Plexus: A Scalable Peer-to-Peer Protocol Enabling Efficient Subset Search

Reaz Ahmed and Raouf Boutaba, *Senior Member, IEEE*

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

Plexus has a *partially decentralized* architecture involving superpeers. It adopts a novel *structured routing* mechanism derived from the theory of *Error Correcting Codes (ECC)*. Plexus achieves better resilience to peer failure by utilizing replication and redundant routing paths. Routing efficiency in Plexus 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 Plexus to partial keyword search utilizing the extended Golay code.

***Index Terms*—Bloom filter, distributed pattern matching, error correcting codes, peer-to-peer search, structured overlay network.**

I. Introduction

GIVEN a list of patterns[1] $P_1, P_2, \ldots, P_a$ and a search pattern Q, the subset matching problem is to find all i such that $Q \subseteq P_i$, i.e., 1-bits of Q is a subset of the 1-bits in P_i. The Distributed Pattern Matching (DPM) introduced in [4] is the distributed version of subset matching problem, where the patterns $P_1, P_2, \ldots, P_a$ are distributed across a large number of networked nodes. State-of-the-art solutions for subset matching ([5], [15]) in centralized environment, hold linear relationship with the number of patterns to be matched against (i.e., a). An equivalent solution in a distributed environment will require flooding the network with search messages.

In large-scale distributed systems, such as, P2P content sharing, service discovery and P2P XML databases, it is unusual for a user to know the exact (complete) information about an advertisement. Instead, queries are based on partial knowledge about a target advertisement. Search problems in such distributed systems can be mapped to the DPM problem. Bloom filters can be used as the advertisement patterns (P_i) and query pattern (Q). As depicted in Fig. 1, an advertisement in P2P content sharing systems consists of a number of keywords describing the file being shared. For such systems, advertisement (or query) Bloom filter can be constructed using trigrams from advertised (or queried) keywords. Subset relationship between advertised and queried trigrams will hold for advertisement and query Bloom filters. For example, in Fig. 1 trigrams for the first query constitute a subset of the advertised trigrams; as a result query pattern Q_1 is a subset of the advertisement pattern P. On the other hand, trigrams from the second query do not correspond to any subset of the advertised pattern, and with high probability Q_2 will not be a subset of P.

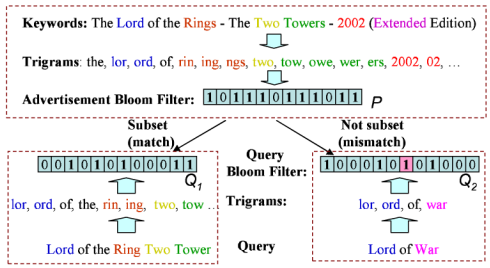


Fig. 1. Partial keyword matching using DPM.

For P2P database systems, as shown in Fig. 2, XML documents are used as advertisements and XPath is the most commonly used query language. In this case, path prefixes from an XML document or the XPath expression can be used as the *set elements* for Bloom filter construction (see Fig. 2). For most service discovery systems a service description is advertised as a set of attribute value pairs and a query for a service consists of a subset of the advertised attribute-value pairs. It is evident as shown in Fig. 2 that an efficient solution to the DPM problem will enable us to generate satisfactory solutions to the search problem in these three important application domains.

Existing P2P search techniques are based on either unstructured hint-based routing or structured Distributed Hash Table (DHT)-based routing ([31], [33], [36]). Neither of these two paradigms can provide satisfactory solution to the DPM problem. Unstructured techniques are not efficient in terms of the generated volume of search messages; moreover, no guarantee on search completeness is provided. Structured techniques, on the other hand, strive to build an additional layer on top of a DHT protocol for supporting partial-prefix matching. DHT-mechanisms cluster keys based on numeric distance. But, for efficient subset matching, keys should be clustered based on Hamming distance.[2] As a result, these solutions generate

Manuscript received February 18, 2007; revised November 05, 2007; approved by IEEE/ACM TRANSACTIONS ON NETWORKING Editor B. Levine. First published September 09, 2008; current version published February 19, 2009.

R. Ahmed is with the Bangladesh University of Engineering and Technology (BUET), Dhaka-1000, Bangladesh (e-mail:reaz@cse.buet.ac.bd)

R. Boutaba is with the School of Computer Science, University of Waterloo, Waterloo, Ontario, Canada N2L 3G1 (e-mail: rboutaba@uwaterloo.ca).

Digital Object Identifier 10.1109/TNET.2008.2001466

[1]We consider a pattern to be a bit-sequence of fixed width.

[2]Hamming distance between patterns X and Y (of same length) is calculated as $d(X, Y) = |X \oplus Y|$ = no. of bits on which they disagree.

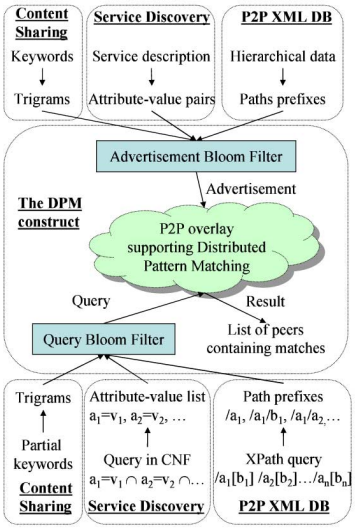


Fig. 2. Possible applications for Distributed Pattern Matching (DPM).

several independent DHT-lookups, each followed by local flooding, for resolving a single query.

We have identified the DPM problem and have proposed a hierarchical architecture, utilizing *semi-structured* [6] search in [4]. The contribution of this paper is a novel *structured* [6] routing mechanism for distributed subset search, named Plexus, based on the theory of *Error Correcting Codes (ECC)*. The novelty of the proposed approach lies in the use of Hamming distance based routing, in contrast to the numeric distance based routing adopted in traditional DHT-approaches. This property makes subset matching capability intrinsic to the underlying routing mechanism. Plexus uses patterns (like Bloom filters [9]) to summarize the identifying properties associated with a shared object. The number of routing hops and the number of links maintained by each indexing peer scales logarithmically with the number of peers. Plexus attains better resilience to peer failure using replication and redundant routing paths. The concept presented in this paper is supported with theoretical analysis, and simulation results obtained from the application of Plexus to partial keyword search utilizing the extended Golay code [22].

The rest of this paper is organized as follows. Section II compares Plexus with related work. Preliminaries on coding theory and Bloom filter are presented in Section III. Section IV explains the theoretical model of Plexus, while Section V presents the overlay topology construction and maintenance protocols. Simulation results, supporting our claims, are presented in Section VI. In Section VII we present a comparative performance evaluation of Plexus with possible alternatives. Finally, we conclude and outline our future research goals in Section VIII.

II. Related Work

Subset matching in distributed environments has been addressed in [4]. That work presented 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. DPMS uses Bloom filters as meta-information for routing, and relies on replication for fault-resilience. DPMS uses a don't care based aggregation scheme to reduce the volume of indexed information. DPMS requires $O(\log(N/\log N))$ additional hops for finding each match, after a group of $O(\log N)$ peers at the topmost level has been flooded. On the contrary, Plexus can discover all the matches by searching a limited number of superpeers.

Several research activities (including [8], [20], [26], [35], [38]) add a layer on top of DHT to support keyword search. Squid [35] adopts space-filling-curves to map similar keywords to numerically close keys, and uses Chord [36] for routing. It supports partial prefix matching queries (e.g., net^*). In contrast, Plexus supports true partial matching for queries like $^*net^*$. pSearch [38] aims to support full-text search. It uses Information Retrieval techniques, like vector space model and latent semantic indexing, on top of CAN [33]. Queries and data are represented by term vectors. Searches are performed in multidimensional Cartesian space. In that technique search performance degrades with the increase in dimensionality. In [20], a content is described as a set of attribute-value (AV) pairs. Each AV-pair is hashed and indexed individually with the underlying DHT-system. Though the system can search a content in one DHT-lookup using any subset of the advertised AV-pairs, advertisement overhead is very high, rendering the approach unsuitable for large networks of transient peers. MKey [26] uses Chord and offers restricted subset matching by advertising and searching specially chosen subsets of advertised keys and search keys, respectively.

Unstructured systems (flooding [1] and random walk [30]) can support partial keyword search, though the generated volume of query traffic does not scale with network size. Many research activities are aimed toward improving the routing performance of unstructured P2P systems by adopting routing hints. In [39] and [41], peers learn from the results of previous routing decisions, and bias future query routing based on this knowledge. In [12], routing is biased by peer capacity; queries are routed to peers of higher capacity with a higher probability. In [14], peers are organized according to common interests, and restricted flooding is performed in different interest groups. In [12], [29] and [41], 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.

Secure Service Discovery Service (SSDS) [17] and Twine [7], 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 hierarchal indexing and uses Bloom filters for representing service descriptions. SSDS suffers from a load-balancing problem and is vulnerable to the failure of higher level directory entities along the index hierarchy. Twine [7], on the other hand, uses a hierarchical naming scheme and relies on Chord as the underlying routing mechanism. Twine generates a set of substrings from the advertisement or query, computes keys for each substring, and uses these keys for DHT-lookups. Thus, the number of DHT-lookups increases with the number of attribute-value pairs in a name.

Several research works on P2P databases have adopted DHT-based structured P2P techniques, such as Chord [36], CAN [33] and Hypercubes [34], for routing. A number of these proposals, including [10], [11] and [19], rely on Chord as the underlying P2P substrate. On the other hand, hint-based (unstructured) routing is adopted by several research works including [18], [28] and [32].

Plexus 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 [7], [35], and the lack of scalability displayed by hint-based unstructured systems such as [1], [14] and [29].

III. Preliminaries

A. Linear Covering Codes

Let $\mathbb{F}_2^n$ define the linear space of all n-tuples (or vectors) over the finite field $\mathbb{F}_2 = \{0, 1\}$. A linear binary code of length n is a *subspace* $\mathcal{C} \subset \mathbb{F}_2^n$. Each element in $\mathcal{C}$ is called a *codeword*. A linear covering code is specified by using four parameters $(n, k, d)f$. Here, k is the dimension of the code. d is the minimum Hamming distance between any two codewords and n is the length of each codeword in bits. The *covering radius* f is the smallest integer such that every vector $P \in \mathbb{F}_2^n$ is covered by at least one $B_f(C_i)$. Here, $B_f(C_i) = \{P \in \mathbb{F}_2^n | d(P, C_i) \leq f\}$ is the *Hamming sphere* of radius f centered at codeword C_i.

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*, containing 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[3] of $G_{\mathcal{C}}$. The generator matrix for any linear code $\mathcal{C}$ can be expressed as,

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

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,

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

Here, $x \cdot c$ represents vector dot product over $\mathbb{F}_2$. 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 $\mathcal{C}$, $c \in \mathcal{C} \implies \bar{c} \in \mathcal{C}$, where $\bar{c}$ represents the bit-wise complement of c.

B. Extended Golay Code

The extended Golay code, $\mathcal{G}_{24}$, is a (24, 12, 8)4 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[4] distribution of this code is $0^1\ 8^{759}\ 12^{2576}\ 16^{759}\ 24^1$. 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*).

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

[4]The number of 1-bits in a pattern (say P) is known as its weight $(= |P|)$.

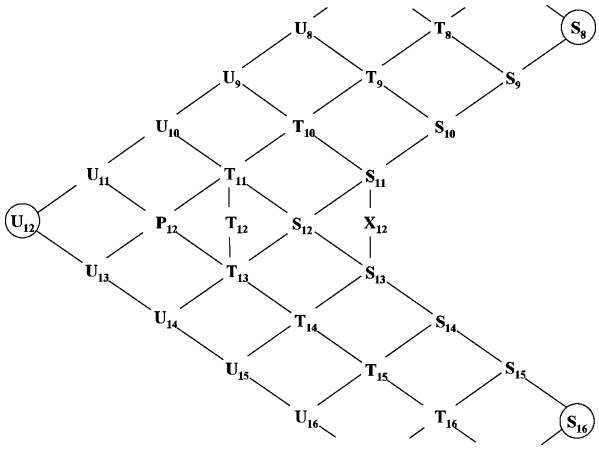


Fig. 3. Relationships among the orbits of the vectors in $\mathbb{F}_2^n$ w.r.t. to the codewords in $\mathcal{G}_{24}$. Circled orbits correspond to the codewords of $\mathcal{G}_{24}$.

Any vector in $\mathbb{F}_2^{24}$ can be categorized into 49 orbits w.r.t. $\mathcal{G}_{24}$ (see [16, Fig. 1]). These orbits are denoted as $S_w (0 \leq w \leq 24)$, $T_w (8 \leq w \leq 16)$, $U_w (6 \leq w \leq 18)$, P_{12} and X_{12}, where the subscript w denotes the weight of the vectors in that orbit. All vectors in a given orbit exhibit identical distance properties from the codewords in $\mathcal{G}_{24}$. Fig. 3 (a portion of Fig. 1 in [16]) 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 Fig. 3. Orbits S_8, U_{12} and S_{16} correspond to the *special octads*, *umbral dodecads* and *16-sets*, respectively.

C. Bloom Filters

A Bloom filter [9] is a compact data-structure used to represent a set. However, the set membership test operation may result into false (erroneously) positives with a small probability. An m-bit array is used to represent a Bloom filter. $\hbar$ different hash functions need also 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 $\hbar$ hash functions to obtain $\hbar$ positions in the bit-array and corresponding $\hbar$ 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 $\hbar$ hash functions and corresponding $\hbar$ positions in the bit-array are checked. If any of these $\hbar$ 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 the $\hbar$ bits equal to 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 ω-element set, is calculated as $\epsilon = (1 - (1 - (1/m))^{\omega\hbar})^{\hbar}$; ϵ is minimized when $\hbar = \ln 2(m/\omega)$. For example, with $m/\omega = 6$ and $\hbar = 3$, $\epsilon \approx 0.06$. For a well designed Bloom filter about 33$\sim$50% of the bits are equal to 1.

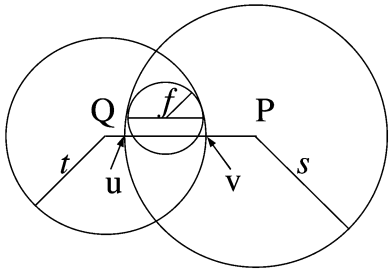


Fig. 4. Bound on $d(P, Q)$.

IV. Theoretical Model of Plexus

A. Core Concept

The originality of this work lies in the use of Hamming distance based clustering of pattern-space, as opposed to numeric distance utilized by DHT-techniques. For a $(n, k, d)f$ code $\mathcal{C}$, a codeword $C_i \in \mathcal{C}$ represents all the patterns in $B_f(C_i)$. Each peer is assigned one (or more, as explained later) codeword (C_i) and becomes responsible for all the patterns within its Hamming sphere $B_f(C_i)$. The basic concept is to map a query Q to a set of codewords $(\mathcal{Q}(Q) \subset \mathcal{C})$ and to map an advertised pattern P to another set of codewords $(\mathcal{A}(P) \subset \mathcal{C})$, such that $\mathcal{Q}(Q)$ and $\mathcal{A}(P)$ has *at least one* codeword in common whenever the 1-bits of Q constitute a subset of the 1-bits in P. Mathematically,

$$Q \subseteq P \Longrightarrow \mathcal{Q}(Q) \cap \mathcal{A}(P) \neq \emptyset. \tag{2}$$

Now the challenge is to compute $\mathcal{A}(P)$ and $\mathcal{Q}(Q)$, and to devise a mechanism for routing within the overlay. Coding theory literature does not provide any straight forward way to calculate $\mathcal{A}(P)$ and $\mathcal{Q}(Q)$, satisfying (2). We present the algorithm for computing $\mathcal{A}(P)$ and $\mathcal{Q}(Q)$ in Section IV-B, and the routing algorithm is presented in Section IV-C.

B. Computing $\mathcal{A}(P)$ and $\mathcal{Q}(Q)$

This section presents the algorithms for computing $\mathcal{Q}(Q)$ and $\mathcal{A}(P)$. Given a code $\mathcal{C}$, a trivial way of computing $\mathcal{A}$ and $\mathcal{Q}$ is to use list decoding; i.e.,

$$\mathcal{A}(P) = B_s(P) \cap \mathcal{C} = \{Y | Y \in \mathcal{C} \wedge d(Y, P) \leq s)\} \text{ and}$$
$$\mathcal{Q}(Q) = B_t(Q) \cap \mathcal{C} = \{Y | Y \in \mathcal{C} \wedge d(Y, Q) \leq t)\}$$

for positive integers s and t. Now we want to compute the minimum $d(P, Q)$ without violating (2). If the covering radius of $\mathcal{C}$ is f then the Hamming sphere of radius f around any arbitrary point should contain at least one codeword. Hence, according to Fig. 4

$$d(u, v) = s + t - d(P, Q) \geq 2f$$
$$\Rightarrow d(P, Q) \leq s + t - 2f. \tag{3}$$

In other words, if we advertise to all the codewords in $B_s(P) \cap \mathcal{C}$ and search all the codewords in $B_t(Q) \cap \mathcal{C}$ then any subset Q of P within distance $d(P, Q) \leq s + t - 2f$ can be discovered.

We define *query stretch* as the maximum value of $d(P, Q)$ without violating (2). $|\mathcal{Q}|$ and $|\mathcal{A}|$ are proportional to query stretch. In practice we can achieve higher query stretch than (3) while keeping $|\mathcal{Q}|$ and $|\mathcal{A}|$ within reasonable limits. In the rest of this section we describe a method for obtaining $\mathcal{A}(P)$ and $\mathcal{Q}(Q)$ for $\mathcal{G}_{24}$, with a query stretch of 11, which will allow maximum $|P|$ to be 14-bits and minimum $|Q|$ to be 3-bits.

TABLE I
Distance Distribution of Orbits From Octads and Dodecads

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

As explained in Section III-B, any pattern P 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 I. 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.

Number of 1-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 following property of Bloom filter. As discussed in Section III-C, 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 $\mathcal{Q}$ and $\mathcal{A}$ calculation.

Pseudocodes for finding $\mathcal{Q}$ and $\mathcal{A}$ are presented in Algorithm 1 and Algorithm 2, respectively, in light of the preceding discussion. The algorithm for finding $\mathcal{Q}(Q)$ 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 codewords that are reachable in one hop from the current members of $\mathcal{Q}$ and are within distance 7 (if $|Q|$ is odd) or 8 (if $|Q|$ is even) from Q.

$E(|\mathcal{A}|)$[5] is inversely proportional to $E(|\mathcal{Q}|)$, which in turn is proportional to τ. Hence, τ can be tuned to achieve a desirable ratio of search and advertisement traffic. For our experiments, $\tau = 5$ is used. 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 $\mathcal{A}(P)$ has two stages. The first stage (lines 4–9) is to find set $\mathcal{P}$ of $\mathcal{Q}(Q)$s that will be searched by all possible Q matching the advertised pattern P. Now the problem

[5] $E(X)$ is the expected or average value of variable X.

Algorithm 1 find $\mathcal{Q}(Q)$

1: Input: $Q \in \mathbb{F}_2^{24}$
2: External: τ controls size of $\mathcal{Q}$ and $\mathscr{A}$
3: Returns: $\mathcal{Q} \subset \mathcal{G}_{24}$
4: $\mathcal{Q} \leftarrow \{Y | (Y \in S_8 \wedge d(Y,Q) \leq 5) \bigvee (Y \in U_{12} \wedge d(Y,Q) \leq 6)\}$
5: **if** $|\mathcal{Q}| < \tau$ **then**
6: $\imath \leftarrow (|Q| \text{ is odd})?\ 7 : 8$
7: $\mathcal{Q}' \leftarrow \emptyset$
8: **for** each $Y \in \mathcal{Q}$ **do**
9: $\mathcal{Q}' \leftarrow \mathcal{Q}' \cup \{Z | Z \in neighbors(Y) \wedge d(Z,Q) \leq \imath \wedge Z \notin \mathcal{Q}\}$
10: **end for**
11: $\mathcal{Q} \leftarrow \mathcal{Q} \cup \mathcal{Q}'$
12: **end if**
13: **return** $\mathcal{Q}$

Algorithm 2 find $\mathscr{A}(P)$

1: Input: $P \in \mathbb{F}_2^{24}$
2: Returns: $\mathscr{A}(P) \subset \mathcal{G}_{24}$
3: $\mathscr{A} \leftarrow \{Y | (Y \in S_8 \wedge d(Y,P) \leq 5) \bigvee (Y \in U_{12} \wedge d(Y,P) \leq 6)\}$
4: $\mathscr{P} \leftarrow \emptyset$
5: **for** each Q s.t. $Q \wedge P = Q \wedge |Q| \geq 3 \wedge d(P,Q) > 3$ **do**
6: **if** $\mathscr{A} \cap \mathcal{Q}(Q) = \emptyset$ **then**
7: $\mathscr{P} \leftarrow \mathscr{P} \cup \{\mathcal{Q}(Q)\}$
8: **end if**
9: **end for**
{*use greedy heuristic to find minimum hitting set*}
10: **while** $\mathscr{P}$ not empty **do**
11: find Y s.t. Y is in maximum no. of sets $\mathscr{S} \in \mathscr{P}$
12: $\mathscr{A} \leftarrow \mathscr{A} \cup Y$
13: $\mathscr{P} \leftarrow \mathscr{P} - \{\mathscr{S} | \mathscr{S} \in \mathscr{P} \wedge Y \in \mathscr{S}\}$
14: **end while**
15: **return** $\mathscr{A}$

is to find a small (preferably minimum) set of codewords $\mathcal{A}$ such that $\mathcal{A}$ contains at least one element from each set in $\mathcal{P}$. This is essentially the *minimum hitting set* problem, which in turn, is equivalent to the *minimum set cover* problem. To find $\mathcal{A}$, we have applied the greedy algorithm (line 10–14) based on [13].

Time complexity of computing $\mathcal{A}(P)$ is dominated by the loop in lines 5–8, where we compute $\mathcal{P}$. The upper bound for size of $\mathcal{P}$ is $|\mathcal{P}| \leq \sum_{i=3}^{|P|} \binom{|P|}{i} 2^k$. For $\mathcal{G}_{24}$ and smaller codes we can compute $\mathcal{A}(P)$ in reasonable time. However for larger codes the time complexity will be an obstacle for the implementation. This is one of the main reasons behind selecting $\mathcal{G}_{24}$ for the implementation. With the ongoing research on list decoding techniques we can hope to see efficient algorithms for dealing with larger codes.

C. Routing

In this section, we first explain the mechanism for routing a message originating at peer X to a single target peer Y. Then we provide an algorithm for multicasting a message to multiple destinations. By "peer X" we mean a peer responsible for codeword X. In this section, a peer is assumed to be associated with a single codeword. Section V-B presents a method for assigning multiple codewords to a peer.

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:

$$X_i = \begin{cases} X \oplus g_i & 1 \leq i \leq k \\ X \oplus g_1 \oplus g_2 \oplus \ldots \oplus g_k & i = k+1 \end{cases} \quad (4)$$

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 (4). In such an overlay, it is possible to route a query from any source to any destination codeword in less than or equal to $k/2$ 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 $(1, 2, \ldots, k)$,

$$X \in \mathcal{C} \Longrightarrow Y = (X \oplus g_{i_1} \oplus g_{i_2} \oplus \ldots \oplus g_{i_t}) \in \mathcal{C} \quad (5)$$

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) (see (5)). 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 (4)). 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 hop closer to X_1 than X. X_1 can route the message to any of $X_1 \oplus g_{i_2}$, $X_1 \oplus g_{i_3}, \ldots, X_1 \oplus g_{i_t}$ peers in one hop. In this way, the query can be routed from X to Y in exactly t-hops.

If $t \leq (k/2)$, then our claim is justified. Now let $t > (k/2)$. 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 (4) and (5), 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 $t > (k/2)$ we will need at most $(k-t-1+1) \leq (k/2)$ hops. ■

Given the above described routing protocol, peer X will need a way to find the rows of $G_{\mathcal{C}}$, satisfying (5), 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 (5). 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 (5), 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 the i_jth row of $G_{\mathcal{C}}$ (i.e., g_{i_j}) has a 1-bit in i_jth 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. ■

In Plexus, replication is employed to mitigate the performance problem arising from the failure of superpeers. 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 as follows. It can be proved that if $G_{\mathcal{C}}$ is in standard form then $Y = X \oplus g_{i_1} \oplus \ldots g_{i_t} \Longrightarrow \bar{Y} = X \oplus g_{i_{t+1}} \oplus \ldots g_{i_k}$. Thus, paths from X to Y and X to $\bar{Y}$ are disjoint. This increases fault-resilience and influences uniform distribution of query traffic, especially in cases where peer Y is holding a popular index.

Algorithm 3 $X.route(msg, \mathcal{Y})$

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 (4)*}
{*update* $\mathcal{Y}$}
for each $Y_i \in \mathcal{Y}$ **do**
  if $X = Y_i \vee X = \bar{Y}_i$ **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

As depicted in Fig. 6 (step 5), the routing algorithm will always be subjected to a set of target peers instead of just one peer. A significant portion of routing hops can be reduced by utilizing the shared common paths to different targets as we will show later in Section VI-E. Algorithm 3 presents a pseudocode for multicasting a message from source peer X to a set of destination peers $\mathcal{Y} = \{Y_1, Y_2, \ldots, Y_u\}$.

The pseudocode presented in Algorithm 3 is a simplified version of the routing algorithm used in our 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 $(k/2)+2$. 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 $k/2$. Hence, in the presence of failures, a maximum of $(k/2) + 2$ hops will be required to reach any peer or its replica.

V. Architecture of Plexus

A. Topology

In Plexus architecture (see Fig. 5) a peer is classified as either superpeer or leaf-peer, similar to the two-tier modern Gnutella architecture. Superpeers have longer uptime and higher capacity (in terms of bandwidth and storage) than leaf-peers. Many leaf-peers connect to a superpeer for advertising content index. Based on the composition of current two-tier Gnutella network, we assume that less than 25% of the participating peers will act as superpeers. We apply Plexus routing mechanism within superpeer tier to efficiently locate a content based on partial information about that content. Each superpeer is assigned one or more codeword(s).

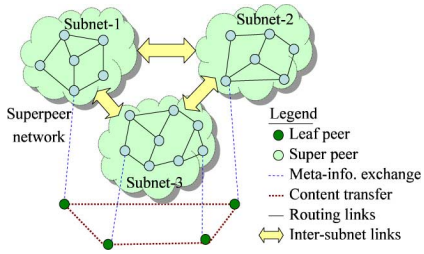


Fig. 5. Architectural overview of Plexus.

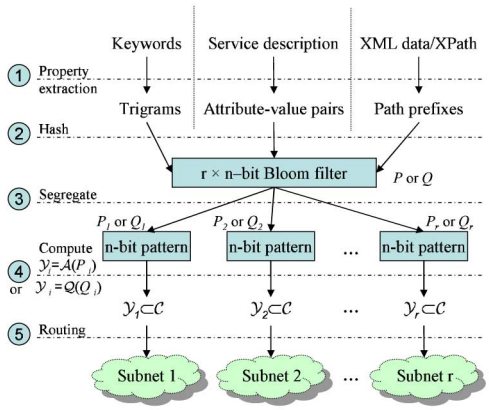


Fig. 6. Search/advertisement process in Plexus.

Superpeers are organized into groups called subnets. Superpeers within a subnet share same codeword space and participate in message routing as explained in Section IV-C. Due to the complexity of computing $\mathcal{A}(P)$ we have to use small codes (e.g., $\mathcal{G}_{24} \equiv (24, 12, 8)$). This restricts the number of superpeers to a few thousands. But we need about 100∼200-bits in a bloom filter to encode about 20∼30 properties (e.g., trigrams or attribute-value pairs) associated with a shared object. According to [37], the modern two-tier Gnutella network contains around a million peers of which 18% are superpeers. Based on these limitations and requirements, the suggested number of subnets is 6∼10. The exact number of subnets in a system is statically defined at the time of deployment and can be computed using the maximum expected size of the superpeer network and the size of the Error-Correcting-Code in use.

Fig. 6 presents the search/advertisement process in Plexus. *Step 1* and *step 2* are specific to the application under consideration, e.g., keyword search, service discovery etc. The input to Plexus is a $(r \times n)$-bit pattern (in this example, a Bloom filter), representing an advertisement (or a query). Here, r is the number of subnets in the system and $n = 24$ as we have used $\mathcal{G}_{24}$. In *step 3* the input pattern is segregated into r chunks (P_i for advertisement or Q_i for query), each n bits long. In *step 4* an n-bit chunk is mapped to a set of codewords $\mathcal{Y}_i$ (i.e., $\mathcal{A}(P_i)$ or $\mathcal{Q}(Q_i)$) and forwarded to any superpeer in subnet i. Finally in *step 5*, the superpeer routes the message in $O((1/2)|\mathcal{Y}_i| \log |C|)$ hops to the target superpeers in $\mathcal{Y}_i$ within subnet i. Thus, each subnet uses different parts (bits) from an advertised or queried pattern and routes independently. The process of mapping patterns to codewords (*Step 4*) is presented in Section IV-B and

the routing mechanism within a subnet (*Step 5*) is presented in Section IV-C.

Instead of propagating an advertisement (or a query) to all the r-subnets, we adopted the *Voting algorithm* [21]. In particular, an advertisement is propagated to $\lfloor (r+1)/2 \rfloor$ subnets, and a query message is propagated to $\lfloor (r/2)+1 \rfloor$ subnets. This ensures that there exists at least one subnet receiving an advertisement, and any query matching that advertisement. The result of a query is computed as the union of the results obtained from each of the $\lfloor (r/2)+1 \rfloor$ subnets. $Subnet_i$ is selected for advertisement P (or query Q) if the weight of the i^{th} chunk i.e., $|P_i|$ (or $|Q_i|$) is within the query stretch as explained in Section IV-B.

Step 1 and *step 2* in Fig. 6 are always executed by a leaf-peer. *Step 3* and *step 4* can be executed either by a leaf-peer or by a superpeer. We prefer the leaf-peers to calculate the $\mathcal{A}_i$ and $\mathcal{Q}_i$, 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 is the responsibility of a superpeer to maintain extra links (marked as inter-subnet links in Fig. 6) to randomly selected superpeers outside its own subnet, and forward a message to appropriate subnets. The number of inter-subnet links has no impact on Plexus routing and is implementation dependent. For our implementation we have assumed that a superpeer in subnet i would maintain a link to a randomly selected superpeer in subnet $((i+1) \bmod r)$. In the rest of this paper we will use the terms *peer* and superpeer interchangeably.

B. Mapping Codewords to Superpeers

So far we assumed that a superpeer is responsible for a unique codeword, which is not a practical assumption. In this section, we present a way of partitioning the codeword space, and dynamically 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 (5)). The codewords can be partitioned using a logical binary partitioning tree with height at most k. At ith level of the tree, partitioning takes place based on the presence (or absence) of g_i (the ith row of $G_{\mathcal{C}}$) in a codeword. Fig. 7 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. Fig. 7 illustrates the routing table entries for superpeer $X = g_1 \oplus g_3 \oplus g_6 \oplus g_9$ with an equivalent prefix of $g_1\bar{g}_2g_3\bar{g}_4$. Here $\bar{g}_i$ indicates the absence of the i^{th} row i.e., g_i.

In order to incorporate the concept of partitioning codeword space in Algorithm 3, 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.

C. The Join Process

In this section we present the protocol that a superpeer has to follow to join a Plexus network. The first superpeer in the system begins with a random codeword, and all entries in its routing table point to itself. A new superpeer joins the system by taking over a part of the codeword space from an existing peer, say 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 ith row in the formation of X, respectively. 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 $G_{\mathcal{C}}$.

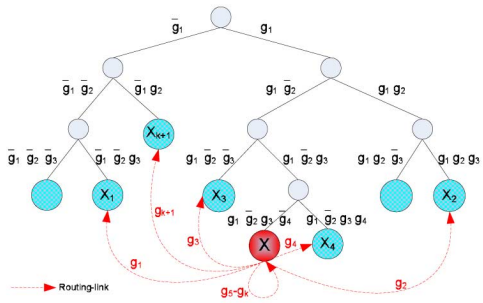


Fig. 7. Logical binary partitioning tree for assigning codewords to superpeers. The routing table entries for peer X are also presented.

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 2 hops (see Theorem. 3) 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.

Theorem 3: If $X (= \rho_1 \cdot \rho_2 \ldots \rho_t \cdot \rho_{t+1} \cdot \rho_k)$ with t prefix bits is split to $Y (= \rho_1 \cdot \rho_2 \ldots \rho_t \cdot \rho_{t+1} \cdot)$ and $Z (= \rho_1 \cdot \rho_2 \ldots \rho_t \cdot \bar{\rho}_{t+1} \cdot)$ then all neighbors of Y and Z will be within 2 hops of X.

Proof: Let, $\Psi_t(X)$ denotes the first t bits of X. Then $\Psi_{t+1}(Y)$ and $\Psi_{t+1}(Z)$ differ in exactly one bit. Again $\Psi_{t+1}(Y_i)$ and $\Psi_{t+1}(Z_i)$ differ from $\Psi_{t+1}(X_i)$ by at most 1 bit. Thus, $\Psi_{t+1}(Y_i)$ and $\Psi_{t+1}(Z_i)$ will differ from $\Psi_{t+1}(X)$ by at most two bits, i.e., at most 2 hops away. ■

To reduce the possibility of unbalanced partitioning of the codeword space, a joining peer should crawl the neighborhood of the seed peer, until a local 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. Since the maximum height of the binary tree (see Fig. 7) is k and the crawling process is greedy (i.e., height of current minima is reduced by one at each hop) it would take at most k hops to find a local minima. Since we are using the crawl mechanism for each join operation, the maximum height difference between any two superpeers is much smaller than k. In our experiments with Golay code $(k = 12)$ we found that it takes around 3.4 hops on average to find a local minima.

A peer usually joins the subnet of the seed peer. Alternatively, the joining peer can use the inter-subnet link(s) of the seed peer if the subnet of the seed is saturated, which is a very unlikely case for a well-designed system.

D. 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 in Plexus 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; replace itself with Z_t from the routing tables of all of its neighbors; and finally rejoin the system as Y. Z_t has to reduce its prefix string by one, in order to accommodate the changes. In case Z_t has also failed then Z should start the recovery for Z_t first.

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

E. 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 $|\mathcal{A}|$ and $|\mathcal{Q}|$ be the average size of $\mathcal{A}(P)$ and $\mathcal{Q}(Q)$, respectively. $\gamma_{\mathcal{Q}}$ and $\gamma_{\mathcal{A}}$ stand, respectively, for the fraction of routing hops reduced due to the presence of multicasting during search and advertisement. If N is the total number of superpeers in the system, then the number of superpeers in a subnet is $2^k \approx N/r$. Now, according to Theorem 1 it will require at most $(k/2) \approx (1/2)\log_2(N/r)$ hops to route a message within a subnet. Consequently, considering the use of the Voting algorithm as explained earlier, the expected number of hops required for routing an advertisement is computed to be

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

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

$$H_{\mathcal{Q}} = \frac{1}{2}\left\lfloor \frac{r}{2}+1 \right\rfloor (1-\gamma_{\mathcal{Q}})|\mathcal{Q}|\log_2 \frac{N}{r}. \tag{7}$$

For estimating $|\mathcal{Q}|$ we can adopt the extension of Johnson bound proposed by V. Guruswami and M. Sudan (Theorem 1 in [23]) as follows. Assume a $\langle n, k, d\rangle f$ binary code (C). Let δ and γ $(0 < \delta, \gamma < 1)$ be constants such that $d = (1/2)(1-\delta)n$, $t = (1/2)(1-\gamma)n$ and $\gamma > \sqrt{\delta}$, then it can be derived from Theorem 1 in [23] that

$$|\mathcal{Q}| = |B_t(Q) \cap \mathcal{C}| \leq min\left\{n, \frac{1-\delta}{\gamma^2 - \delta}\right\}. \tag{8}$$

Estimating $|\mathcal{A}|$ as computed in Algorithm 2 is not a straight forward process. Rather we can compute an upper bound. Let $\mathcal{S}$ be the set of all subsets (Q) of an arbitrary pattern P such that, $|Q| \geq u$, where u is the minimum allowed weight of a query pattern. We compute $|\mathcal{A}|$ to be a subset of the codewords required to cover $\mathcal{S}$, with a covering radius f greater than the error correcting radius, say $e = \lfloor (d/2) - 1 \rfloor$, of the code. Hence, the number of codewords required to cover $\mathcal{S}$ with error correcting radius e will be an upper bound for $|\mathcal{A}|$.

Let $\mu = |P \wedge C|$ and $\eta = |P \wedge \overline{C}|$ for some codeword $C \in \mathcal{C}$. Evidently, $|P| = \eta + \mu$. We define query stretch s to be the difference between $|P|$ and the minimum weight of a query that we want to be discovered, i.e., $s = \mu + \eta - u$. Now we can construct a query Q from P by taking i bits from the μ 1-bits of P (where C has 1-bits) and j bits from the η 1-bits of P (where C has 0-bits) in $\binom{\mu}{i}\binom{\eta}{j}$ ways. Trivially, $d(P, Q) = i + j$ and $d(C, Q) = w + i - j$, where $w = d(P, C)$. In order for Q to be in $\mathcal{S}$, $d(P, Q) \leq s$ and $d(C, Q) \leq e$. Thus, the total number of elements in $\mathcal{S}$ covered by C can be computed as

$$\sum_{i=0}^{\frac{1}{2}(e+s-w)} \sum_{j=0}^{\frac{1}{2}(s-e+w)} \binom{\mu}{i}\binom{\eta}{j}.$$

We can compute $|\mathcal{S}| = \sum_{l=1}^{\mu+\eta-u} \binom{\mu+\eta}{l}$. Hence, the upper bound on $|\mathcal{A}|$ can be computed as

$$|\mathcal{A}| \leq \frac{\sum_{l=1}^{\mu+\eta-u} \binom{\mu+\eta}{l}}{\sum_{i=0}^{\frac{1}{2}(e+s-w)} \sum_{j=0}^{\frac{1}{2}(s-e+w)} \binom{\mu}{i}\binom{\eta}{j}}. \tag{9}$$

For the experiments presented in this work we have used the extended Golay code and have set the minimum query weight $u = 3$ and the maximum advertisement weight $|P| = 14$. With this wide query stretch we obtained the average value of $|\mathcal{A}|$ and $|\mathcal{Q}|$ as 21.08 and 17.53 respectively for the dataset used in our experiments. Our experiments with Golay code reveal that the value of $\gamma_{\mathcal{A}}$ and $\gamma_{\mathcal{Q}}$ is proportional to $|\mathcal{A}|$ and $|\mathcal{Q}|$, respectively. For the average sizes of $|\mathcal{A}|$ and $|\mathcal{Q}|$, $\gamma_{\mathcal{A}}$ and $\gamma_{\mathcal{Q}}$ are reported to be 0.78 and 0.85, respectively (see Fig. 12).

VI. Experimental Evaluation

In this section we evaluate the effectiveness of Plexus protocol on three aspects of a music-sharing P2P system: routing efficiency, search completeness and fault-resilience.

A. Simulation Setup

We have simulated a growing network, where the overlay is gradually grown from an initial set of few superpeers. The simulator goes through a growing phase and a steady-state phase to achieve different network sizes. During the growing phase, arrival rate is higher than departure rate (up to five times). Once a target population size is reached the simulator turns to the

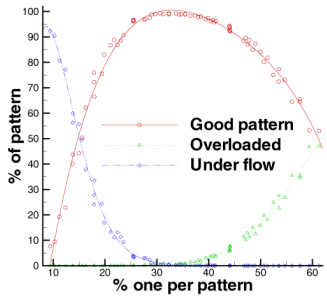


Fig. 8. Fitness of patterns for advertisement.

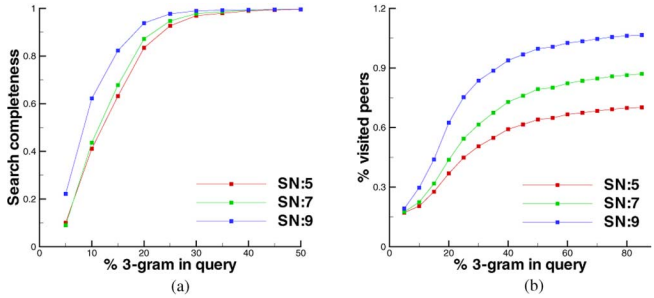


Fig. 9. Effect of information content of a query on search completeness. (a) Search completeness. (b) Impact on routing.

steady state phase. While in steady state, arrival rate is approximately equal to departure rate and the network size does not vary much over time. During this period, advertisements and queries are performed and performance metrics like routing efficiency, search completeness, etc., are measured.

As explained in Fig. 6, problems that can be mapped to distributed subset matching can be solved with Plexus. For this experiment we have applied Plexus to partial keyword search in music-sharing P2P system. The music information used in this simulation is based on the online database of more than 200 000 songs information available at http://www.leoslyrics.com/. For a $\langle song-title, artist\rangle$ pair, we constructed a Bloom-filter of length $m(=r*24)$-bits. Here, r is the number of subnets in the network. A Bloom filter represents the set of trigrams extracted from a $\langle song-title, artist\rangle$ pair. We have experimented with three values of r: 5, 7, and 9; since m is dependent on r, the percentage of 1-bits in a pattern will vary if a fixed value for $\hbar$ (number of hash functions) is used. To find the proper value of $\hbar$ we constructed Bloom filters for different values of r and $\hbar$. For each Bloom filter we constructed r 24-bit chunks and tested for fitness. A chunk is considered to be fit for advertisement if it contains 6$\sim$14 1-bits (see Section IV-B), and a $r\times 24$-bit pattern (i.e., Bloom filter) is considered to be fit for advertisement if it contains at least $\lfloor (r+1)/2 \rfloor$ fit chunks. Fig. 8 plots the percentage of fit (or good) patterns as a function of the percentage of 1-bits. It also depicts the percentage of overflow patterns (majority of the chunks having more than 14-bits) and underflow patterns (majority of the chunks having less than 6-bits). Based on the peak values in the curve we have used $\hbar = 3$, 4 and 5 for $r = 5$, 7 and 9, respectively. It should be noted that the possibility of false positives due to the use of Bloom filters will not degrade correctness of search result. It will only increase routing traffic to some extent as some superpeers, not containing any match, may be visited by the search. This effect is incorporated in the measured search traffic of the system.

B. Impact of Query Content on Search Completeness

Search completeness is measured as the percentage of advertised patterns (matching the query pattern) that were discovered by the search. A query is formed as a Bloom filter consisting of $\beta\%$ of trigrams (randomly chosen) from a $\langle song-title, artist\rangle$ pair. Fig. 9(a) presents search completeness as a function of β, which is varied from 5%–50% in 5% steps on networks of about 20 000 superpeers. For each step we performed 5000 queries. The number of 24-bit chunks having at least three 1-bits decreases as lower percentages of trigrams are taken from the original advertisement. Hence, read quorum for the *Voting algorithm* could not be met for lower values of β. This can also be observed from Fig. 9(b), which is a plot of the percentage of visited peers against β. The sharp rise in the percentage of visited peers in Fig. 9(b) justifies the increase in search completeness around $\beta = 30\%$ in Fig. 9(a). It should also be noted that the percentage of visited peers is higher for higher values of r, because, the number of subnets to be searched is proportional to r. A completeness level of around 97% is achieved for $\beta = 33\%$. Only 2% increase in search completeness is achievable for $\beta > 33\%$, though at the expense of a higher percentage of visited peers. Therefore, we have used $\beta = 33\%$ in the subsequent experiments. However, there is no requirement that a query has to be formed using 33% trigrams from an advertisement. If lower than 33% trigrams are used then this will yield lower level of search completeness at the cost of lesser search traffic.

C. Scalability and Routing Efficiency

The impact of network size on routing efficiency and distribution of indexing load are considered in this section. Fig. 10(a) and (b) plot the average percentage of visited superpeers per search and advertisement, respectively, against the logarithm of the total number of superpeers in the network. The linear decrease in the curves confirms our assertion in Theorem 1, i.e., number of visited peers per search and advertisement holds logarithmic relation with the total number of peers in the system. It should also be noted that the percentage of visited peers increases with the increase in the number of subnets (i.e., r). For networks with fixed size (say N) and varying r (say r_1 and r_2), ratio of the percentage of visited peers can be calculated as $(H_1/H_2) \approx (r_1 \log(N/r_1))/(r_2 \log(N/r_2)) \approx (r_1/r_2)$ (for small r_1 and r_2, see (7)), i.e., $H \propto r$. Similar results are obtained for advertisement traffic, as presented in Fig. 10(b).

Fig. 10(c) presents the percentage of visited peers for join operation as a function of network size. As discussed in Section V-C and reflected in Fig. 10(c), the join process updates links only within a subnet and is thus independent of the number of subnets in the system. A join operation may require at most $O(\log N)$ links (for this experiment 12 as $\mathcal{G}_{24}$ has been used) to be established. This justifies the decrease of the curves against the logarithm of network size.

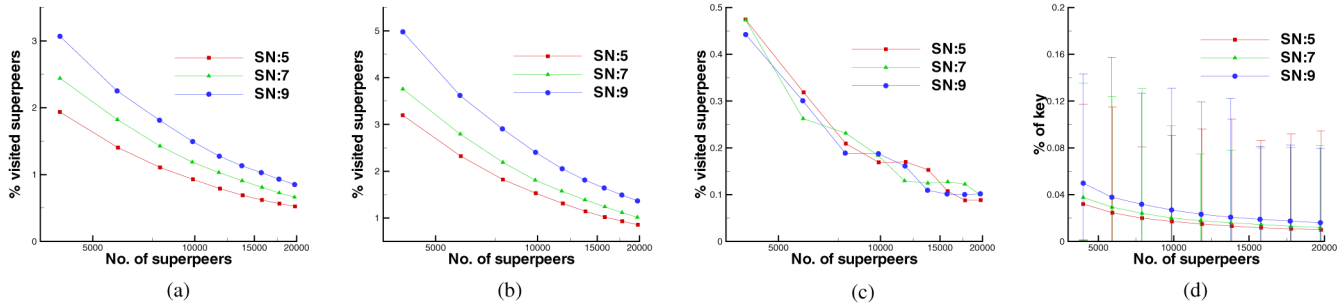


Fig. 10. Routing efficiency and scalability with network size. (a) Routing efficiency (search); (b) routing efficiency (advertisement); (c) join overhead; (d) load distribution. average, 1st and 99th percentiles.

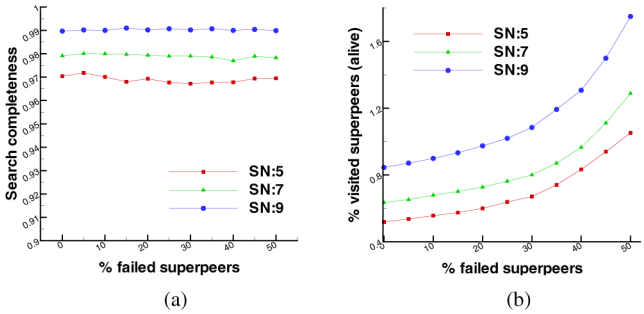


Fig. 11. Fault resilience. (a) Search completeness. (b) Query routing traffic.

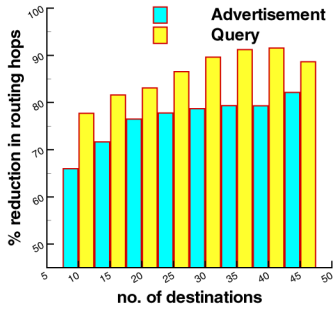


Fig. 12. Effectiveness of simultaneous routing to multiple targets: reduction in routing hops as a function of the number of targets.

Fig. 10(d) presents the distribution of advertised patterns over the network. Average number of keys per pattern as well as 1st and 99th percentiles are presented. The skew in indexing load is higher for smaller networks and reduces gradually as the network size increases. Note that the average values of *% key per peer* in Fig. 10(d) are very close to the expected values $= 100/N$ (N is the network size) and the 99th-percentiles are within reasonable limit.

D. Fault Tolerance

In this section, we analyze the robustness of Plexus in presence of simultaneous failures of a large number of superpeers. We start with a steady-state network of about 20,000 superpeers and cause each peer to fail with probability p. After the failures have occurred we perform 5000 queries and measure search completeness [Fig. 11(a)] and the percentage of visited superpeers per query [Fig. 11(b)]. 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.

The number of replicas of an advertised pattern is proportional to r, thus much better search completeness is achieved for higher values of r. 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 IV-C.

It can be observed from Fig. 11(a) that search completeness is almost identical regardless of the percentage of failed peers (up to 50%). It should be noted that the query patterns where formed using 33% of the trigrams from existing patterns only; lost patterns due to the failure of a superpeer where not considered. The high levels of search completeness indicate that the superpeers remain reachable even in the presence of a large number of failures. This is possible because of the existence of multiple paths connecting any two superpeers within a subnet. However, this resilience to failure comes at an expense of increased routing overhead as observed in Fig. 11(b).

E. Effectiveness of Multicast-Routing

Network distance of the codewords in $\mathcal{A}(P)$ and $\mathcal{Q}(Q)$ have significant impact on routing efficiency. The routing algorithm described in Section IV-C 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 pairwise routing. Fig. 12 shows the reduction in routing hops (rrh) calculated as

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

The reduction in routing hops takes place within a subnet and hence does not depend on the number of subnets present in the system. The bar chart in Fig. 12 displays the average rrh for groups of 5 targets, i.e., 6–10, 11–15, etc. The denominator for rrh equation has been calculated as: $\sum_{Y \in \mathcal{Y}_i} d(\phi(X), \phi(Y))$, where X is the source peer, $\phi(X)$ returns the k-bits of a codeword corresponding to the I_k part of $G_{\mathcal{C}}$, and $\mathcal{Y}_i$ is the set of

TABLE II
SIMULATION PARAMETERS FOR COMPARATIVE EVALUATION

	Param	Value	Description
Flooding	α	15	Topology parameter controlling network density
	Δ	12	Average degree
	R_F	120	Average replica per advertisement
	TTL_F	4	Time to live
R-walk	α	15	Topology parameter
	Δ	12	Average degree
	R_R	120	Average replica per advertisement
	TTL_R	10	Time to live
	W_R	15	Number of walkers
DHT	R_C	4	Replica per key
	X	Chord	DHT routing protocol
DPMS	H	6	Height of indexing hierarchy
	R_D	2	Recursive replication constant
	B	4-6	Branching factor
	A	0.6	Aggregation ratio
	W	180	Pattern width
	$\hbar_D$	3	No. of hash functions in Bloom filter
Plexus	r	7	Number of subnets
	$\hbar_P$	3	No. of hash functions in Bloom filter
	u	3	Minimum query weight per chunk
	v	14	Maximum advertisement weight per chunk

target peers calculated as $\mathcal{A}(P_i)$ or $\mathcal{Q}(Q_i)$ (see Fig. 6). By observing the high values of rrh, it can be inferred that the codewords close in Hamming distance are assigned to the superpeers in close vicinity within the overlay.

VII. COMPARATIVE EVALUATION

In Section VI we presented experimental results for assessing performance of Plexus. In this section we will compare the performance of Plexus w.r.t. other P2P search techniques. These techniques can be broadly classified into three categories:

1) **Unstructured** techniques do not build any index and use uninformed search, like flooding and random walk.
2) **Semi-structured** techniques build index information but do not place any restriction on index placement. Indexed information contains hints on possible location of the content. Freenet and DPMS fall into this category.
3) **Structured** techniques rely on some index placement rule that allows one to pinpoint the peer(s) responsible for a given index. Each peer knows the exact location of the contents it has indexed. Examples in this category include DHT techniques (e.g., Chord [36], CAN [33], Kademlia [31], etc.), Plexus and Skipnet [25].

The DPM problem can be solved using unstructured and semi-structured techniques. Plexus, on the other hand, is the only known structured technique that can solve the DPM problem. Thus, it is not possible to compare Plexus with other structured techniques if we try to solve the DPM problem. Instead, we have compared the performance of different search techniques w.r.t. an application of the DPM problem; here partial keyword matching.

The search techniques used for comparison are flooding, random-walk, DPMS and Chord. Table II describes the system specific parameters for each of these search techniques and the values used in the experiment.

Unstructured search techniques like flooding and random-walk do not place any restriction on the underlying network topology. For these two cases we have adopted the network model proposed in [3], which mimics the characteristics of the contemporary Internet topology.

We have used Chord as the representative for the DHT routing protocols. To enable partial keyword matching we adopted the strategy presented in [7] and [24]. In brief, each keyword is broken into 3-grams, which are then hashed and routed to the responsible peers. For improving fault resilience the advertisement is replicated at (R_C) peers along Chord ring following the responsible peer. This approach is not suitable for real life implementation because of the volume of advertisement traffic it generates. Yet there exists few proposals for reducing advertisement traffic by adopting specialized hash functions like Locality Preserving Hashing (LPH) as used in Squid [35] and Fingerprint function as used in [10]. These techniques support prefix matching only but we are interested here in partial (or infix) matching.

DPMS [4] uses a hierarchical topology and indexing structure. Lossy aggregation is used for controlling the index volume at higher level peers. Recursive replication along the indexing hierarchy is adopted for increasing fault-resilience and load distribution. Keyword to pattern construction process in DPMS is similar to the one in Plexus.

For this experiment we have used PeerSim [2]; a cycle driven simulator, written in Java. User-defined Java components can be plugged into the simulator for defining and monitoring overlay topology, routing mechanism, join/leave protocol, replication strategy, etc. The dataset used for this experiment is the same as the one used in Section VI. The rest of the section presents relative performance of the five search techniques w.r.t. different performance metrics.

A. Topology Maintenance Overhead

Topology maintenance overhead is dependent on the average degree of the peers. Topology maintenance overhead is usually lesser for a smaller value of average node degree. Order of node degree in Plexus is $O(\lg(N/r) + r)$ and for chord it is $O(\lg N)$. For flooding and random walk we have used the internet topology generation model, in which the average degree is governed by topology parameter δ. Finally, in DPMS node degree is $(B + R_D + \vartheta)$, where ϑ is the cardinality of neighborhood-list used by the Newscast protocol [40]. As reflected in Fig. 13(a), the average degree depends on network size for Plexus and DHT-technique, whereas it is constant for unstructured or semistructured techniques.

Minimizing peer join overhead is crucial for handling peer dynamism in large scale distributed systems. It is mostly governed by the protocol used to define the logical overlay topology. Join overhead is minimum for unstructured cases as the only rule for join protocol is to keep the degree below a maximum value. In Plexus, a joining peer has to locate $O(\lg(N/r))$ neighbors, all of which are located within two hops of the seed peer (see Theorem 3), while the r external links can be obtained from the seed and the neighbors. In essence, the effect of joining is localized in Plexus and therefore the low overhead. The higher overhead of peer join in DPMS results from the strict rules of connectivity within the indexing hierarchy. Chord, and in general DHT-techniques, exhibit higher join overhead because of the global impact it has. In Chord,

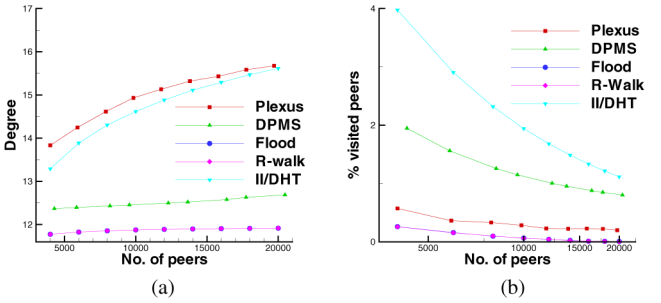


Fig. 13. Topology maintenance overhead. (a) Average degree. (b) Join overhead.

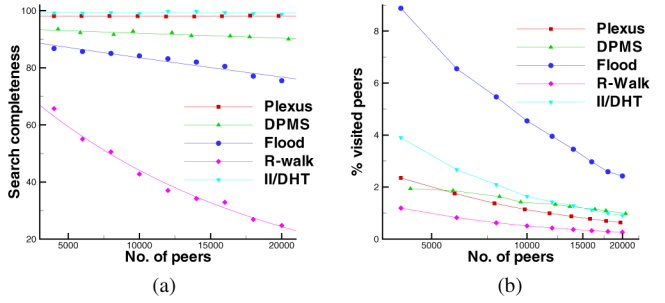


Fig. 15. Search efficiency. (a) Search completeness. (b) Query traffic.

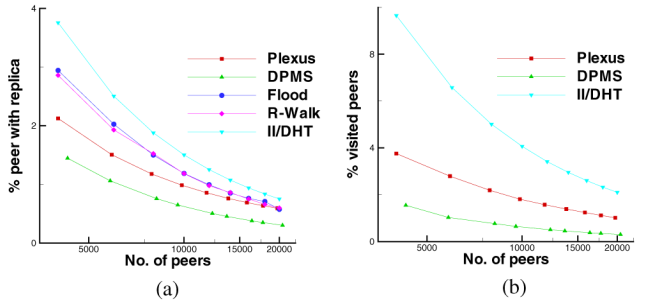


Fig. 14. Advertisement efficiency. (a) Replication factor. (b) Advertisement traffic.

the joining peer becomes a predecessor of the seed peer. The joining peer cannot benefit from the seed peer's finger table because of the unidirectional nature of Chord routing (see [27]). In summary, the average degree in Plexus is close to that in DHT-techniques though the join overhead is much less.

B. Advertisement Efficiency

Replication is used by search techniques for different purposes. Unstructured techniques rely on content replication for improving search completeness, while structured techniques use index replication for improving fault resilience. We define *replication factor* to be the percentage of peers containing a given advertisement (i.e., index or content). Update propagation traffic is proportional to average replication factor. Fig. 14(a) presents the average replication factor, whereas Fig. 14(b) plots the average advertisement traffic. Advertisement traffic for Flooding and R-walk have not been presented as in these two cases there exists no explicit advertisements and content information is propagated as a result of the search process.

In Plexus, the expected number of replica for some advertisement, say P, can be calculated as $|\mathcal{A}(P)|\lfloor(r+1)/2\rfloor$. Here, $|\mathcal{A}(P)|$ depends on the Error Correcting Code (ECC) in use and query stretch. In DPMS, the replication factor depends on H and R_D (see Table II), and not on network size. It can be calculated as $\sum_{i=1}^{H-1} R_D^i$. For Flooding and Random-walk we have used uniform replication with an average of 120. In Chord, replication overhead depends on R_C and the average number of 3-grams per advertisement which is 29.37 for the experimental dataset. Advertisement traffic is low in DPMS because it uses bulk advertisement. DHT/Chord requires many independent DHT lookups to register the 3-grams within the Chord ring. Plexus, on the contrary, exploits multicast routing and the presence of closeness (in Hamming distance) within the target codewords to reduce advertisement traffic.

C. Search Efficiency

In this experiment we simulated networks of different sizes and measured two aspects of search efficiency : search completeness[6] [Fig. 15(a)] and generated traffic [Fig. 15(b)]. A query has been constructed using a random 35% 3-gram from a randomly chosen advertisement. Flooding and Random-walk yields the two extremes in terms of search traffic. In spite of visiting a large percentage of peers, search completeness in Flooding is low. On the other extreme, Random-walk generates the least traffic and lowest level of search completeness.

Unlike structured techniques DPMS does not assign indexes to peers. Advertisements matching a given query cannot be found in any predefined (or computable) set of peers, rather a user has to search additional peers for discovering each match. In the experiment with DPMS we have stopped searching after 20 matches were found. Better search completeness is achievable by increasing this termination criteria, though at the expense of additional search traffic.

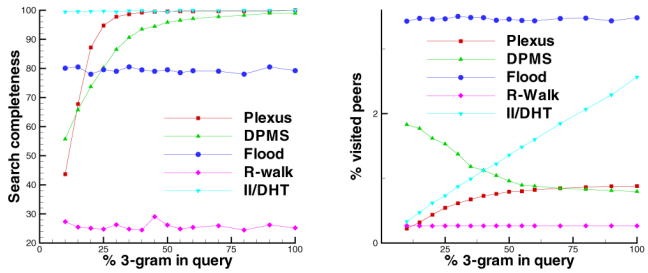
Search traffic is almost half in Plexus than that in DHT/Chord; yet search completeness in these two systems is almost identical. On the other hand, Plexus and DPMS generate similar search traffic, yet Plexus provides higher level of search completeness.

D. Search Flexibility

One of the major target of Plexus is to discover advertisements using only partial information. Search flexibility refers to the ability of a search system to discover matching advertisements using partial information. For this experiment we consider overlays of about 20,000 peers and vary the percentage of 3-grams (PNG) from the advertisements used for constructing the queries. Fig. 16(a) and (b) present the impact of PNG on search completeness and query traffic, respectively.

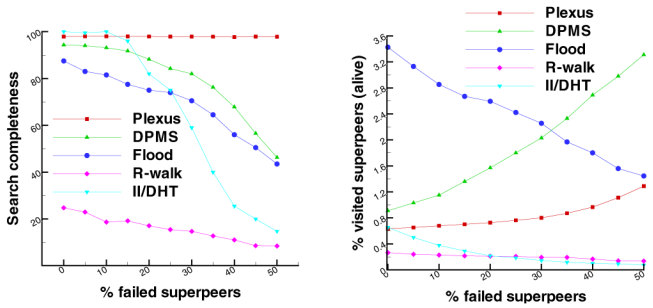
For Flooding and Random-walk, search traffic and completeness are invariant to the change in PNG. In the case of DPMS, we used an iterative search method and stopped after 20 matches were found. However at low PNG, the number of matching advertisements is more than 20, which results in low completeness levels. On the other hand, false-match probability is higher

[6]Search completeness $= E(\text{no. of discovered distinct matches}/\text{available matches (distinct)} \times 100)$.



(a) (b)

Fig. 16. Search flexibility. (a) Search completeness. (b) Query traffic.



(a) (b)

Fig. 17. Fault resilience. (a) Search completeness. (b) Query traffic.

at low PNG levels, which explains the higher search traffic in DPMS for low PNG levels.

In Plexus, many read quorum could not be satisfied at low PNG as there were too few bits per chunk. This results into lower search completeness and search traffic in Plexus for $PNG < 25\%$. On the contrary, DHT/Chord produces almost complete (99.8%) search results for all PNG levels, though search traffic increases linearly with PNG. In contrast, search traffic for Plexus is almost constant for $PNG > 40\%$.

E. Fault Resilience

For this experiment $PNG = 35\%$ is used for constructing queries from the available advertisements. We started with overlays of about 20,000 peers and gradually caused randomly chosen peers to fail in 5% steps.

In all techniques, except for Plexus, search completeness falls with increase in the percentage of failed peers (PFP) [Fig. 17(a)]. The fall is sharpest for DHT/Chord because of its unidirectional routing table and lack of alternate routing paths. On the contrary, the use of multi-path routing and replication allows Plexus to achieve an almost constant level of search completeness regardless of failure rate, though at the expense of higher search traffic.

For Plexus and DPMS, search traffic increases with PFP [Fig. 17(b)], because these two techniques adopt alternate paths to reach the target peer(s) in presence of failures. On the other hand, search traffic decreases with the increase in PFP for DHT/Chord and flooding because in these two cases the effective search tree gets pruned as more peers fail. This results into decreased search completeness. Finally for Random-walk, search traffic is independent of PFP as a walker is not aborted before TTL expires unless all the neighbors of an intermediate peer along the walker have failed.

VIII. Conclusion and Future Work

We identified the distributed subset matching problem in [4] and this paper presents Plexus as an efficient solution to the problem. Plexus has a *partially decentralized* architecture utilizing *structured search*. As demonstrated by the simulation results, for a network of about 20000 superpeers Plexus needs to visit only 0.7%∼1% of the superpeers to resolve a query and can discover about 97%∼99% of the advertised patterns matching the 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. Plexus delivers a high level of fault-resilience by using replication and redundant routing paths. Even with 50% failed superpeers, Plexus can attain a high level of search completeness (about 97%∼99%) by visiting only 1.4%∼2% of the superpeers. Plexus can route queries and advertisements to target peers in $O(logN)$ hops and by using $O(logN)$ links.

The originality of our approach lies in 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.

Although we have focused on subset matching in this paper, Plexus can easily be tailored to support inexact (or edit distance) matching. We intend to investigate this possibility as a future extension to this work. We also intend to extend the experiments with other linear codes, alternate types of Bloom filters, possible alternatives to the Voting algorithm and more efficient mechanisms for $\mathcal{A}(P)$ computation.

References

[1] Gnutella website. [Online]. Available: http://www.gnutella.com
[2] PeerSim Peer-to-Peer Simulator. [Online]. Available: http://www.peersim.sourceforge.net/
[3] A. Fabrikant, E. Koutsoupias, and C. Papadimitriou, "Heuristically optimized trade-offs: A new paradigm for power laws in the internet," in *Proc. ICALP*, 2002, pp. 110–122.
[4] R. Ahmed and R. Boutaba, "Distributed pattern matching: A key to flexible and efficient P2P search," *IEEE J. Sel. Areas Commun.*, vol. 25, no. 1, pp. 73–83, Jan. 2007.
[5] A. Amir, E. Porat, and M. Lewenstein, "Approximate subset matching with don't cares," in *Proc. SODA*, 2001, pp. 305–306.
[6] S. Androutsellis-Theotokis and D. Spinellis, "A survey of peer-to-peer content distribution technologies," *ACM Computing Surveys*, vol. 45, no. 2, pp. 195–205, Dec. 2004.
[7] M. Balazinska, H. Balakrishnan, and D. Karger, "INS/Twine: A scalable peer-to-peer architecture for intentional resource discovery," in *Proc. Pervasive Computing*, 2002, pp. 195–210.
[8] M. Bawa, T. Condie, and P. Ganesan, "LSH Forest: Self-tuning indexes for similarity search," in *Proc. 14th Int. World Wide Web Conf. (WWW2005)*, May 2005, pp. 651–660.
[9] B. H. Bloom, "Space/time trade-offs in hash coding with allowable errors," *Commun. ACM*, vol. 13, no. 7, pp. 422–426, 1970.
[10] A. Bonifati, U. Matrangolo, A. Cuzzocrea, and M. Jain, "XPath lookup queries in P2P networks," in *Proc. WIDM*, 2004, pp. 48–55.
[11] M. Cai and M. Frank, "RDFPeers: A scalable distributed RDF repository based on a structured peer-to-peer network," in *Proc. WWW Conf.*, 2004, pp. 650–657.
[12] Y. Chawathe, S. Ratnasamy, L. Breslau, N. Lanham, and S. Shenker, "Making Gnutella-like P2P systems scalable," in *Proc. ACM SIGCOMM*, 2003, pp. 407–418.

[13] V. Chvtal, "A greedy heuristic for the set covering problem," *Math. Oper. Res.*, vol. 4, pp. 233–235, 1979.
[14] E. Cohen, A. Fiat, and H. Kaplan, "Associative search in peer to peer networks: Harnessing latent semantics," in *Proc. IEEE INFOCOM*, 2003, electronic edition, 11 pp.
[15] R. Cole and R. Harihan, "Tree pattern matching and subset matching in randomized $O(n \log^3 m)$ time," in *Proc. ACM STOC*, 1997, pp. 66–75.
[16] J. Conway and N. Sloane, "Orbit and coset analysis of the Golay and related codes," *IEEE Trans. Inf. Theory*, vol. 36, no. 5, pp. 1038–1050, Sep. 1990.
[17] 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. ACM MOBICOM*, 1999, pp. 24–35.
[18] E. Franconi, G. Kuper, A. Lopatenko, and I. Zaihrayeu, "The coDB robust peer-to-peer database system," in *Proc. WWW Conf.*, New York City, May 2004, pp. 382–393.
[19] L. Galanis, Y. Wang, S. Jeffery, and D. DeWitt, "Locating data sources in large distributed systems," in *Proc. Very Large Data Bases (VLDB) Conf.*, 2003, pp. 874–885.
[20] J. Gao and P. Steenkiste, "Design and evaluation of a distributed scalable content discovery system," *IEEE J. Sel. Areas Commun.*, vol. 22, no. 1, pp. 54–66, Jan. 2004.
[21] D. K. Gifford, "Weighted voting for replicated data," in *Proc. Symp. OS Principles*, 1979, pp. 150–162.
[22] M. J. E. Golay, "Notes on digital coding," *Proc. IRE*, vol. 37, no. 657, 1949.
[23] V. Guruswami and M. Sudan, "Extensions to the Johnson Bound," MIT, 2001 [Online]. Available: http://theory.lcs.mit.edu/~madhu/papers/johnson.ps
[24] M. Harren, J. M. Hellerstein, R. Huebsch, B. T. Loo, S. Shenker, and I. Stoica, "Complex queries in DHT-based peer-to-peer networks," in *Proc. IPTPS*, 2002, pp. 242–259.
[25] N. Harvey, M. B. Jones, S. Saroiu, M. Theimer, and A. Wolman, "Skipnet: A scalable overlay network with practical locality properties," in *Proc. USITS*, Mar. 2003, electronic edition, 14 pp.
[26] Y. Joung, L. Yang, and C. Fang, "Keyword search in DHT-based peer-to-peer networks," *IEEE J. Sel. Areas Commun.*, vol. 25, no. 1, pp. 46–61, Jan. 2007.
[27] X. Kaiping, H. Peilin, and L. Jinsheng, "FS-chord: A new P2P model with fractional steps joining," in *Proc. AICT/ICIW*, 2006, p. 98.
[28] J. Kim and G. Fox, "A hybrid keyword search across peer-to-peer federated databases," in *Proc. ADBIS 2004 — Local Proc.*, Budapest, Hungary, Sep. 2004, electronic edition, 13 pp.
[29] M. Li, W. Lee, and A. Sivasubramaniam, "Neighborhood signatures for searching P2P networks," in *Proc. IDEAS*, 2003, pp. 149–159.
[30] C. Lv, P. Cao, E. Cohen, K. Li, and S. Shenker, "Search and replication in unstructured peer-to-peer networks," in *Proc. ICS*, 2002, pp. 84–95.
[31] P. Maymounkov and D. Mazières, "Kademlia: A peer-to-peer information system based on the XOR metric," in *Proc. IPTPS*, 2002, pp. 53–65.
[32] W. S. Ng, B. C. Ooi, K. L. Tan, and A. Zhou, "PeerDB: A P2P-based system for distributed data sharing," in *Proc. ICDE*, 2003, pp. 633–644.
[33] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker, "A scalable content-addressable network," in *Proc. ACM SIGCOMM*, 2001, pp. 161–172.
[34] M. Schlosser, M. Sintek, S. Decker, and W. Nejdl, "A scalable and ontology-based P2P infrastructure for semantic web services," in *Proc. P2P Computing*, Sep. 2002, pp. 104–111.
[35] C. Schmidt and M. Parashar, "Enabling flexible queries with guarantees in P2P systems," *IEEE Internet Computing*, vol. 8, no. 3, pp. 19–26, May-Jun. 2004.
[36] 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. Networking*, vol. 11, no. 1, pp. 17–32, Feb. 2003.
[37] D. Stutzbach and R. Rejaie, "Characterizing unstructured overlay topologies in modern P2P file-sharing systems," in *Proc. Internet Measurement Conf. (IMC 2005)*, Oct. 2005, pp. 49–62.
[38] C. Tang, Z. Xu, and M. Mahalingam, "pSearch: Information retrieval in structured overlays," presented at the First Workshop on Hot Topics in Networks (HotNets-I), Princeton, NJ, Oct. 2002.
[39] D. Tsoumako and N. Roussopoulos, "Adaptive probabilistic search for peer-to-peer networks," in *Proc. IEEE Int. Conf. P2P Computing*, 2003, pp. 102–109.
[40] S. Voulgaris, M. Jelasity, and M. van Steen, "A robust and scalable peer-to-peer gossiping protocol," in *Proc. AP2PC*, 2003, pp. 47–58.
[41] B. Yang and H. Garcia-Molina, "Improving search in peer-to-peer networks," in *Proc. ICDCS*, 2002, pp. 5–14.

Reaz Ahmed received the B.Sc. and M.Sc. degrees in computer science from the Bangladesh University of Engineering and Technology (BUET), Dhaka, Bangladesh, in 2000 and 2002, respectively. He received the Ph.D. degree in computer science from the University of Waterloo, Ontario, Canada, in 2007.

He is currently an Assistant Professor in the Department of Computer Science and Engineering, BUET. His research interests include wide area service discovery, loosely-coupled distributed databases and content-sharing peer-to-peer networks with focus on search flexibility, efficiency and robustness.

Raouf Boutaba (M'93–SM'01) received the M.Sc. and Ph.D. degrees in computer science from the University Pierre & Marie Curie, Paris, France, in 1990 and 1994, respectively.

He is currently a Professor of computer science at the University of Waterloo, Ontario, Canada. His research interests include network, resource and service management in wired and wireless networks.

Dr. Boutaba is the founder and Editor-in-Chief of the IEEE TRANSACTIONS ON NETWORK AND SERVICE MANAGEMENT and is on the editorial boards of several other journals. He is currently a distinguished lecturer of the IEEE Communications Society, and the chairman of the IEEE Technical Committee on Information Infrastructure and the IFIP Working Group 6.6 on Network and Distributed Systems Management. He has received several best paper awards and other recognitions including the Premier's Research Excellence Award.