Peer-to-Peer Unstructured Anycasting Using Correlated Swarms

Pushkar Patankar*, Gunwoo Nam*, George Kesidis*, Takis Konstantopoulos† and Chita R. Das*
*Dept of Computer Science and Engineering, The Pennsylvania State University,
University Park, PA 16802
Email: {patankar, gnam, kesidis, das}@cse.psu.edu
† School of Mathematical and Computer Sciences, Heriot-Watt University,
Edinburgh EH14 4AS, UK
Email: takis@ma.hw.ac.uk

***Abstract*—Over the recent years, social network analysis has received renewed interest because of the significant increase in the number of users relying on applications based on them. An important criterion for the success of any social-networking based application is the efficiency of search. In this paper, we propose and analyze a method of anycast search based on correlated communities or subgroups, *i.e.,* using group-to-group caching. It works by restricting search to peers that belong to communities which are highly correlated with the requested community. We analytically prove that our proposed method works better than basic random walk, which remains a widely used method for performing search in these networks. Indeed our experiments prove that the proposed method reduces the search time by as much as 30% to that based on random walk. Our experiments also indicate that the proposed method outperforms basic random walk even under considerable peer-churn.**

I. Introduction

Social networks have gained in popularity over the recent years because of their ability to represent complex relationships between entities interacting over the Internet. These relationships can be developed either due to an explicit action, such as a person joining a community of interest or a file sharing application requesting a desired object, or due to an activity seemingly unknown to the user, such as the spread of bots and viruses from one infected host to another. Social networks have been used to model various file sharing applications such as BitTorrent [9] and certain peer-to-peer streaming frameworks, social networking sites such as Facebook and MySpace, VoIP applications such as Skype [3], email chains, and generic communities of interest such as chatrooms. In general they can be used to model social behavior and to help understand how communities-of-interests are formed, *i.e.,* community-based networking.

Basic elements of social networks are:

- **Search**: Efficient resource discovery is a critical task in these networks. Search can be performed in three ways.
  - Centralized search employing an underlying infrastructure, *e.g.,* relying on Google to search for a particular .torrent file or a well known BitTorrent website in order to join a desired swarm.
  - Decentralized search employing a distributed search mechanism such as:
    - structured addressing scheme for peers and/or data objects with associated distance metrics, or
    - unstructured search based on techniques such as Random Walk (RW) or Limited Scope Flooding (LSF) or both.
  - Relying on superpeers, a set of densely connected peers in the network, in which search is achieved by biasing it towards them.
- **Transactions**: Once the identity (and hence location) of the desired resource is obtained, the requester and the provider may perform a transaction involving data delivery/exchange (email, chunk-swap, streaming, e-chat, VoIP, *etc*) and/or a payment (including a micropayment).
- **Feedback**: It is based on the quality of transaction outcome. Cumulative reputations, e-currency, *etc.,* can be used to aggregate transaction outcomes and rank peers in the network. Transactions typically affect future search, since peers tend to favor those that have favorable transaction outcomes.

We focus herein on decentralized peer-to-peer social networks which can be modeled as a graph whose links connect peers that directly communicate *through* the social network system[1]. Of course, *any* two peers can directly communicate in a generic fashion over the Internet underlay, but a social network (much smaller than the Internet as a whole) is a specific framework, which brings together peers with shared specific interests. Within a social network, each peer has a group of known "nearest neighbors" with which it directly communicates. Such social networking graphs often possess the "small world" phenomenon [28], [5], [12], [16] *i.e.,* the shortest path (measured in hops) between any two peers is less than some small number, famously six. So the graph is said to have six degrees of separation or a diameter of 6 hops.

A challenge is in *finding* such short paths in social networks so that two peers who are strangers (not neighbors in particular) can communicate. One peer may wish to find another that possesses certain attributes or information of

This work was supported by NSF CNS grant 0524202 and a Cisco Systems URP.

[1]Note that in this paper, the terms "peer", "user" and "node" are used interchangeably.

interest. To do so, the peer launches an *anycast query* in the social network via its neighbors. To resolve such queries, different search frameworks have been designed which are broadly classified as structured and unstructured. A great deal of recent research activity was devoted to structured peer-to-peer file sharing systems based on hashing peer and data-object identifiers (presumed known herein) to a common key-space so that peers can resolve all data-objects whose keys are proximal to their own [25], [21]. Structured search (query forwarding/relaying) is aided by a metric in the key-space allowing peers to judge distances between themselves. There is some evidence that in the presence of peer and data-object churn, peer-to-peer distributed structured systems experience more overhead and a greater risk of network partition [4] than unstructured systems [18]. Structured systems were originally designed for somewhat more centralized and wholly privatized content distribution network (CDN) operation.

In this paper, we focus on unstructured systems (without a graphical metric structure). We assume search will be conducted under reverse-path forwarding (RPF) [10], [11], [8] so that the query path is loop free and that all peers on that path may be informed of its peer membership once the query succeeds. Thus, the topology of the social network is dynamic under RPF. If the reverse path communication also includes the information (peer identities or data-object) sought, then a distributed caching system results that is larger in membership, at least temporarily, than the peer community-of-interest (for that data-object). In the following, we do not consider such *data* caching or hybrid methods that employ publish-subscribe systems together with an unstructured search [26].

Throughout the paper we focus on community based social networks, where every community is identified using a set of keywords that best describes it [2][2]. Peers query for a community based on these keywords. Once an anycast query is resolved, the querying peer may become a member of that community. Keyword aggregation and difficulties in setting unified naming conventions are some of the important issues in using keyword-based techniques [14], [7], [23]. Note that we do not try to address these issues here, focusing instead on search issues. Specifically, we consider different search techniques namely, RPF "without caching", RPF with "peer-to-group caching" and RPF with "group-to-group caching". Our proposed search using group-to-group caching relies on correlations between groups to restrict a search to only those peers who might have a high chance of being a member of the community that is being queried or are likely to know someone who might be a member of that community. We also develop an analytical model, to estimate search latency in these flat unstructured social networks. Further, we analytically and experimentally prove that search using RPF with group-to-group caching can outperform search using RPF without caching, a commonly used technique. Our simulation results indicate that search using group-to-group correlations can reduce average search latency by as much as 30% when compared to search without caching. We also subjected these search strategies to heavy peer-churn and observed that RPF with group-to-group caching still performs better than RPF without caching.

The rest of the paper is organized as follows. In Section II we give a brief description of prior search techniques for unstructured networks. In Section III, we present an anycast search framework using group-to-group caching and motivate its use. In Section IV, we analyze the performance of individual queries and thereby the spread of communities (or content) in the network. We present the simulation results comparing different search techniques in Section V. Finally, we conclude with a summary and future work in Section VI.

II. Related Work

Unstructured systems often have no formal address space with which to conduct search, or a "flat" address space (as with Ethernet MAC addresses on the Internet), rendering it difficult to implement distributed routing protocols operating in the background (*e.g.,* distance-vector routing based on hop counts) whose performance is susceptible to peer churn. As a result, unstructured search is typically based on random walks (RWs) or branching processes (floods limited in depth (TTL) and breadth). Approaches based on hybrid RWs and LSF have also been proposed (see, *e.g.,* [26], [16], [17]) including randomized versions of the latter [15]. In [16], RWs are said to perform better than LSFs in the presence of peer "clustering", *i.e.,* when a sufficient number of "superpeers" exist in a social network each having a very large neighborhood of peers[3]. [24] showed that the total query rate is minimized for caching based on RPF for simplified model of unstructured search, wherein the number of peers caching peer subgroup s is proportional to the square root of the query rate for s. It has also been proposed to *bias* random walks or branching processes toward such superpeers, e.g., [6], [29], [22].

III. Different Search Strategies

In this section we discuss three different search strategies employed for performing search in flat unstructured networks, namely: RPF "without caching", RPF using "peer-to-group caching" and RPF using "group-to-group caching". We assume that all the three search strategies employ random walk.

Consider a community of $N \gg 1$ peers any two of which can contact one-another directly. That is, the underlying physical "graph" networking them is essentially fully connected as would be the case of a community of peers at the periphery of (*i.e.,* communicating over) the Internet. Anycast search for a specific subgroup of peers (possessing a certain set of attributes) begins by a querying peer selecting another in the community of N. If the queried peer is not a member of the subgroup itself, it will relay the query to another peer, and so on until a resolving peer is reached. Under RPF [11], [8], all peers that relay a query inscribe their identity on it. Once resolved, the query will flow back to the querying peer along the *peer* path on which it was forwarded, thereby allowing all participating peers to note which two peers (querying

[2]The terms group, subgroup, community and swarm are used interchangeably.

[3]In this case, it is sometimes said that the graph is said to have an out-degree distribution that is heavy tailed.

peer and resolving peer) now belong to the peer subgroup targeted by the query. Once the querying peer is aware of the resolving peer (*i.e.,* the query is resolved), the querying peer may directly contact the resolving peer, e.g., to request a data-object associated with the subgroup in the context of a file-sharing system.

Each peer, X, maintains a permanent finger table associating X to peers in communities to which X belongs to. So, without considering peer departures or finger table removals, each community is minimally connected as a tree with (bidirectional) links according to the permanent finger tables of its members. Under RPF "without caching", search is performed using a simple unbiased random walk. Once the query is resolved, the identities of resolving peer along with the queried community can be entered in the querying peer's primary finger table.

The second search strategy employs a secondary finger table along with the primary finger table. A peer's secondary finger table associates peers to communities to which it does not belong. To resolve a query, the peer first performs a lookup in its primary finger table. If this lookup fails, it searches the secondary finger table. If neither finger table is able to resolve the query, it is then relayed to the next peer by performing an unbiased random walk. Once the query is resolved, all the intermediate nodes along the reply path cache the identity of the resolving peer and the queried group to which it is associated. As group memberships within the network increase, as a result of querying, the size of secondary finger table naturally increases, thereby reducing the search time for future queries. We call this search strategy RPF using "peer-to-group caching". Privacy and security concerns might prevent a peer from divulging identities of the querier and resolver to all the intermediate relaying peers along the query path [11], [8]. In-turn, this would prevent secondary finger table entries from being learnt as a result of querying. Moreover, as the number of peers tend to be much larger than number of groups, a peer may have to incur considerable memory and computational cost to learn and maintain such secondary cache entries.

To address security and privacy concerns, and to reduce memory and computation overhead, we propose a search strategy RPF using "group-to-group caching". Under this approach, intermediate peers along the query path maintain group-to-group correlation caches associating correlated communities with each other, instead of maintaining a secondary finger table. To ensure privacy, suppose that onion routing is employed such that, peers more than two hops away are not aware of the query originator or forwarders [11]. This search strategy works by exploiting associations amongst groups or communities of interest, *e.g.,* a group of researchers with an interest in emerging nanotechnology applications may be correlated to one focused on biotechnology, since a key application for nanotechnology is drug delivery. Such correlations among group-to-group aggregates can not only help resolve queries but may involve less overhead (lower memory and look-up complexity) as compared to systems in which secondary caches store peer information.

A peer's correlation cache can be roughly viewed as a correlation matrix relating communities which are not in its primary cache, to communities which are. The former corresponds to the rows of the matrix and the latter to its columns. For example, suppose peer x receives a query from peer p for group G, which includes information about p's other interest groups, $\mathcal{G}_p$; where x is not a member of G so x needs to relay the query. If G is not present in x's primary cache (so that is not aware whether any of its neighbors are members of G), x would create a row for G in its secondary (correlation) cache (unless G is not already associated with a row). Similarly, elements of $\mathcal{G}_p$ that are not in the primary cache could become rows if they are not already so, and the remaining elements could be entered as column indices if they are not already so. The correlation cache is updated as follows: When a peer receives a query for G the elements of its row corresponding to columns indexed in $\mathcal{G}_p$ are incremented by a small quantity $\varepsilon > 0$. Once the sum of a column's entries exceeds 1, the column entries are normalized so they sum to 1. This normalization mechanism "ages" data in a simple first-order autoregressive manner. Related logic can remove rows from the matrix if its entries are all small. Also, if the *primary* cache grows too large, group entries could be removed, *e.g.,* according to a least recently used (LRU) rule or by considering the magnitude of the entries in corresponding columns of the secondary cache.

Just as secondary finger tables, group-to-group correlation caches are learnt as a result of the querying process. Search employing a group-to-group correlation cache would involve the following: A querying node, while generating a query, inscribes in the query its present group membership(s), which it thinks might be correlated with the requested group. Every intermediate node that receives the query updates the correlation matrix based on the groups included in the query, with the requested group. If an intermediate node is not a member of the requested group, it tries to forward the query to a peer in its primary finger table that belongs to that group and, failing this, a peer (again from its primary finger table) that belongs to the most correlated group according to the correlation cache. If it does not know such a peer, then it relies on unbiased random walk. Just as in the two previous search strategies, once the query is resolved, the querying peer can enter the resolving peer's identity and the associated group in its primary finger table.

Once the resolving peer's identity and the queried community is added in the querying peer's primary finger table, further queries to that community that are relayed by this querying peer can be resolved in one hop count. However, note that in time, such querying for different communities if not aged out appropriately, can saturate the querying peer's primary finger table. To study the effects of primary finger table saturation versus secondary cache learning on search, we decided to evaluate these search strategies under cases where, the identities of resolving peers are added to the querying peer's primary finger table only after the interested community is queried successively for multiple times, *c.f.,* Section V. Note that in such queries, where the identities of

resolving peer and the queried community is not added to the querying peer's primary finger table upon query resolution, the secondary caches of peers along the query path under both the search strategies RPF with peer-to-group caching and RPF with group-to-group caching are still updated. This delay in the updating of primary finger table ensures that peers keep track of communities which are of consistent interest, rather than those that are fleeting.

IV. Performance of individual queries under correlated search

Here we analyze number of hops that a query needs to traverse before it reaches a peer belonging to the requested community. For this purpose, we simply consider fixed-size graphs with static community memberships and no peer churn. We develop an analytical model that compares the expected hop counts under RPF "without caching" and RPF using "group-to-group" caching. These models are considered under two cases. In the first case, a fully connected graph is assumed, while in the second case, incomplete graphs are considered. We also investigate the expected number of queries required to form a subgroup comprising a specific number of peers.

A. Complete Graphs

Consider a set of $\mathcal{N}$ peers with $N = |\mathcal{N}|$. One of them launches a query to try to reach any one of a subset of $\mathcal{N}_s$ peers with $1 < N_s \equiv |\mathcal{N}_s| < N$. We are interested in the number of "hops" before a peer among the $\mathcal{N}_s$ subset is reached.

1) Search without replacement: First suppose the query records all peers who have previously handled it so that no peer receives the same query more than once, as would be the case under RPF [10]. This corresponds to choice without replacement. Otherwise, all peers are assumed chosen independently and uniformly at random. For $1 \leq h \leq N - N_s$, the probability that h peers are chosen by a query (including the final choice in $\mathcal{N}_s$) is

$$P_h(N, N_s) \quad = \quad Q_{h-1}(N, N_s)\frac{N_s}{N-h}, \tag{1}$$

where $Q_{h-1}(N, N_s)$ is defined as the probability of $h-1$ choices among $N - N_s - 1$ peers (*i.e.,* not including the peer that launched the query) without replacement, and is computed as

$$Q_{h-1}(N, N_s) \quad = \quad \prod_{i=1}^{h-1} \frac{N - N_s - i}{N - i}$$

where $\prod_{i=1}^{0} := 1$. The number of hops in this case is therefore a *negative hypergeometric* random variable whose mean is given by [19]:

$$\begin{aligned} H(N, N_s) &= \sum_{h=1}^{N-N_s} hP_h(N, N_s) \\ &= \frac{N}{N_s + 1}. \end{aligned} \tag{2}$$

Equation (2) relates mean hop counts to resolve a query when RPF without caching is used.

Now suppose that there is a set of peers $\mathcal{N}_c$ that are "correlated" with those numbering N_s, and

$$N_c \quad = \quad |\mathcal{N}_c|$$

so that $\mathcal{N}_c \cap \mathcal{N}_s := \mathcal{N}_{c\cap s} \neq \emptyset$ in particular, *c.f.*, condition (3). Once the set $\mathcal{N}_c$ is contacted, search is thereafter *restricted* to $\mathcal{N}_c$.

So, for restricted search without replacement, we can condition on the number of choices k in $\overline{\mathcal{N}_c \cup \mathcal{N}_s} := \mathcal{N}_{\overline{c\cup s}}$ to get the probability that h peers are chosen in a query as

$$\begin{aligned} &P_h^{(c)}(N, N_s, N_c, N_{c\setminus s}) \\ &= \sum_{k=0}^{h-1} Q_k(N, N_{c\cup s})\tilde{P}_{h-k}(N_s, N_c, N_{c\setminus s}), \end{aligned}$$

where $\tilde{P}_1(N_s, N_c, N_{c\setminus s}) = N_s/N_{s\cup c}$ and $\tilde{P}_i(N_s, N_c, N_{c\setminus s}) = P_i(N_c, N_{s\cap c})$ for $i > 1$. We define the mean search hops to resolve a query when RPF with group-to-group caching is used as

$$H^{(c)}(N, N_s, N_c, N_{c\setminus s}) = \sum_{h=1}^{N-N_s} hP_h^{(c)}(N, N_s, N_c, N_{c\setminus s}).$$

Obviously if $N_s \subseteq N_c$, then search outside of N_c is unnecessary once N_c is contacted.

Theorem 1. *Under the following correlation condition,*

$$\frac{N_{c\cap s}}{N_c} \quad > \quad \frac{N_s}{N}, \tag{3}$$

and if $N_s \not\subseteq N_c$ [4], constrained search is faster on an average, i.e.,

$$H^{(c)}(N, N_s, N_c, N_{c\setminus s}) \quad \leq \quad H(N, N_s).$$

Proof: As a consequence of the independence of selection,

$$\begin{aligned} &H^{(c)}(N, N_s, N_c, N_{c\setminus s}) = \\ &\qquad H(N, N_{c\cup s}) + \frac{N_{c\setminus s}}{N_{c\cup s}} H(N_c, N_{c\cap s}), \end{aligned}$$

where $N_{c\setminus s}/N_{c\cup s}$ is the probability that when the query reaches $\mathcal{N}_{c\cup s}$, it contacts $\mathcal{N}_{c\setminus s}$ before $\mathcal{N}_s$. Thus,

$$\begin{aligned} &H^{(c)}(N, N_s, N_c, N_{c\setminus s}) \\ &= H(N, N_s)\left(\frac{H(N, N_{c\cup s})}{H(N, N_s)} + \frac{N_{c\setminus s}}{N_{c\cup s}} \cdot \frac{H(N_c, N_{c\cap s})}{H(N, N_s)}\right) \\ &\leq H(N, N_s)\left(\frac{H(N, N_{c\cup s})}{H(N, N_s)} + \frac{N_{c\setminus s}}{N_{c\cup s}}\right) \quad \text{by (2) and (3)} \\ &= H(N, N_s)\left(\frac{N_s + 1}{N_{c\cup s} + 1} + \frac{N_{c\setminus s}}{N_{c\cup s}}\right) \quad \text{by (2)} \\ &\leq H(N, N_s). \end{aligned}$$

□

Theorem 1 proves that RPF using group-to-group caching is better than RPF without caching in fully connected graphs. Note that a "pathwise" argument used while computing equation (2) will not work here as some long paths h may so

[4] *i.e.,* $N_{s\setminus c} \geq 1 \Rightarrow N_{c\cup s} \equiv N_s + N_{c\setminus s} \geq N_c + 1$.

greatly reduce the remaining population of peers in the group $\mathcal{N}_{\overline{c\cup s}}$ from which to choose that restricting search to $\mathcal{N}_{c\cup s}$ (once contacted) would lengthen search rather than shorten it. This leads to the idea of path dependent search that basically updates the correlation condition (3) at every hop. But this mechanisms will likely have minimal benefit as such long paths h are intrinsically unlikely and will be killed-off in practice by TTL limitations.

2) Search with replacement: Finally, we briefly discuss the simpler case of unrestricted search with replacement. The probability of a query involving h hops is

$$\tilde{P}_h(N, N_s) \quad = \quad \left(\frac{N - N_s - 1}{N - 1}\right)^{h-1} \frac{N_s}{N - 1} \quad \text{for } h \geq 1,$$

i.e., a geometric distribution. Note the "-1" terms are due to the fact that peers do not forward to themselves. Thus,

$$\tilde{H}(N, N_s) \quad = \quad \sum_{h=1}^{\infty} h\tilde{P}_h(N, N_s) \quad = \quad \frac{N - 1}{N_s}. \qquad (4)$$

Note how similar a query's hop count in mean is to that of the case with replacement (recall Equation (2)). The differences in the probabilities P_h and $\tilde{P}_h$ of longer "paths" h negligibly effect the means H and $\tilde{H}$.

The proof of the following result is similar to that of Theorem 1 for the case of search without replacement and so is stated as a corollary:

Corollary 1. *Under (3), the mean time for search with replacement is also shorter when restricted.*

B. Incomplete Graphs

The discussion so far has assumed a complete (fully connected) graph. In [16], the authors argue that the distribution of the points visited by a random walk in a "well-connected" graph approximates that for a complete graph, *i.e.*, under uniform sampling. In general, however, the topology of the graph may have a significant effect on the conditions under which search by RPF with group-to-group caching will be faster on an average than that using RPF without caching. The following theorem illustrates this for an idealized case.

Theorem 2. *For a symmetric random walk on the circle, if $\mathcal{N}_s \subset \mathcal{N}_c$ or*

$$N_{c\setminus s} \quad < \quad \tfrac{1}{2}(N - N_s) = \tfrac{1}{2}N_{\overline{s}}, \qquad (5)$$

then constrained (to $\mathcal{N}_c$ once contacted) search is faster than unconstrained/free search (for $\mathcal{N}_s$) for every starting point in $\mathcal{N}_{\overline{s}}$.

Proof: Here as above, the case where $\mathcal{N}_s \subset \mathcal{N}_c$ ($\Leftrightarrow \mathcal{N}_{\overline{c}} \cap \mathcal{N}_s = \emptyset$) is obvious as we are simply removing the possibility of excursions back to $\mathcal{N}_{\overline{c}}$ (once $\mathcal{N}_c$ is contacted if the starting point is $\notin \mathcal{N}_c$).

Let $S := N_s$.

Without loss of generality, we will enumerate the nodes in $\mathcal{N}_{\overline{s}}$ as $1, 2, 3, ..., N-S$. Thus, nodes enumerated 0 or $N-S+1$ are both elements of the (assumed connected) set $\mathcal{N}_s$.[5] For a symmetric random walk X, let T_k^X be the first time $n \geq 0$ such that $X_n = k$. The expected first contact time of $\mathcal{N}_s$ starting from state $i \in \{0, 1, ..., N - S + 1\}$ is:

$$u_i \quad := \quad \mathsf{E}(T_0^X \wedge T_{N-S+1}^X \mid X(0) = i).$$

For search using RPF without caching, the forward equation governing u_i is simply

$$u_i \quad = \quad \tfrac{1}{2}u_{i-1} + \tfrac{1}{2}u_{i+1} + 1 \quad \text{for } 0 < i < N - S + 1,$$

with boundary conditions $u_0 = 0 = u_{N-S+1}$ [20]. Thus,

$$u_i \quad = \quad (N - S + 1)i - i^2 \quad \text{for } 0 \leq i \leq N - S + 1.$$

Now suppose the search is constrained to the connected set $\mathcal{N}_c$ whose nodes are, without loss of generality, enumerated $K, K+1, ...,$ where $0 < K < N - S + 1$. Constraining search to $\mathcal{N}_c$ amounts to making impossible the transition from K to $K - 1$.[6] Let Y be a thus constrained random walk in $\mathcal{N}_c$. So, the expected contact time of $\mathcal{N}_s$ under RPF with group-to-group caching, r_i, satisfies: if $K \leq i \leq N - S$ then

$$r_i \quad = \quad \mathsf{E}(T_{N-S+1}^Y \mid Y(0) = i) \quad =: \quad y_i,$$

else if $0 < i < K$ then

$$r_i = \mathsf{E}(T_0^X \wedge T_K^X \mid X(0) = i) + \mathsf{P}(T_K^X < T_0^X \mid X(0) = i)y_K.$$

Now, by similar forward equation arguments,

$$\mathsf{P}(T_K^X < T_0^X \mid X(0) = i) = i/K \qquad \text{for } 0 \leq i \leq K, \text{ and}$$
$$y_i = (N - S + 1 \ - K)^2 - (i - K)^2 \qquad \text{for } K \leq i \leq N - S + 1.$$

Thus,

$$r_i = \begin{cases} (N - S + 1 \ - K)^2 - (i - K)^2 & \text{if } K \leq i \leq N - S \\ Ki - i^2 + \frac{i}{K}(N - S + 1 \ - K)^2 & \text{if } 0 < i < K \end{cases}$$

By direct comparison in both cases of initial starting point for search, $i < K$ or $i \geq K$, we get that

$$u_i > r_i \quad \Leftrightarrow \quad \tfrac{1}{2}(N - S + 1) < K,$$

where, by definition, $K < N - S + 1$.[7] □

Hence Theorem 2 proves that RPF using group-to-group caching is better than RPF without caching for an example incomplete graph.

C. Time for creation of social groups

In this subsection, we simply note how to compute the mean "formation times" of social groups for the example of "complete graph" search dynamics. T_{N_s} represents the total number of hops of the $N_s - 1$ queries required to grow a swarm of size 1 to $N_s < N$. So, by Lemma 1 (attributed to Karp and stated and proved in the Appendix),

$$\mathsf{E}(T_{N_s}) \quad \leq \quad \int_1^{N_s} \frac{1}{H(N, N - z)} \mathrm{d}z. \qquad (6)$$

[5] Nodes enumerated 0 and $N-S+1$ are in fact the same in the case where $S = 1$.

[6] Alternatively, if the nodes of $\mathcal{N}_c$ are enumerated $..., K-1, K$, then the transition from K to $K+1$ would be impossible under constrained search.

[7] If $\mathcal{N}_c$ was enumerated $\{..., K-1, K\}$ the condition for smaller search time when constrained would have been $0 < K < \frac{1}{2}(N - S + 1)$.

This model assumes that at every stage, the next hop peer is chosen with equal probability from among the remaining set of peers and the finger table entries are set to be permanent, *i.e.,* no peer departures.

To exactly compute $\mathsf{E}(T_{N_s})$, one can use a forward equation argument [20] to get for $1 \leq n < N_s$,

$$\begin{aligned} g(n) &\equiv \mathsf{E}(T_{N_s}) \\ &= \sum_{h=1}^{N_s-n} [g(n+h)+1]P_h(N,n), \end{aligned}$$

where $g(N_s) = 0$ and $g(N_s - 1) = 1$. Thus,

$$\begin{aligned} g(N_s-2) &= [g(N_s-1)+1]P_1(N,N_s-2) \\ g(N_s-3) &= [g(N_s-2)+1]P_1(N,N_s-3) \\ &\quad + [g(N_s-1)+1]P_2(N,N_s-3) \\ &\quad \textit{etc.} \end{aligned}$$

Thus $g(n)$ represents the number of queries required to form a subgroup of size n.

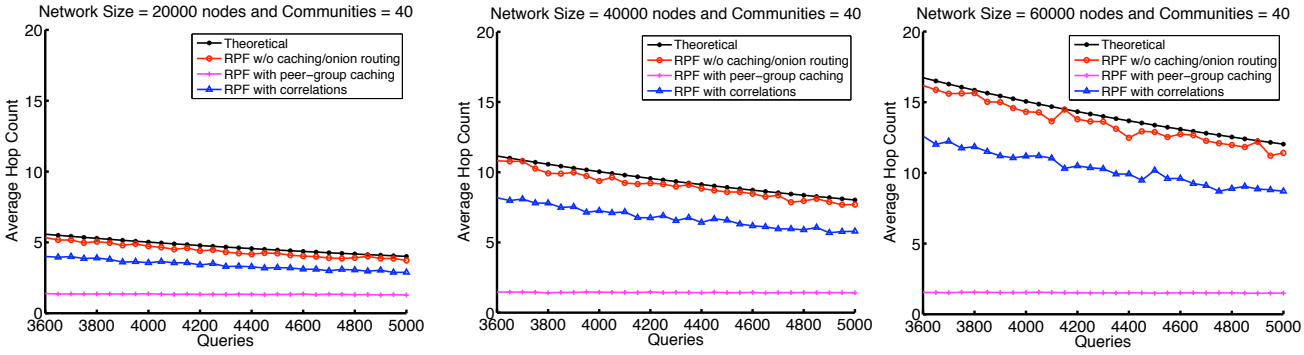
V. Simulation Study

In this section we experimentally investigate and compare search performance of the strategies described in Section III. To evaluate the accuracy of the analytical model described in Section IV, we also wanted to compare the "expected" hop count with those search strategies. These search strategies were analyzed under static and dynamic peer population. For this purpose, we created a social network of varying sizes comprising of 20,000, 40,000 and 60,000 peers. We introduced a varying number of communities in these networks. Every peer had about 5 other peers in its primary finger table (permanent cache). This formed the initial topology of the social network. At the start of the simulation, only about 50 peers were assigned to every community. During the course of simulation, we generated queries from peers for communities that they were not members of. We tracked the hop count using the three search strategies for every query that was being generated. To compare these search strategies with our analytical model, after every query we also derived the expected hop count using equation (2). To achieve high confidence intervals, we repeated the same experiment 100 times and averaged over them.

A lot of communities in social networks are highly correlated with each other, for instance through simple semantic proximity like in peer-to-peer file sharing systems as demonstrated in [13], [27]. To achieve highly correlated group memberships in our simulations, 85% of the queries generated were either for one of two given communities. Over the course of few hundred queries, the group membership for these two communities highly overlapped.

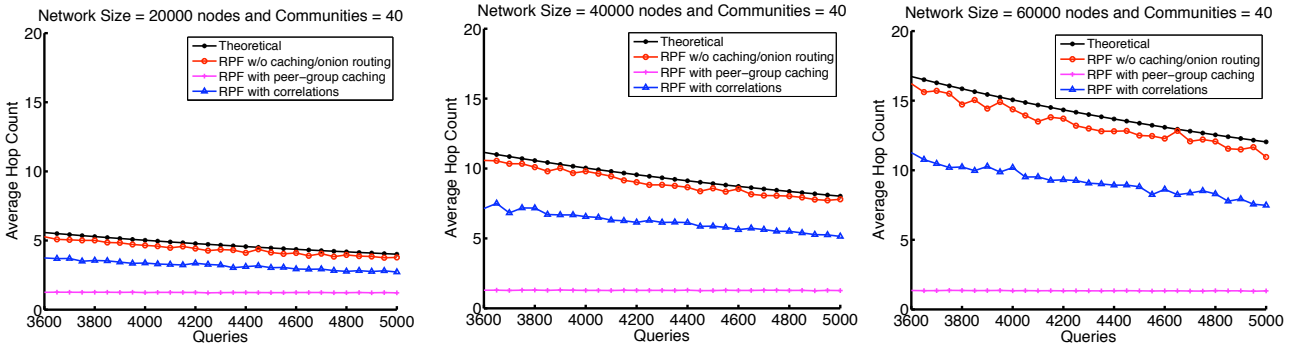
Three different sets of experiments were run to assess the impact of primary cache saturation and secondary cache learning on search performance (recall the discussion at the end of Section III), and peer-churn. The first two sets of experiments evaluated the impact of primary and secondary cache learning on search performance under fixed-sized graphs without peer churn. In the third experiment we analyzed these search techniques under dynamic peer-churn. For each experiment, we compared the average hop count for resolving a query versus the number of queries launched. To keep the comparisons fair, we evaluated the hop counts for only one of the correlated communities. We observed that increasing the number of communities did not affect search performance as a result of which, the number of communities were fixed to 40. During the course of simulations, it was observed that search using RPF with correlations (group-to-group caching) started performing better than RPF without caching only after a few hundred queries. This was specifically because the group-to-group correlation cache of peers needed to be trained to know which communities were correlated with each other. Moreover, after a few thousand queries, RPF with group-to-group caching performed only marginally better than RPF without caching. This was because, at this stage, a peer's primary finger table was rich with information so that, correlated cache was rarely used for making forwarding decisions. Nevertheless, we believe that in practical situations, peer churn will prevent such saturation of primary finger table. To prevent cluttering, we did not plot the confidence intervals (CI) but we observed sufficiently small CI at 95%. For instance, for one of the search strategies during one experimental setting when the expected hop count after launching 2450 queries for one of the correlated communities was around 12.5922, CI was +/- 0.0729.

In the first set of experiments as depicted in Figure 1, the identifies of the query resolver and the queried community were added in the querier's primary finger table after every successful *third* query. Note that, this changes the topology of the network. Search using RPF with group-to-group caching performed better than RPF without caching as the size of the network increased. In fact for a network of 60,000 peers, RPF with group-to-group caching performed 20% better than RPF without caching. Moreover, we observed that as the size of the social network increased, the hop count also increased. This is obvious since the graph became sparser and so it took longer to resolve the queries. In any case, however RPF with peer-to-group caching performed much better than the other two search strategies. But recall the discussion in Section III about the disadvantages of using peer-to-group caching. It was also observed that the analytical model of equation (2) was an upper bound. This is because, the analytical model does not take into account the information stored in the peer's primary finger table which biases the random walk. Similar trends were observed in the second set of experiments as shown in Figure 2, in which the identities of the query resolver and queried community were added after *five* queries. To this end we observed that RPF with group-to-group caching performed better by as much as 30% (Figure 3(c)) than RPF without caching. This was primarily because there was a delay in adding the query resolver in the querier's primary finger table while the queries were issued multiple times. This allowed for a larger period of time between the point when secondary group correlation cache was trained and the point when primary finger table saturated.



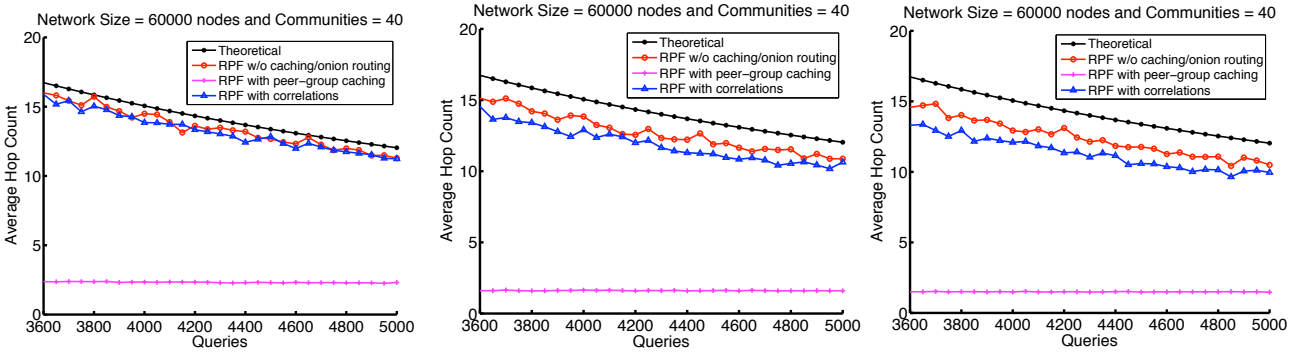
(a) (b) (c)

Fig. 1. Primary finger table updated after *three* successful queries



(a) (b) (c)

Fig. 2. Primary finger table updated after *five* successful queries



(a) Identities added after *every* query (b) Identities added after *three* queries (c) Identities added after *five* queries

Fig. 3. Peer Churn

In the third set of experiments we subjected the search strategies to peer-churn. To model peer-churn we introduced a mixed set of peers that stayed online for varying durations. During the depicted set of simulations, 50% of the peers were online for 50% of the time, 25% of the peers were online for 90% of the time, and the remaining peers were online for 10% of the time. The graphs in Figure 3 represent the scenarios when primary finger table entries were updated after one, three and five queries respectively. The results indicate that even under this heavy peer departure and arrival scenario, RPF with group-to-group caching still outperforms RPF without caching. We also evaluated a scenario where all of the peers were online 50% of the time, giving results similar to those depicted here.

VI. Conclusions and Future Work

The popularity of various applications relying on social networks has prompted renewed interest in their study and analysis. In this paper we attempt to optimize search by proposing a technique called RPF with group-to-group caching. It relies on correlated communities by restricting search to only those peers that might be correlated to the community that is being queried for. We also analytically prove that this technique

performs better than search relying on RPF without caching, a typically used technique. Furthermore, we compared the proposed search technique with two other search strategies. Though we found that search using RPF with peer-to-group caching performs the best, privacy, memory and computation overhead might not make this technique feasible. Given these conditions, we found that search using RPF with group-to-group caching reduced search time by as much as 30% when compared with RPF without caching. Further to this end we also demonstrated that our proposed technique can perform better than RPF without caching even under considerable peer-churn. In the future, we will evaluate search performance under more complicated scenarios involving dynamic peer population and dwindling swarm interest.

REFERENCES

[1] J. Aspnes, Z. Diamadi, and G. Shah. Fault-tolerant routing in peer-to-peer systems. In *Proc. ACM Symp. Principles of Distributed Computing*, 2002.

[2] L. Banks, S.F. Wu, and D. DeFigueiredo. Davis social links: A scalable social network for peer-to-peer communication. In *Proc. ACM SIGCOMM Workshop on Larg-Scale Attack-Defense (LSAD)*, Kyoto, Aug. 2007.

[3] S.A. Baset and H. Schulzrinne. An analysis of the Skype peer-to-peer Internet telephony protocol. http://www.citebase.org/cgi-bin/citations?id=oai:arXiv.org:cs/0412017, 2004.

[4] A. Binzenhofer, D. Stachle, and R. Henjes. On the stability of chord-based p2p systems. In *Proceedings of IEEE GLOBECOM*, St. Louis, MO, December 2005.

[5] R. Bollobas. *Random Graphs*. Cambridge University Press, 2001.

[6] Y. Chawathe, S. Ratnasamy, L. Breslau, N. Lanham, and S. Shenker. Making gnutella-like p2p systems scalable. In *Proc. Conf. on Application Technologies, Architectures and Protocols for Computer Communications*, 2003.

[7] M.-S. Chen, P.S. Yu, and K.-L. Wu. Decentralized consensus protocols with multiport communications. In *Proc. Conference on Distributed Computing Systems*, 1993.

[8] I. Clarke, S.G. Miller, O. Sandberg, B. Wiley, and T.W. Hong. Protecting freedom of information online with Freenet. *IEEE Internet Computing*, Jan. 2002.

[9] B. Cohen. BitTorrent Protocol Specification. http://www.bittorrent.com/protocol.html.

[10] Y. K. Dalal and R. M. Metcalfe. Reverse path forwarding of broadcast packets. *Commun. ACM*, 21(12):1040–1048, 1978.

[11] R. Dingledine, N. Mathewson, and P. Syverson. TOR: the second-generation onion router. In *Proc. 13th USENIX Security Symposium*, Aug. 2004.

[12] R. Durrett. *Random Graph Dynamics*. Cambridge University Press, 2007.

[13] F. Le Fessant, S. Handurukande, A.-M. Kermarrec, and L. Massoulie. Clustering in peer-to-peer file sharing workloads. In *Proc. IPTPS*, 2004.

[14] M.J. Fischer. The consensus problem in unreliable distributed systems (a brief survey). *Tecnical Report, Yale University*, 1983.

[15] R. Gaeta, G. Balbo, Bruell S, M. Gribaudo, and M. Sereno. A simple analytical framework to analyze search strategies in large-scale peer-to-peer networks. *Journal of Performance Evaluation*, 62(1-4):1–16, 2005.

[16] C. Gkantsidis, M. Mihail, and A. Saberi. Random walks in peer-to-peer networks. In *Proceedings of IEEE INFOCOM*, 2004.

[17] C. Gkantsidis, M. Mihail, and A. Saberi. Hybrid search schemes for unstructured peer to peer networks. In *Proceedings of IEEE INFOCOM*, 2005.

[18] S. He, S. Li, and H. MA. Effect of edge removal on topological and functional robustness of complex networks. *Physica A: Statistical Mechanics and its Applications*, 388(11), 2009.

[19] N.L. Johnson, A.W. Kemp, and S. Kotz. *Univariate Discrete Distributions*. John Wiley, 2005.

[20] S. Karlin and H.M. Taylor. *A Second Course in Stochastic Processes*. Academic Press Inc., New York, NY, 1981.

[21] P. Maymounkov and D. Mazieres. Kademlia: A peer-to-peer information system based on the XOR metric. In *Proceedings of IPTPS*, Cambridge, USA, March 2002.

[22] P. Patankar, G. Nam, G. Kesidis, and C. Das. Exploring anti-spam models in large scale VoIP systems. In *Proc. IEEE Int'l Conf. on Distributed Computing Systems (ICDCS)*, Beijing, June 2008.

[23] W. Ren, R.W. Beard, and E.M. Atkins. A survey of consensus problems in multi-agent coordination. In *Proc. ACC*, Portland, 0R, June 2005.

[24] K.W. Ross and D. Rubenstein. Tutorial on P2P systems. Available at http://cis.poly.edu/∼ross/papers/P2PtutorialInfocom.pdf, 2004.

[25] Ion Stoica, Robert Morris, David Liben-Nowell, David R. Karger, M. Frans Kaashoek, Frank Dabek, and Hari Balakrishnan. Chord: a scalable peer-to-peer lookup protocol for internet applications. *IEEE/ACM Trans. on Networking*, 11(1):17–32, 2003.

[26] W.W. Tepstra, J. Kangasharju, C. Leng, and A.P. Buchmann. Bubblestorm: Resilient, probabilistic, and exhaustive peer-to-peer search. In *Proc. ACM SIGCOMM*, Kyoto, Aug. 2007.

[27] S. Voulgaris, A.-M. Kermarrec, L. Massoulie, and M. Van Steen. Exploiting semantic clustering peer-to-peer content searching. In *Proc. Future Trends in Distributed Computing Systems*, 2004.

[28] D. Watts. *Small Worlds*. Princeton University Press, 1999.

[29] M. Zhong and K. Shen. Popularity-biased random walks for peer-to-peer search under the square-root principle. In *Proc. 5th International Workshop on Peer-to-Peer Systems (IPTPS)*, Santa Barbara, CA, Feb. 2006.

APPENDIX: A LEMMA ATTRIBUTED TO KARP [1]

Consider a process $X_0, X_1, \ldots$ with $\mathsf{E}X_0 < \infty$. Assume that $X_k \geq X_{k+1} \geq 0$ a.s. for all k. Let $\mathcal{F}_k \equiv \sigma(X_0, \ldots, X_k)$ where $\sigma(X_0, \ldots, X_k)$ is the σ-field generated by $X_0, \ldots, X_k$, and

$$\mu_k(z) \equiv \mathsf{E}(X_k - X_{k+1} | X_k = z, \mathcal{F}_k), \quad k \geq 0.$$

Lemma 1. *If there is a nondecreasing deterministic function $\mu(z)$ such that*

$$\mu_k(z) \geq \mu(z), \quad k \geq 0, \quad \textit{a.s.},$$

and

$$T := \min\{n \geq 0 : \ X_n \leq 1\},$$

then

$$\mathsf{E}T \leq \mathsf{E} \int_{X_T}^{X_0} \frac{dz}{\mu(z)}.$$

Proof:

$$\begin{aligned}
\mathsf{E}T = \mathsf{E} \sum_{k=0}^{T-1} 1 &= \mathsf{E} \sum_{k=0}^{\infty} \mathbf{1}(X_k > 1) \\
&= \mathsf{E} \sum_{k=0}^{\infty} \mathbf{1}(X_k > 1) \mathsf{E} \left(\frac{X_k - X_{k+1}}{\mu_k(X_k)} \mid \mathcal{F}_k \right) \\
&= \mathsf{E} \sum_{k=0}^{\infty} \mathsf{E} \left(\mathbf{1}(X_k > 1) \frac{X_k - X_{k+1}}{\mu_k(X_k)} \mid \mathcal{F}_k \right) \\
&= \mathsf{E} \sum_{k=0}^{\infty} \mathbf{1}(X_k > 1) \frac{X_k - X_{k+1}}{\mu_k(X_k)} \\
&= \mathsf{E} \sum_{k=0}^{T-1} \frac{X_k - X_{k+1}}{\mu_k(X_k)} = \mathsf{E} \sum_{k=0}^{T-1} \int_{X_{k+1}}^{X_k} \frac{dz}{\mu_k(X_k)} \\
&\leq \mathsf{E} \sum_{k=0}^{T-1} \int_{X_{k+1}}^{X_k} \frac{dz}{\mu(z)} = \mathsf{E} \int_{X_T}^{X_0} \frac{dz}{\mu(z)}.
\end{aligned}$$

□