works. The preliminaries on coding theory and Bloom filters are presented in Section 3. Section 4 explains the new system, and proves several of its properties. Simulation results, supporting our claims, are presented in Section 5. Finally we conclude and outline our future research goals in Section 6.

2 Related Works

Sate-of-the-art solutions for subset matching [4], [12] in centralized environments, hold linear relationship with the number of patterns to be matched against. Hence, equivalent solutions in a distributed environment, where the patterns are distributed across networked nodes, will require flooding the network.

Pattern matching in distributed environments has been addressed in [3]. This work presents the Distributed Pattern Matching System (DPMS). DPMS organizes indexing peers in a lattice-like hierarchy and uses restricted flooding (within $O(logN)$ peers) at the topmost level of the hierarchy. DPMS uses Bloom filters as meta-information for routing, and relies on replication for increasing fault-resilience. DPMS uses a don't care based aggregation scheme to reduce the volume of index information at higher level peers. DPMS requires $O\left(log\frac{N}{logN}\right)$ additional hops for finding each match, after a set of $O(logN)$ peers at the topmost level has been flooded. On the contrary, we can discover all the matches by searching a limited number of superpeers.

Several research activities (including [23], [26]) add a layer on top of DHT to support keyword search. Squid [23] adopts space-filling-curves to map similar keywords to numerically close keys, and uses Chord [24] for routing. It supports partial prefix matching (e.g. queries like compu* or net*) and multi keyword queries. In contrast to the new system, Squid does not have the provision for supporting true inexact matching for queries like **net**. pSearch [26] aims to support full-text search. It uses Information Retrieval techniques, like vector space model and latent semantic indexing, on top of CAN [22]. Queries and data are represented by term vectors. Searches are performed in multidimensional Cartesian space. In this technique search performance degrades with increase in dimensionality.

Unstructured systems ([2],[1]) can support partial keyword search. These systems depend on blind search techniques, such as flooding [2] and random walk [20]. Hence, the generated volume of query traffic does not scale with the growth in network size. Many research activities are aimed towards improving the routing performance of unstructured P2P systems by adopting hint-based routing strategies. In [28] and [27], peers learn from the results of previous routing decisions, and bias future query routing based on this knowledge. In [9], routing is biased by peer capacity; queries are routed to peers of higher capacity with a higher probability. In [11], peers are organized according to common interests, and restricted flooding is performed in different interest groups. In [9], [28] and [19], peers store index information from other peers within a neighborhood radius of 2 or 3 hops. These techniques reduce the query traffic volume to some extent, but do not provide any guarantee on search completeness or any bound on the volume of query/advertisement traffic.

Service discovery is another application of the new system presented in this paper. Service discovery systems rely on three-party architecture, composed of clients, services and directory entities. Directory entities gather advertisements

from services and resolve queries from clients. Major protocols for service discovery from industry, like SLP, Jini, Salutation, etc, assume a few directory agents, and do not provide any structured way of locating service descriptions. Solutions from academia, like Secure Service Discovery Service (SSDS) [14] and Twine [6], target Internet-scale service discovery and face the challenge of achieving efficiency and scalability in locating service descriptions based on partial information. SSDS relies on a tree-like hierarchy of directory entities. It uses Bloom filters for representing service descriptions. Bit-wise OR-base aggregation scheme is adopted for reducing the volume of index information at higher level directory entities. SSDS suffers from load-balancing problem and is vulnerable to the failure of higher level directory entities along the index tree.

Twine [6], on the other hand, uses a hierarchical naming scheme and relies on Chord as the underlying routing mechanism. Twine generates a set of strands (substrings) from the advertisement or query, computes keys for each of these strands, and finally performs search or advertisement using these keys. The number of DHT-lookup increases with the number of attribute-value pairs in a name and so the amount of generated traffic is high. Also, load-balancing is a major problem in this system. Peers responsible for small or popular strands become overloaded, and the overall performance degrades.

In summary, the proposed system can be used to solve the generic problem of subset search in distributed environments without compromising scalability and efficiency requirements. In particular, it can help in reducing search traffic, resulting from multiple DHT-lookups, as in [6],[23], and the lack of scalability displayed by hint-based unstructured systems such as [2], [19] and [11].

3 Preliminaries

In this section, we explain the properties of linear codes, extended Golay code and Bloom filters. We highlight only the properties that will be required for our discussion in subsequent sections.

3.1 Linear Binary Codes

Let $\mathbb{F}_2^n$ define the linear space of all n-tuples (or vector) over the finite field $\mathbb{F}_2 = \{0, 1\}$. A linear binary code of length n is a *subspace* $\mathcal{C} \subset \mathbb{F}_2^n$. The n-tuples, forming the *subspace*, are called *codewords*. A linear binary code is specified by using three parameters (n, k, d). Here, k is the dimension (or rank) of the code. This indicates that there exists a total of 2^k codewords in the code. d is the minimum hamming distance between any two codewords. A good (n, k, d)-code has a small n (for fast transmission), a large k (for increased information content), and a large d (for correcting many errors).

Since the set of codewords in $\mathcal{C}$ is a *subspace* of $\mathbb{F}_2^n$, the XOR of any two codewords, u and v, is also a codeword, i.e., $\forall u, v \in \mathcal{C} \implies u \oplus v \in \mathcal{C}$. This property allows the entire code to be represented in terms of a minimal set of codewords, known as a *basis* in linear algebra terminology. A *basis* for an (n, k, d) code contains exactly k codewords. These k codewords, $g_1, g_2, \ldots, g_k$, are collated in the rows of a $k \times n$ matrix known as the *generator matrix*, $G_{\mathcal{C}}$, for code $\mathcal{C}$. The codewords of $\mathcal{C}$ can be generated by XORing any number of rows[2] of G. The generator matrix for any linear code can be expressed as,

$$G = [I_k B] = [g_1 g_2 \ldots g_k]^T \tag{1}$$

where, I_k is the $k \times k$ identity matrix, and B is a $k \times (n - k)$ matrix.

The dual code $\mathcal{C}^\perp$ of linear code $\mathcal{C}$ is defined as

$$\mathcal{C}^\perp = \left\{ x \in \mathbb{F}_2^n | x \cdot c = \vec{0} \forall c \in \mathcal{C} \right\}$$

Here, $x \cdot c$ represents bit-wise product. A linear code is said to be *self-dual*, if $\mathcal{C}^\perp = \mathcal{C}$. For any codeword c of a self-dual linear code $\mathcal{C}$, $c \in \mathcal{C} \implies \bar{c} \in \mathcal{C}$, $\bar{c}$ represents the bit-wise complement of c.

[2]Note that $\sum_{i=0}^{k} \binom{k}{i} = 2^k$.

3.2 Extended Golay Code

The extended Golay code, $\mathcal{G}_{24}$, is a $(24, 12, 8)$ self-dual linear binary code. It has $4096 (= 2^{12})$ codewords of length 24-bits each. The minimum distance between any two codewords is 8. The weight[3] distribution of this code is $0^1\, 8^{759}\, 12^{2576}\, 16^{759}\, 24^1$. $\mathcal{G}_{24}$ contains the all zero vector $\vec{0}$[4] and the all one vector $\vec{1}$. Exactly 759 codewords have weight 8 (known as *special octads*), 2576 codewords have weight 12 (known as *umbral dodecads*), and 759 codewords have weight 16 (called *16-sets*).

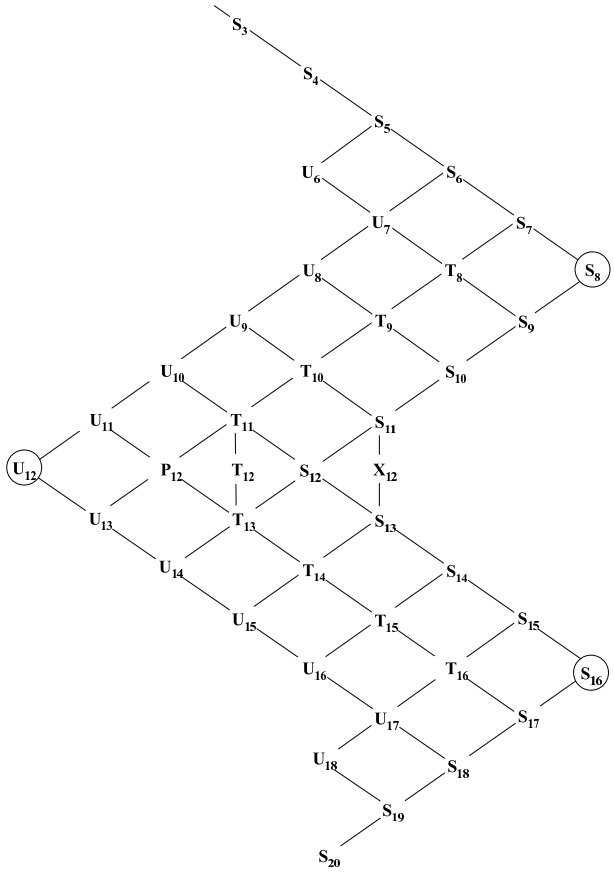


Figure 1: Relationships among the orbits of the vectors in $\mathbb{F}_2^n$ w.r.t. to the codewords in $\mathcal{G}_{24}$. Circled orbits correspond to the codewords of $\mathcal{G}_{24}$.

All the possible vectors in $\mathbb{F}_2^n$ can be categorized into 49 *orbit*s w.r.t. $\mathcal{G}_{24}$ [13]. These orbits are denoted as $S_w (0 \leq w \leq 8)$, $T_w (8 \leq w \leq 16)$, $U_w (6 \leq w \leq 18)$, P_{12} and X_{12}, where the subscript w denotes the weight of the vectors in that orbit. Figure 1 (a portion of Figure 1 in [13])) depicts some of these orbits. An edge between orbits A and B indicates that a vector in orbit B can be obtained from some vector in orbit A (and vice versa) by complementing a single bit. The minimal hamming distance of a vector in orbit A from some vector in orbit B is essentially the length of the shortest path from node A to node B in the graph of Figure 1. Orbit S_8 consists of the 759 *special octads*, orbit U_{12} consists of the 2576 *umbral dodecads*, and orbit S_{16} comprises of the 759 16-sets.

3.3 Bloom Filters

A Bloom filter [7] is a compact data-structure used to represent a set. Bloom filters are used in many network applications [8] due to their space-efficiency, and their ability to support set membership test operation. However, the membership test operation may result into false (erroneously) positives with a small probability.

[3]The number of 1-bits in a bit vector is known as its weight.

[4]Any linear code contains the all zero vector, $\vec{0}$

An m-bit array is used to represent a Bloom filter. k different hash functions are also required to be defined. In an empty Bloom filter all the bits are set to zero. To insert an element in a Bloom filter, it is hashed with the k hash functions to obtain k positions in the bit-array and corresponding k-bits are set to 1. The membership test process is similar to the insert process. The element, say x, to be tested for set membership, is hashed with the same k-hash functions and corresponding k-positions in the bit-array are checked. If any of these k-bits is not 1 then x is definitely not a member of the set represented by this Bloom filter. On the other hand, if all of these k-bits are 1, then there is a high probability that x is a member of the set. The false-positive probability for a Bloom filter, representing an n-element set, is calculated as

$$\epsilon = \left(1 - \left(1 - \frac{1}{m}\right)^{nk}\right)^{k}$$

ϵ is minimized when $k = \ln 2 \cdot (m/n)$. For example, with $m/n = 6$ and $k = 3$, $\epsilon \approx 0.06$. For a well designed Bloom filter about $33 - 50\%$ of the bits are 1.

4 The New System

In this section, we define a protocol that governs the connectivity, and eventually the structure of a P2P overlay for enabling flexible subset search.

4.1 Overview

From a functional point of view, peers in our system can be categorized as superpeers and leaf peers. In addition to the functionalities (mostly file transfers) carried out by the regular leaf peers, superpeers are responsible for indexing meta-information about the content published by the leaf peers and other superpeers, and for routing queries based on this information. Superpeers connect to a larger number of peers than the leaf peers, have higher capacity (bandwidth and processing power) and longer uptime. According to the classification presented in [5], our system has a *partially decentralized* architecture utilizing *structured search* in the superpeer network.

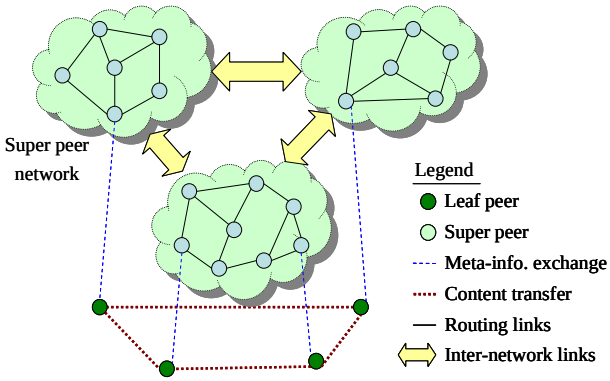


Figure 2: Overview of the new system.

As depicted in Figure 2, superpeers collaborate in subnets for routing advertisements and queries. Each subnet is capable of routing messages independently of the other subnets in the system. As reported in [25], the modern two-tier Gnutella network contains around a million peers. About 18% of these peers participate as superpeers. This implies that each subnet should contain around 4000-16000 (i.e., $2^{12} - 2^{14}$) superpeers for a network with 10-20 subnets.

We have used Bloom filters [7] as meta-information for routing. Bloom filters are used for summarizing the properties of shared objects and for enabling subset matching.

Similar to DHT-based systems, we employed a three-party rendezvous mechanism for query resolution. Advertisements and queries are routed to different sets of superpeers in such a way that a query set and an advertisement set has at least one superpeer in common, whenever a query has a subset of the 1-bits, as present in an advertised pattern. The idea is to partition the entire pattern space $\mathbb{F}_2^n$ into clusters and select a unique representative for each cluster. Let $\mathcal{C}$ be the set of these representative patterns. For any advertised pattern P and search pattern Q, we need to find two sets: $advSet(P) \subset \mathcal{C}$ and $qSet(Q) \subset \mathcal{C}$, satisfying Equation 2.

$$Q \subseteq P \implies qSet(Q) \cap advSet(P) \neq \emptyset \tag{2}$$

To minimize the cardinality of $advSet$ and $qSet$, patterns in close proximity (w.r.t Hamming distance and not Euclidean distance) should be grouped under the same cluster.

Clustering of pattern space has been extensively studied in Artificial Intelligence (AI) and Coding Theory literature. AI-based clustering techniques [17] require priori knowledge of the pattern space (e.g., pattern density distribution) and training phases. Coding theory constructs, on the other hand, assume that all the patterns are equally likely. These techniques distinguish a set of patterns as cluster-heads (codewords), which cover the entire (or most of the) pattern space with no (or very little) overlaps. Each superpeer is assigned a codeword and is responsible for the patterns within its *covering radius*, consisting of all patterns within Hamming distance $\frac{d}{2}$.

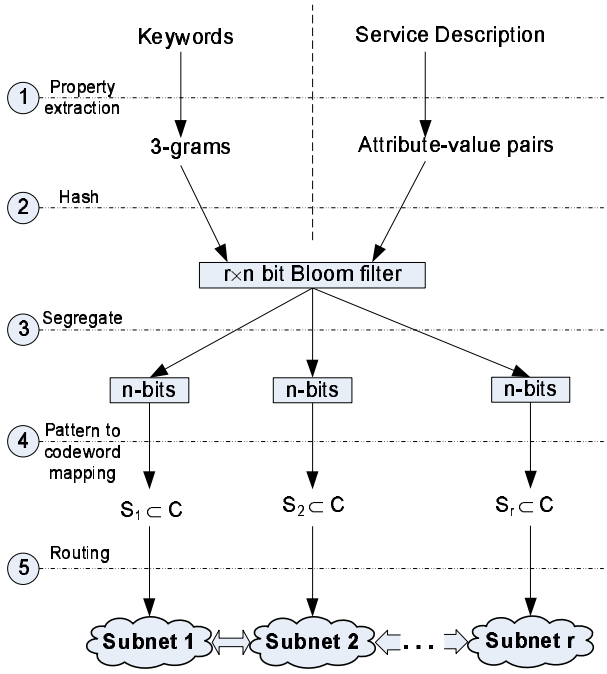


Figure 3: Search/advertisement process.

Figure 3 depicts the search/advertisement process. In this figure, step 1 and 2 are specific to the application under consideration, e.g., keyword search, service discovery etc. The input to the system is a $r \times n$ bit pattern (in this example, a Bloom filter), representing a query or an advertisement. Here, n is the length of the linear code used for the routing process, and r is the number of subnets in the system. In our implementation, we have used the extended Golay code $\mathcal{G}_{24}$, and thus $n = 24$. The input bit-vector is segregated into r chunks (step 3), each n bits long. An n-bit chunk is then mapped to a set of codewords S_i ($S_i \subset \mathcal{G}_{24}$ for our implementation) and forwarded to any superpeer in subnet i. The superpeer routes the message in $O(\frac{1}{2}|S_i| \log |C|)$ hops to the target superpeers in S_i. The algorithms for mapping an n-bit pattern to a set of codewords ($S_i \subset \mathcal{G}_{24}$) for the search and advertisement processes are explained in Section 4.3. The routing algorithm within a subnet is presented in Section 4.2.

If a query/advertisement message is propagated to all of the r-subnets, then the implied redundancy will be very high. We have reduced this overhead by adopting the concept of voting algorithm as presented in [15]. In particular, an advertisement is propagated to $\lfloor \frac{r}{2} + 1 \rfloor$ subnets, and a query message is propagated to $\lfloor \frac{r+1}{2} \rfloor$ subnets. This ensures that there exists at least one subnet receiving an advertisement, and any query matching that advertisement.

Steps 1 and 2 in Figure 3 are always executed by a leaf peer. Steps 3 to 5 can be executed either by a leaf peer or by a superpeer, depending on the implementation choice. We prefer the leaf peers to calculate the *advSet* and *qSet*, since these operations are CPU intensive. The leaf peer can then submit the message, containing the list of target codewords, to any known superpeer. It should be left as the responsibility of a superpeer to maintain extra links to other superpeers outside its subnet, and forward a message to appropriate subnets.

4.2 Routing within a Subnet

In this section, we present an algorithm for routing within a subnet. First, we explain the mechanism for routing a message originating at peer X to a single target peer Y. Then we provide an algorithm for multicasting the message to multiple destinations. By "peer X" we mean a super-peer responsible for codeword X. In this section, a superpeer is assumed to be associated with a single codeword. Later in Section 4.5, a method of associating multiple codewords with a superpeer is described.

As discussed in Section 3.1, the codewords of a linear code (C) form a vector subspace of $\mathbb{F}_2^n$. The basis vectors of this vector subspace are represented as the rows of the generator matrix for the code. Consider a (n, k, d) linear code (C) with generator matrix $G_{\mathcal{C}} = [g_1, g_2, \ldots, g_k]^T$. To route using this code, peer X has to maintain links to $(k+1)$ superpeers with IDs $X_1, X_2, \ldots, X_{k+1}$, computed as follows:

$$X_i = \begin{cases} X \oplus g_i & 1 \leq i \leq k \\ X \oplus g_1 \oplus g_2 \oplus \ldots \oplus g_k & i = k+1 \end{cases} \tag{3}$$

Theorem 1. *Suppose we are using a (n, k, d) linear code $\mathcal{C}$ and each superpeer is maintaining $(k+1)$ routing links as specified in (3). In such an overlay, it is possible to route a query from any source to any destination codeword in less than or equal to $\frac{k}{2}$ routing hops.*

Proof. According to the definition of linear codes, $\vec{0} \in \mathcal{C}$, the rows of $G_{\mathcal{C}}$ (i.e., $g_1, g_2 \ldots g_k$) form a basis for the subspace $\mathcal{C}$, and $\mathcal{C}$ is closed under XOR operation. This implies, for any permutation $(i_1, i_2, \ldots, i_k)$ of $(1, 2, \ldots, k)$,

$$X \in \mathcal{C} \implies Y = (X \oplus g_{i_1} \oplus g_{i_2} \oplus \ldots \oplus g_{i_t}) \in \mathcal{C} \tag{4}$$

for $1 \leq t \leq k$, i.e., 2^k distinct codewords of $\mathcal{C}$ can be generated by XORing any combination of $1, 2, \ldots, k$ rows of $G_{\mathcal{C}}$ with X.

Suppose peer X (source) wants to route a message to peer Y (target), as defined in Equation 4. Now, X can route the message to any of $X_j = X \oplus g_{i_j}$ in one hop by using its routing links (see Equation 3). Suppose X routes to $X_1 = X \oplus g_{i_1}$. X_1 will evaluate Y as $Y = X_1 \oplus g_{i_2} \oplus \ldots g_{i_t}$. Note that Y is one hope closer to X_1 than X. X_1 can route the message to any of $X_1 \oplus g_{i_2}$, $X_1 \oplus g_{i_3}$,...,$X_1 \oplus g_{i_t}$ peers in one hop. In this way, the query can be routed from X to Y in exactly t-hops.

If $t \leq \frac{k}{2}$, then our claim is justified. Now let $t > \frac{k}{2}$. For this case, we can write $Y = X_{k+1} \oplus g_{i_{t+1}} \oplus g_{i_{t+2}} \oplus \ldots \oplus g_{i_k}$, according to the definitions of X_{k+1} and Y in Equation 3 and 4, respectively. Now using the $(k+1)$-th link X can route the message to X_{k+1} in one hop, and X_{k+1} can route the message to Y in $(k-t-1)$ hops. Hence for $t > \frac{k}{2}$ we will need at most $(k-t-1+1) \leq \frac{k}{2}$ hops. □

Given the above mentioned routing protocol, peer X will need a way to find the rows of $G_{\mathcal{C}}$, satisfying eqation (4), in order to route a message to peer Y. To deal with this problem, the standard form of the generator matrix, $G_{\mathcal{C}} = [I_k|B]$ is used in conjunction with the following theorem.

Theorem 2. *Suppose peer X wants to route to peer Y and needs to find the g_{i_j}'s satisfying Equation 4. If $G_{\mathcal{C}}$ is in standard form, then the first k-bits of $X \oplus Y$ have 1-bits in exactly $\{i_1, i_2, \ldots, i_t\}$ positions.*

Proof. Let $\theta = X \oplus Y$. By using the definition of Y in Equation 4, we get $\theta = g_{i_1} \oplus g_{i_2} \oplus \ldots \oplus g_{i_t}$. Since $G_{\mathcal{C}}$ is in the standard form, only i_j-th row of $G_{\mathcal{C}}$ (i.e., g_{i_j}) has a 1-bit in i_j-th bit position for any $i_j \in \{1, 2, \ldots, k\}$. Therefore, row g_{i_j} has to be present in the linear combination of the rows of $G_{\mathcal{C}}$ producing θ □

The extended Golay code, $\mathcal{G}_{24}$, (like other linear codes) can be defined using different sets of basis vectors (i.e., generator matrices). The generator matrix used in our implementation is $G_{24} = [I_{12}|B_{12}]$, where

$$B_{12} = \begin{bmatrix} 1 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 1 & 1 & 0 & 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 & 1 & 0 & 0 & 1 & 1 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 1 \end{bmatrix}$$

This generator matrix has two desirable properties:

1. $g_1 \oplus g_2 \oplus \ldots \oplus g_k = \vec{1}$
2. $\forall X \in \mathcal{G}_{24}, \qquad d(X, X \oplus g_i) = |X \oplus X \oplus g_i| = 8$

The first property implies that $X_{k+1} = X \oplus g_1 \oplus g_2 \oplus \ldots \oplus g_k = X \oplus \vec{1} = \bar{X}$, and according to Equation 4 $X_{k+1} = \bar{X} \in \mathcal{G}_{24}$. This construct is possible because $\mathcal{G}_{24}$ is a self-dual code. The second property ensures the minimum distance of 8 between any peer and any of its first k-neighbors $(X_1, \ldots, X_k)$. These two properties influence the routing strategy based on $\mathcal{G}_{24}$ as follows.

In any 3-party rendezvous architecture the negotiating middle entity can become a performance and a single point of failure unless appropriate measures are taken. Replication is employed to mitigate the performance problem, arising from the failure of a rendezvous peer. The information indexed at peer Y is replicated at peer $\bar{Y}$. The choice of $\bar{Y}$ as the replica for Y can be justified by the following observations.

Firstly, $Y = X \oplus g_{i_1} \oplus \ldots g_{i_t} \implies \bar{Y} = X \oplus g_{i_{t+1}} \oplus \ldots g_{i_k}$ (using Property 1 and Equation 4). Thus, any path from X to Y and $\bar{Y}$ is disjoint. This increases fault-resilience and influences uniform distribution of query traffic, especially in cases where peer Y is holding a popular index. Secondly, for ensuring at most $\frac{k}{2}$ hop routing, a link to peer X_{k+1} must be maintained. Since $X_{k+1} = \bar{X}$, the same link can be used for replication and routing purposes. Finally, as explained in Section 4.3, the 759 special octads in S_8 are likely to face a higher number of advertisements and queries than the 2576 umbral dodecads in U_{12}. On the other hand, *advSet* and *qSet* do not contain any codeword from S_{16} (the 759 special 16-sets). Since, $X \in S_8 \implies \bar{X} \in S_{16}$, we can shed the extra load on $X \in S_8$ by replicating to $\bar{X} \in S_{16}$. Note that, $X \in U_{12} \implies \bar{X} \in U_{12}$

As depicted in Figure 3 (step 5), the routing algorithm will always be subjected to a set of target peers instead of just one target. The routing algorithm can reduce a significant portion of routing hops by utilizing the shared common paths to different targets. Algorithm 1 presents a psedocode for multicasting a message from source peer X to a set of destination peers $\{Y_1, Y_2, \ldots, Y_u\}$.

The pseudocode presented in Algorithm 1 is a simplified version of the routing algorithm used in the simulator. It should be noted that the *msg* parameter contains a field named *msg.hops*, which is incremented at each hop. The routing of a message is suspended, if *msg.hops* reaches a value of $\frac{k}{2} + 2$. Suppose, peer Y has failed, and a query targeted towards Y reaches one of its neighbors $Y_i (= Y \oplus g_i)$. Y_i can route the query to $\bar{Y}$ in two hops as $\bar{Y} = \bar{Y_i} \oplus g_i$, and $\bar{Y_i}$ is one hop away from Y_i. The maximum length of a path between any two peers is $\frac{k}{2}$. Hence, in the presence of failures, a maximum of $\frac{k}{2} + 2$ hops will be required to reach any peer or its replica.

Algorithm 1 $X.route(msg, \mathcal{Y})$

Inputs:
    msg: Message e.g. search, advertise, join, etc.
    $\mathcal{Y}$: $\{Y_1, Y_2, \ldots, Y_u\}$ set of target peers
Externals:
    k: Dimension of the self-dual linear code
    $X_1, \ldots, X_{k+1}$: $(k+1)$ neighbors of X {*see (3)*}
{*update* $\mathcal{Y}$}
for each $Y_i \in \mathcal{Y}$ **do**
  if $X = Y_i$ **then**
    $\mathcal{Y} \leftarrow \mathcal{Y} - \{Y_i\}$
    process-message(msg)
  else if $d(X, Y_i) > \frac{k}{2} \bigvee$
      $(d(X, Y_i) = 1 \wedge !isAlive(Y_i))$ **then**
    $\mathcal{Y} \leftarrow (\mathcal{Y} - \{Y_i\}) \cup \{\bar{Y}_i\}$
  end if
end for
{*find suitability of each neighbor as next hop*}
$\mathscr{R} \leftarrow \{T_1, \ldots, T_{k+1} | \ \mathcal{T}_i \subseteq \mathcal{Y} \wedge$
      $Y \in \mathcal{T}_i \implies X_i$ is alive and on $X \rightsquigarrow Y\}$
{*do actual routing*}
while $\mathcal{Y}$ not empty **do**
  find s s.t. $\forall \mathcal{T}_i \in \mathscr{R}, |\mathcal{T}_s| \geq |\mathcal{T}_i|$
  if no such s exists **then**
    break {*remaining peers in* $\mathcal{Y}$ *are not reachable*}
  end if
  $\mathscr{R} \leftarrow \mathscr{R} - \{\mathcal{T}_s\}$
  $\mathcal{Y} \leftarrow \mathcal{Y} - \mathcal{T}_s$
  $X_s.route(msg, T_s)$
end while

Table 1: Distance distribution of orbits from octads and dodecads

	S_8	U_{12}		S_8	U_{12}
S_3	$5:21$		S_{11}	3:1,5:2	$5:16$
S_4	$4:5$		T_{11}	$5:5$	$3:1, 5:15$
S_5	3:1,5:20		U_{11}		$1:1$
S_6	$2:1$	$6:16$	S_{12}	$4:1$	$4:4, 6:48$
U_6	$4:6$	$6:18$	T_{12}		$4:6, 6:40$
S_7	$1:1$		U_{12}		$0:1$
U_7	3:1,5:15	$5:6$	X_{12}	$4:3$	$6:64$
S_8	$0:1$		P_{12}		$2:1, 6:55$
T_8	$2:1$	$6:42$	S_{13}	$5:3$	$5:16$
U_8	$4:4$	$4:2, 6:32$	T_{13}	$5:1$	$3:1, 5:15$
S_9	$1:1$		U_{13}		$1:1$
T_9	3:1,5:7	$5:14$	S_{14}		$6:56$
U_9	$5:12$	$3:1, 5:9$	T_{14}		$4:4, 6:42$
S_{10}	$2:1$	$6:56$	U_{14}		$2:1, 6:45$
T_{10}	$4:2$	$4:4, 6:42$			
U_{10}		$2:1, 6:45$			

4.3 Mapping Patterns to Codewords

This section presents the algorithms for finding $qSet(Q)$ and $advSet(P)$. The discussion in this section is specific to $\mathcal{G}_{24}$. Special consideration is required to adopt these algorithms for other linear codes.

As explained in Section 3.2, any pattern Q of length 24 belongs to one of the 49 orbits w.r.t. $\mathcal{G}_{24}$. Any vector in a given orbit has the same distance properties as listed in Table 1. In this table, the construct $d : x$ stands for distance (d) and number of codewords(x) at distance d. For vectors in a given orbit, we have listed only the number of octads and dodecads within distance 5 and 6, respectively.

The number of one bits in a query or advertisement is restricted to the range of 3 to 14. The reason behind this restriction can be justified by observing the property of Bloom filters. As discussed in Section 3.3, $33 - 50\%$ bits of a well-designed Bloom-filter are 1. Therefore, for a 24-bit chunk from a Bloom-filter $8 - 12$ bits are expected to be 1. Queries having fewer than three 1-bits are too generic, and are likely to match a large number of advertisements.

Due to this restriction, we only need to consider the octads and dodecads in $qSet$ and $advSet$ calculation. Most queries (involving $3 - 8$ 1-bits) are closer to the octads than the dodecads or the 16-sets. This results in a bias for the $qSet$ (and eventually the $advSet$) to have an octad:dodecad ratio higher than $2 : 7 (\approx 759 : 2576)$. To reduce the effect of this bias, the codewords in S_{16} are adopted as replicas of the octads (S_8). This helps in reducing the volume of query traffic at the octads.

Algorithm 2 find $qSet(Q)$

Input: $Q \in \mathbb{F}_2^{24}$
External: τ controls size of $qSet$ and $advSet$
Returns: $\mathcal{Q} \subset \mathcal{G}_{24}$
$\mathcal{Q} \leftarrow \{Y | (Y \in S_8 \wedge d(Y, Q) \leq 5) \bigvee (Y \in U_{12} \wedge d(Y, Q) \leq 6)\}$
if $|\mathcal{Q}| < \tau$ **then**
  $\mathcal{Q}' \leftarrow \emptyset$
  for each $Y \in \mathcal{Q}$ **do**
    $\mathcal{Q}' \leftarrow \mathcal{Q}' \cup \{Z | Z \in neighbors(Y) \wedge |Z| = 12\}$
  end for
  $\mathcal{Q} \leftarrow \mathcal{Q} \cup \mathcal{Q}'$
end if
return $\mathcal{Q}$

Pseudocodes for finding $qSet$ and $advSet$ are presented in Algorithm 2 and Algorithm 3, respectively, in light of the preceding discussion. The algorithm for finding $qSet$ starts with finding the set $\mathcal{Q}$ of the octads and dodecads that are within distance 5 and 6, respectively, from the query pattern Q. If $\mathcal{Q}$ contains fewer than τ codewords, then it is appended with the dodecads[5] that are reachable in one hop from the current members of $\mathcal{Q}$. The average size of $advSet$ is inversely proportional to the value of τ. Hence, τ can be tuned to achieve a desirable ratio of search and

[5]Octads are not selected for load-balancing purpose.

advertisement traffic. For our experiments, $\tau = 20$ is chosen, which fixes the average size of $qSet$ and $advSet$ to 25 and 28, respectively. A lower value of τ can be used if the anticipated number of queries is much higher than the number of advertisements.

The algorithm for finding $advSet$ has two stages. The first stage (lines 3-6) is to find set $\mathscr{P}$ of $qSet$s that will be searched by all possible queries matching the advertised pattern P. Now the problem is to find a small (preferably minimum) set of codewords $\mathscr{A}$ such that $\mathscr{A}$ contains at least one element from each set in $\mathscr{P}$. This is essentially the *minimum hitting set* problem, which in turn, is equivalent to the *minimum set cover* problem. To find $\mathscr{A}$, we have applied the greedy heuristic as presented in [10].

Algorithm 3 find $advSet(P)$

1: Input: $P \in \mathbb{F}_2^{24}$
2: Returns: $\mathscr{A} \subset \mathcal{G}_{24}$
3: $\mathscr{P} \leftarrow \emptyset$
4: **for** each Q s.t. $Q \wedge P = Q \wedge |Q| \geq 3$ **do**
5: $\quad \mathscr{P} \leftarrow \mathscr{P} \cup \{qSet(Q)\}$
6: **end for**
$\{$*use greedy heuristic to find minimum hitting set*$\}$
7: $\mathscr{A} \leftarrow \emptyset$
8: **while** $\mathscr{P}$ not empty **do**
9: $\quad$ find Y s.t. Y is in maximum no. of sets $\mathscr{S} \in \mathscr{P}$
10: $\quad \mathscr{A} \leftarrow \mathscr{A} \cup Y$
11: $\quad \mathscr{P} \leftarrow \mathscr{P} - \{\mathscr{S} | \mathscr{S} \in \mathscr{P} \wedge Y \in \mathscr{S}\}$
12: **end while**
13: **return** $\mathscr{A}$

4.4 Analysis

In this section, we estimate the expected number of visited superpeers during an advertisement or a search process. Let r be the number of subnets present in the system. Assume that an (n, k, d) linear code $\mathcal{C}$ is used. Let $|\mathscr{A}|$ and $|\mathcal{Q}|$ be the average size of $advSet$ and $qSet$, respectively. $\gamma_{\mathcal{Q}}$ and $\gamma_{\mathscr{A}}$ stands for the fraction of routing hops reduced due the presence of multicasting during search and advertisement, respectively. If N is the total number of superpeers in the system, then the number of superpeers in a subnet is $2^k \approx \frac{N}{r}$. Now, according to Theorem 1 it will require $\frac{k}{2} \approx \frac{1}{2}\log_2 \frac{N}{r}$ hops to route a message within a subnet. Consequently, considering the use of voting algorithm, the expected number of hops required for routing an advertisement is computed to be

$$H_{\mathscr{A}} = \frac{1}{2}\left\lfloor \frac{r}{2} + 1 \right\rfloor (1 - \gamma_{\mathscr{A}})\,|\mathscr{A}| \log_2 \frac{N}{r} \tag{5}$$

Similarly, the expected number of routing hops for routing a search message can be computed as

$$H_{\mathcal{Q}} = \frac{1}{2}\left\lfloor \frac{r+1}{2} \right\rfloor (1 - \gamma_{\mathcal{Q}})\,|\mathcal{Q}| \log_2 \frac{N}{r} \tag{6}$$

Now, we can compute an upper bound on $|\mathcal{Q}|$ as follows. The number of points covered by a codeword in $\mathcal{C}$ is given by

$$\lambda = \sum_{i \leq e} \binom{n}{i}$$

where $e = \frac{d}{2}$. Suppose, we want to discover all the advertisements within distance $t(= e+f)$ from a query pattern. We can imagine a sphere of radius t around the query pattern Q. We want to count the number of smaller spheres (around each codeword) required to cover the larger sphere around Q. This number will be maximum when $d(Q, C) = e$

for some codeword C. In that case, at least 50% of the points from the sphere around C will fall within the sphere around Q. Thus, the bound on $|\mathcal{Q}|$ can be computed as

$$|\mathcal{Q}| \leq \frac{\sum_{i \leq t} \binom{n}{i}}{\frac{1}{2} \sum_{j \leq e} \binom{n}{j}}$$

Clearly, for small values of $f(= t - e)$ this bound will be a small constant. $|\mathcal{Q}|$ will be smaller with larger values of d. Hence, we can infer from Equation 6 that $H_{\mathcal{Q}}$ is a logarithmic function of N. This claim has also been validated by our simulation results. However, a similar bound cannot be obtained for $|\mathscr{A}|$. Our experiments with the extended Golay code demonstrate that the average value of $|\mathscr{A}|$ is 28, and this value varies from 7 to 52 for patterns with weight 5 and 14, respectively.

4.5 Mapping Codewords to Superpeers

So far, it is assumed that each subnet is saturated, i.e., has close to (2^k) superpeers, and each superpeer is responsible for an unique codeword. But, this is not a practical assumption. In this section, we present a way of partitioning the codeword space, and assigning multiple codewords to a superpeer.

An (n, k, d) linear code $\mathcal{C}$ has k information bits and $(n - k)$ parity check (or redundant) bits. The k information bits correspond to the identity matrix, I_k, part of the generator matrix $G_{\mathcal{C}}$, and uniquely identifies each of the 2^k codewords present in $\mathcal{C}$ (consider $X = \vec{0}$ in Equation 4). The codewords can be partitioned using a logical binary partitioning tree with a height of at most k. At i-th level of the tree, partitioning takes place based on the presence of g_i (the i-th row of $G_{\mathcal{C}}$) in a codeword. Figure 4 presents an example. Each superpeer is assigned a leaf node in this tree and takes responsibility for all the codewords having that particular combination of g_is. The routing table entries at each superpeer are set to point to the appropriate superpeer responsible for the corresponding codeword. Figure 4 illustrates the routing table entries for superpeer $X = g_1 \oplus g_3 \oplus g_6 \oplus g_9$ with an equivalent prefix of $g_1 \bar{g}_2 g_3 \bar{g}_4$.

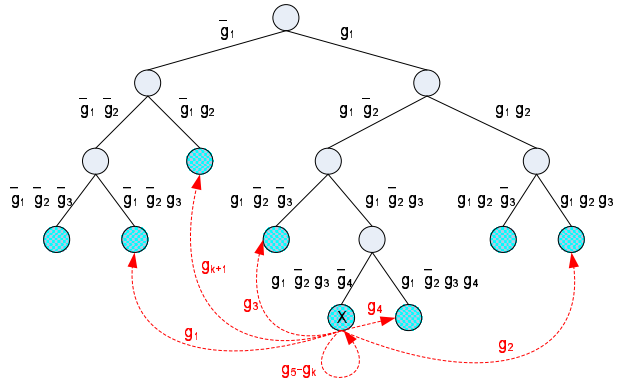


Figure 4: Logical binary partitioning tree for assigning codewords to superpeers. The routing table entries for peer X are also presented.

It is worth noting that the partition process presented in this section has some level of similarity with the Kademlia [21] protocol. Both techniques partition the key space based on the shortest unique prefix of alive peers. In Kademlia, each peer keeps track of a set of peers in every maximal subtree that does not contain it. In contrast, we keep track of a specific peer in every maximal subtree that does not contain it. In addition, the concept of linking to the complemented node $X_{k+1} = \bar{X}$ is not present in the Kademlia routing protocol.

In order to incorporate the concept of partitioning codeword space in Algorithm 1, the comparison, $X = Y_i$, in line 4 should be replaced with $Y_i \in \aleph$, where $\aleph$ represents the set of codewords managed by peer X.

4.6 The Join Process

The first superpeer in the system begins with a random codeword, and all entries in its routing table points to itself. A new superpeer joins the system by taking over a part of the codeword space from an existing peer, say, with codeword X. Assume that the string representation of $X = \rho_1 \cdot \rho_2 \ldots \rho_t \cdot \rho_{t+1} \ldots \rho_k$ and the prefix in peer X has t terms. Here, ρ_i is g_i or $\bar{g_i}$, based on the presence or absence of the i-th row in the formation of X. Peer X extends its prefix by one term and takes responsibility of all the codewords starting with prefix $\rho_1 \cdot \rho_2 \ldots \rho_t \cdot \rho_{t+1}$. The joining peer chooses a codeword, say Y, conforming to prefix $\rho_1 \cdot \rho_2 \ldots \rho_t \cdot \bar{\rho}_{t+1}$ and by selecting a random combination for the rest of the $(k - t - 1)$ rows from $G_{\mathcal{C}}$.

Routing table entries in peer X remain unchanged. Peer Y has to construct its routing table using the routing information from peer X. During this process, two situations can arise. First, the length of the prefix for peer X_i can be greater than t. In this case, Y has to lookup and contact the peer responsible for codeword $Y_i (= Y \oplus g_i)$. Peer Y requires at most $(\frac{k}{2} - t)$ hops to reach peer Y_i via peer X_i. For the second case, the length of the prefix for peer X_i is less than or equal to t. In this case, peer Y sets $Y_i = X_i$ and sends a join message to peer X_i. Peer X_i handles a join message by updating its routing table entry for link $X_t (= X_i \oplus g_t)$ with the address of peer X or peer Y depending on the presence of g_t in X_i.

To reduce the possibility of unbalanced partitioning of the codeword space, a joining peer should crawl the neighborhood of the seed peer, until a minima is reached, and join the minima. By minima we refer to a peer having a prefix of length equal to or less than that of any of its neighbors.

4.7 Handling Peer Failure

The Failure of a peer (say Y) does not hamper the routing process as long as its replica ($\bar{Y}$) is alive. This way temporary failures (or disconnections) of superpeers are automatically handled. Measures adopted to deal with permanent (long term) failures are discussed below.

Failure of peer Y will be detected by one of its neighbors, say Y_i. To avoid unbalanced partitioning of the codeword space, Y_i should crawl its neighborhood until a maxima, say Z, is reached. By maxima, we refer to a peer having a prefix of length equal to or greater than that of any of its neighbors. Clearly, if Z has t terms in its prefix, then $Z_t (= Z \oplus g_t)$ will be a neighbor of Z having a prefix of length t. Z will reassign its portion of codewords to Z_t; will replace itself with Z_t from the routing tables of all of its neighbors; and finally will rejoin the system as Y. Z_t has to reduce its prefix string by one, in order to accommodate the changes.

To handle the failure of leaf peers, we adopt the hybrid (soft state/hard state) technique as presented in [6]. A leaf peer connects to a superpeer for publishing the meta-information about its shared content. A superpeer uses *soft-state* registration mechanism for tracking the failure of a leaf-peer, and explicitly removes (i.e., hard-state) the patterns, advertised by the failed leaf-peer, from the superpeer topology. This hybrid technique can handle churn problem in leaf peers and reduces traffic due to periodic re-advertisement in the superpeer network.

5 Experimental Evaluation

In this section, we present simulation results to validate the concept presented in this work. Our prototype implementation simulates keyword search for a music sharing P2P system. We focus on three aspects of the system: routing efficiency, search completeness, and fault-tolerance.

5.1 Simulation Setup

The music information database used in our simulation contains about 46,500 records of <*song-title,artist*> pairs, extracted from *http://www.leoslyrics.com/* (an online database of song lyrics). For each <*song-title,artist*> pair, we constructed a Bloom-filter with parameters $m = 120$ and $k = 3$. Each Bloom filter represents the set of trigrams

extracted from a <*song-title,artist*> pair. The average number trigrams present in a record is $n \approx 16$.

The routing algorithm used in the simulator is based on the pseudocode presented in Algorithm 1. The $qSet$ and $advSet$ are calculated as discussed in Section 4.3. We have not incorporated the codeword to superpeer mapping algorithm, discussed in Section 4.5, to this preliminary version of the simulator. This implies a complete binary partitioning tree, where each superpeer is responsible for a single codeword. With this setup, the maximum number of hops will be required while routing a message. Failures are simulated by deactivating randomly selected superpeers and by inhibiting any traffic through them.

5.2 Impact of Query Content on Search Completeness

The completeness of a search result is measured as the percentage of advertised patterns, matching the query string, that were discovered by the search. To form a query we first randomly choose a <*song-title,artist*> pair, then take a certain percentage (say β) of trigrams from all the trigrams present in that pair, and then create the query Bloom filter (with $m = 120$ and $k = 3$) from these trigrams. For this experiment we have used a system with 5 ($= \frac{120}{24}$) subnets corresponding to about 20,000 superpeers.

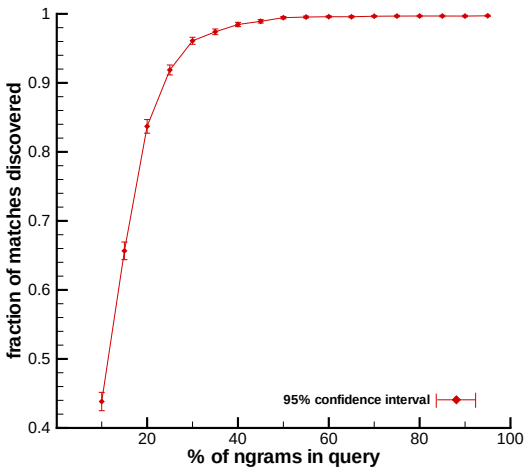


Figure 5: Effect of the Information content of a query on search completeness.

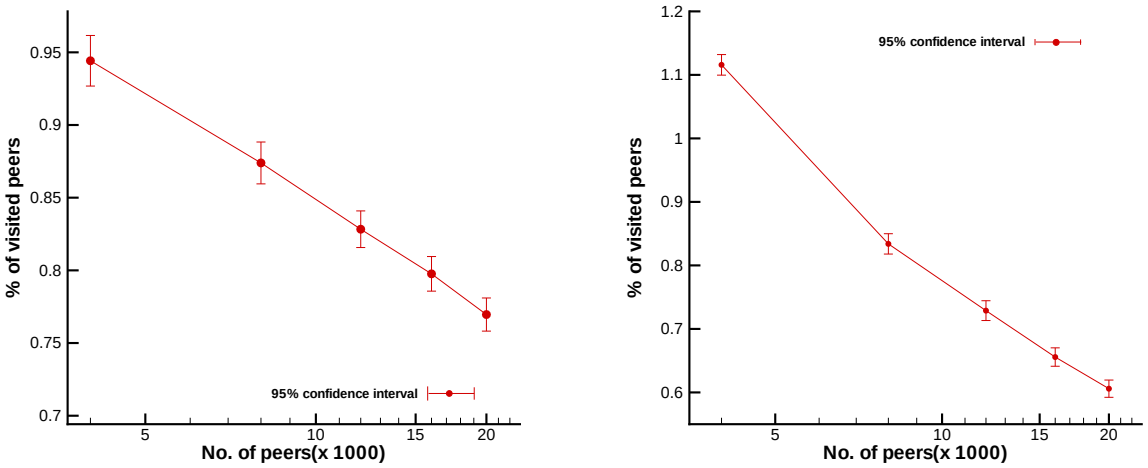
Figure 5 presents search completeness as a function of the percentage of trigrams selected for the construction of the query Bloom filter. In this experiment, β is varied from $10\% - 95\%$ in 5% steps. For each step, we performed 5000 random queries and gathered statistics. The number of 24-bit chunks having at least three 1-bits decreases as lower percentage of trigrams are taken from the original advertisement. Hence, read quorum for the voting algorithm could not be met for lower values of β, which decreases the level of completeness. It can be observed from Figure 5 that a completeness level of around 97% is achieved for $\beta = 33\%$ and only 2% increase is achievable for higher values of β. For the subsequent experiments we have used $\beta = 33\%$.

5.3 Scalability and Routing Efficiency

The impact of network size on routing efficiency is considered in this section. For this experiment, the number of subnets (r) in the system is varied from 1 to 5, with each subnet containing approximately 4000 ($\approx 2^{12}$) superpeers. The corresponding variation in length of the patterns that can be advertised in the system is from 24-bits to 120-bits. For a system with r subnets, we have taken $\lfloor \frac{r+1}{2} \rfloor$ chunks (each 24-bit long), with at least three 1-bits, from the first r chunks of the query Bloom filter. If more than $\lfloor \frac{r+1}{2} \rfloor$ chunks had three 1-bit, then we preferred the chunks with higher number of 1-bits. For each instance of the system we have advertised all the tuples; performed 5000 random

queries and gathered statistics.

It should be noted that, with this setup, search completeness was not same for all values of r. The k different hash functions of a Bloom filter are expected to evenly distribute the 1-bits over all the chunks. Yet, for some rare cases $\lfloor \frac{r+1}{2} \rfloor$ 24-bit chunks from the first r chunks of the Bloom filter did not contain more than three 1-bits. During advertising or querying such Bloom filters, the read/write quorum for the voting algorithm could not be fulfilled. To remove the impact of such patterns on the simulation results, we have refrained from advertising or querying those patterns.



(a) Advertisement

(b) Query

Figure 6: **Routing efficiency: percentage of visited peers as function of total peer.**

Figure 6 presents the percentage of visited peers as a function of the number of total peers in the system for the advertisement (Figure 6(a)) and query (Figure 6(b)) process. For $r = 1$, i.e., with about 4000 super peers each query or advertisement is handled by a single subnet. As the number of subnets increases, not all queries or advertisements are sent to all subnets. Hence the reduction in the percentage of visited peers. The query/advertisement load is distributed over different subnets in the system. From the curves in Figure 5 and 6, we can observe that for a system with 5 subnets we can discover about 97% of the matching patterns by visiting only 0.6% of the superpeers in the system. Yet, in the query string, we need to specify only 33% of the trigrams from the advertised pattern.

5.4 Effectiveness of Multicast Routing

The routing algorithm described in Section 4.2 routes a message to multiple targets simultaneously. This design choice saves a portion of the routing hops that might have occurred if we had used pair-wise routing. Figure 7 shows the reduction in routing hops (rrh) calculated as

$$rrh = \left(1 - \frac{\text{no. of hops with multicast routing}}{\text{no. of hops for pair-wise routing}}\right) \times 100$$

The reduction in routing hops takes place within a subnet and so does not depend on the number of subnets present in the system. For this experiment we have advertised the patterns in a system with $r = 5$. For advertisements in each subnet we recorded the number of targets and required number of hops. The bar chart in Figure 7 displays the average rrh for groups of 5 targets, i.e. $6-10$, $11-15$, etc. The denominator for rrh equation (i.e., *no. of hops for pair-wise routing*) was calculated as: $\sum_{Y \in advSet(P)} d(\phi(X), \phi(Y))$, where X is the source peer and $\phi(X)$ returns the

k-bits of a codeword corresponding to the I_k part of G_C. Figure 7 presents rrh for advertisements only. For queries, the savings is much higher in the range of $72 - 82\%$ because many members of $qSet$ are immediate neighbors.

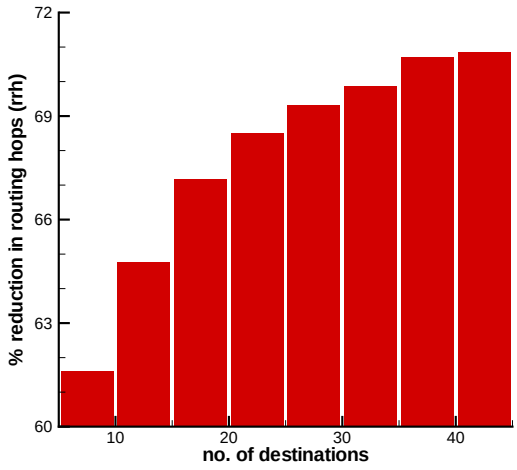


Figure 7: Effectiveness of simultaneous routing to multiple targets: reduction in routing hops as a function of the number of targets.

5.5 Fault Tolerance

In this section, we analyze the robustness of our system in the presence of simultaneous failure of superpeers. We have conducted the experiment in a network with $r = 5$. We first advertise the patterns and then deactivated $5\% - 45\%$ randomly selected superpeers in 5% steps. After each step, 5000 random queries were performed and statistics were gathered. There were no rearrangement in topology to redistribute the responsibility of failed peers to an existing peer. Only the immediate neighbors of a failed peer have the knowledge of the failure. This setup suppresses the effect of recovery mechanism and allows us to observe the effectiveness of replication and multi-path routing in presence of simultaneous peer-failure.

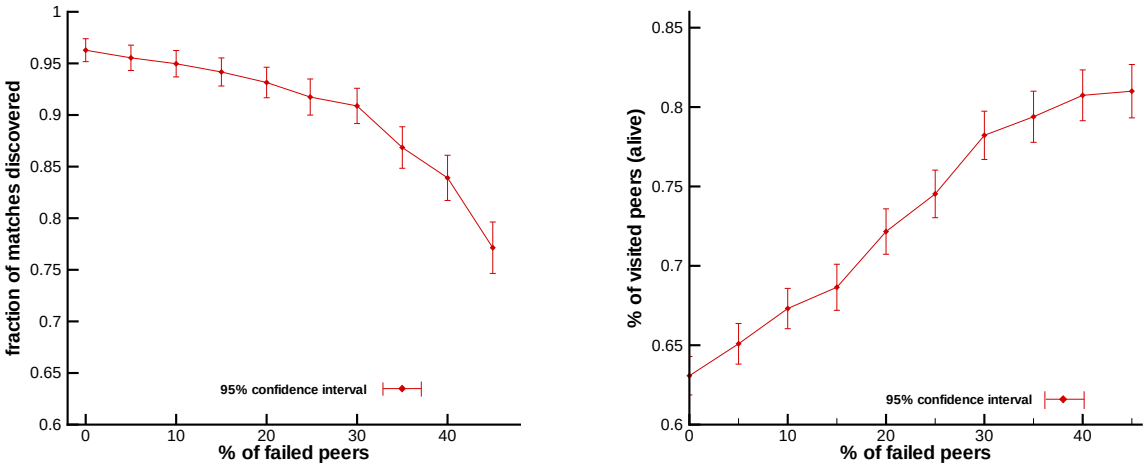
We focus on the impact of simultaneous failures on two metrics: search completeness (Figure 8(a)) and query routing efficiency (Figure 8(b)). Number of patterns lost from the superpeer network increases with the number of failed peers. This results into the decrease in search completeness.

The failure of a peer can not be detected until reaching a neighbor of the failed peer. The percentage of visited peers increases as many hops are wasted in trying to reach a failed peer and its replica, which may also have failed. However, the good thing is that in such cases two extra hops are required to reach the replica, as discussed in Section 4.2.

It can be observed from the results that the system can achieve a high level of search completeness (about 90%), even when 30% of the peers have failed. Even with this level (30%) of failure the percentage of visited peers raises to 0.78% from 0.6% (no failure case). This was possible because of the existence of multiple paths connecting any two superpeers within a subnet.

6 Conclusion and Future Work

In this paper we have presented a partially decentralized architecture, and a protocol for structured and flexible routing within the superpeer network. Subset search has application in many Distributed systems. We have provided an efficient solution to this challenging problem. As demonstrated by the simulation results, only 0.6% of the superpeers are needed to be visited in order to resolve a query and can discover about 97% of the advertised patterns



(a) Search completeness

(b) Query routing traffic

Figure 8: **Fault resilience.**

matching a query. For achieving this level of completeness, the query needs to contain only 33% of the trigrams from an advertised pattern that it should match against. We can achieve a high level of fault-resilience by using replication and redundant routing paths. Even with 30% failed superpeers, we can attain a high level of search completeness (about 90%) by visiting only 0.78% of the superpeers. Queries and advertisements can be routed to target peers in $O(logN)$ hops and by using $O(logN)$ links.

The originality of our approach lies on the application of coding theoretic construct for solving the subset matching problem in distributed systems. We believe that this concept will aid in solving a number of other problems pertaining to P2P networking research, including P2P databases, P2P semantic search and P2P information retrieval.

In light of our experience with the prototype and current design choices, we would like to extend this work in the following directions.

A major reason behind selecting the Extended Golay code for our implementation was its size ($= 2^{12} = 4096$ codewords). The proposed algorithm for finding $qSet$ requires to find all codewords within a specified distance from a given 24-bit vector. We had to use linear search for this step, as no efficient algorithm for this task is known. Use of linear search is not feasible with larger codes with tens of thousands of codewords. It is certainly a very interesting issue that we will further investigate.

Use of the voting algorithm introduces a lot of redundancy to the system. This type of redundancy is good for fault resilience. But, with larger number of subnets in the system, this will become an overhead. With careful observation, it can be realized that selecting appropriate subnets for advertisements and queries is essentially the same problem as that of finding $advSet$ and $qSet$ satisfying Equation 2. Hence, with larger number of subnets we can reduce redundancy in the number of selected subnets by using some linear code with large distance, e.g. 1st order Reed-Muller codes. Another possibility is to use a set of locality preserving hash functions to map the query and advertisements to some of the subnets, such that there exists a common subnet with high probability.

As mentioned in Section 5, read/write quorum can not be met for a Bloom filter exhibiting non-uniform distribution of 1-bits. Although the percentage of such cases is very low, we can avoid or minimize these cases by adopting a slightly different implementation of the Bloom filters [8]. For this variant of Bloom filters, the m bits are partitioned into $\frac{m}{k}$ chunks, and the i-th hash function sets the bits only within the i-th chunk. With a proper alignment of $\frac{m}{k}$ with the length of selected code, better results can be achieved. But for this setup, the allowable number of elements in a Bloom filter becomes an issue, requiring further investigation.

Finally, we would like to run experiments on a larger distributed testbed like PlanetLab, with a large number of participating sites and track the system's performance for various load levels and with a transient population of virtual hosts.

References

[1] Fasttrack peer-to-peer technology, http://www.fasttrack.nu/.

[2] Gnutella website, http://www.gnutella.com.

[3] R. Ahmed and R. Boutaba. Distributed pattern matching for p2p systems. In *Proc. of NOMS'06 (to appear)*, Online [http://bcr2.uwaterloo.ca/∼rboutaba/ Papers/Conferences/noms2006.pdf] 2006.

[4] A. Amir, E. Porat, and M. Lewenstein. Approximate subset matching with don't cares. In *Proc. of SODA*, pages 305–306, 2001.

[5] S. Androutsellis-Theotokis and D. Spinellis. A survey of peer-to-peer content distribution technologies. *ACM Computing Surveys*, 45(2):195–205, December 2004.

[6] M. Balazinska, H. Balakrishnan, and D. Karger. Ins/twine: A scalable peer-to-peer architecture for intentional resource discovery. In *Proceedings of the First International Conference on Pervasive Computing*, pages 195–210. Springer-Verlag, 2002.

[7] B. H. Bloom. Space/time trade-offs in hash coding with allowable errors. *Commun. of ACM*, 13(7):422–426, 1970.

[8] A. Broder and M. Mitzenmacher. Network applications of bloom filters: A survey. *Internet Mathematics*, 1(4):485–509, 2003.

[9] Y. Chawathe, S. Ratnasamy, L. Breslau, N. Lanham, and S. Shenker. Making gnutella-like P2P systems scalable. In *Proc. of ACM SIGCOMM*, pages 407–418, 2003.

[10] V. Chvtal. A greedy heuristic for the set covering problem. *Math. Oper. Res.*, 4:233–235, 1979.

[11] E. Cohen, A. Fiat, and H. Kaplan. Associative search in peer to peer networks: Harnessing latent semantics. In *Proc. of IEEE INFOCOM*, 2003.

[12] R. Cole and R. Harihan. Tree pattern matching and subset matching in randomized $o(n \log^3 m)$ time. In *Proc. of ACM STOC*, pages 66–75, 1997.

[13] J. Conway and N. Sloane. Orbit and coset analysis of the golay and related codes. *IEEE Transactions on Information Theory*, 36(5):1038–1050, September 1990.

[14] S. E. Czerwinski, B. Y. Zhao, T. D. Hodes, A. D. Joseph, and R. H. Katz. An Architecture for a Secure Service Discovery Service. In *Proc. of MOBICOM*, pages 24–35, 1999.

[15] D. K. Gifford. Weighted voting for replicated data. In *Proceedings of the seventh Symposium on Operating Systems Principles*, pages 150–162, 1979.

[16] M. J. E. Golay. Notes on digital coding. In *Proceedings of the IEEE*, volume 37, 1949.

[17] A. Jain, M. Murty, and P. Flynn. Data clustering: A review. *ACM Computing Surveys*, 31(3):264–323, September 1999.

[18] C. Lee and S. Helal. Protocols for service discovery in dynamic and mobile networks. *International Journal of Computer Research*, 11(1):1–12, 2002.

[19] M. Li, W. Lee, and A. Sivasubramaniam. Neighborhood signatures for searching p2p networks. In *Proc. of IDEAS*, pages 149–159, 2003.

[20] C. Lv, P. Cao, E. Cohen, K. Li, and S. Shenker. Search and replication in unstructured peer-to-peer networks. In *Proc. of ICS*, 2002.

[21] P. Maymounkov and D. Mazi. Kademlia: A peer-to-peer information system based on the XOR metric. In *Proc. of IPTPS*, pages 53–65, 2002.

[22] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Schenker. A scalable content-addressable network. In *Proc. of ACM SIGCOMM*, pages 161–172, 2001.

[23] C. Schmidt and M. Parashar. Enabling flexible queries with guarantees in p2p systems. *IEEE Internet Computing*, 8(3):19–26, 2004.

[24] I. Stoica, R. Morris, D. Liben-Nowell, D. R. Karger, M. F. Kaashoek, F. Dabek, and H. Balakrishnan. Chord: a scalable peer-to-peer lookup protocol for internet applications. *IEEE/ACM Trans. on Networking*, 11(1):17–32, 2003.

[25] D. Stutzbach and R. Rejaie. Characterizing unstructured overlay topologies in modern p2p file-sharing systems. In *Internet Measurement Conference*, October 2005.

[26] C. Tang, Z. Xu, and M. Mahalingam. pSearch: Information retrieval in structured overlays. In *ACM HotNets-I*, October 2002.

[27] D. Tsoumakos and N. Roussopoulos. Adaptive probabilistic search for peer-to-peer networks. In *IEEE Intl. Conf. on P2P Computing*, 2003.

[28] B. Yang and H. Garcia-Molina. Improving search in peer-to-peer networks. In *Proc. of ICDCS*, 2002.