Peer-To-Peer Unstructured Anycasting Using Correlated Swarms

Pushkar Patankar*, Gunwoo Nam*, George Kesidis*, Takis Konstantopoulos† and Chita R. Das*

*Department 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 Science
Heriot-Watt University,
Edinburgh, Scotland, UK
Email: takis@ma.hw.ac.uk

Abstract—Over the recent years, social network analysis has received renewed interest since there has been a great rise in the number of users relying on applications based on them. An important criteria 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 our proposed method reduces the search time by as much as 30% when compared to that based on random walk.

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 can 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 [4], 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 for 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 or both (LSF)
  - 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.** Based on 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 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 system1. 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 [29], [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 _finding_ 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 interest. To do

---

1Note that in this paper, the terms “peer”, “user” and “node” are used interchangeably.
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], [20]. Structured search (query forwarding/relaying) is aided by a metric in the key-space allowing peers to judge distances between themselves. This system was originally designed for somewhat more centralized and wholly privatized content distribution network (CDN) operation. Structured search suffers from the effects of peer churn (in CDNs, servers do not nominally depart the system, as peers may, and backup systems/servers can be deployed to recover from faults).

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 [3]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 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 search latency by as much as 30% when compared to search without caching.

The rest of the paper is organized as follows. In Section II we give a brief description of search techniques that have been proposed so far for unstructured networks. In Section III we present an anycast search framework using group-to-group correlations 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 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 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 churn³. 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 limited-scope flooding (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 the social network each having a very large neighborhood of peers². [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], [31], [22]. Other results on structured search are concerned with (layer 4) TCP or transmission-bandwidth issues and associated games, see, e.g., [18], [21], [30].

III. 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 ≥ 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

²But see [28] for a recent development along these lines.
³In this case, it is sometimes said that the graph is said to have an out-degree distribution that is heavy tailed.
all participating peers to note which two peers (querying 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 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 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, Gp, 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 Gp, 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 Gp are incremented by a small quantity ε > 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. 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, incomplete graphs are considered. Theorem 1 proves that RPF using “group-to-group” caching is better than RPF “without caching” (basic random walk) in fully connected graphs. Theorem 2 proves this condition for an example incomplete graph. 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 $N$ peers with $N = |N|$. One of them launches a query to try to reach any one of a subset of $N_s$ peers with $1 < N_s ≤ |N_c| < N$. We are interested in the number of “hops” before a peer among the $N_s$ 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 $1 ≤ h ≤ N − N_s$, the probability that $h$ peers are chosen by a query (including the final choice in $N_c$) is

$$P_h(N, N_s) = Q_{h−1}(N, N_s) \frac{N_s}{N−h},$$

where the probability of $h−1$ choices among $N−N_s−1$ peers (i.e., not including the peer that launched the query) without replacement is

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

where $\prod_{i=1}^{0} = 1$. The number of hops in this case is therefore a negative hypergeometric random variable whose mean is defined as:

$$H(N, N_s) = \sum_{h=1}^{N−N_s} hP_h(N, N_s) = \frac{N}{N_s + 1},$$

see [1] for a discussion on negative hypergeometric random variables.

Now suppose that there is a set of peers $N_c$ that are “correlated” with those numbering $N_s$ and

$$N_c = |N_c|$$

so that $N_c ∩ N_s = N_{c|s} \neq 0$ in particular, c.f., condition (3). Once the set $N_c$ is contacted, search is thereafter restricted to $N_c$.

So, for restricted search without replacement, we can condition on the number of choices $k$ in $N_c \cup N_s = N_{ex}$ to get that the probability that $h$ peers are chosen in a query is

$$P_h(N, N_s, N_c, N_{c|s}) = \sum_{k=0}^{h−1} Q_k(N, N_{c|s})P_{h−k}(N_s, N_c, N_{c|s}),$$

where we note simply how $N_{c|s} = N_{c|s} − N_c = N_s + N_{c|s} − N_c$. We define the mean search hops in this case as

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

**Theorem 1.** Under the following correlation condition,

$$\frac{N_{c|s}}{N_c} > \frac{N_s}{N},$$

constrained search is faster on average, i.e.,

$$H(N, N_s, N_c, N_{c|s}) ≤ H(N, N_s).$$

**Proof:** As a consequence of the independence of selection (as the strong Markov property),

$$H(N, N_s, N_c, N_{c|s}) = H(N, N_{c|s}) + \frac{N_{c|s}}{N_{c|s}}H(N_c, N_{c|s}),$$

where $N_{c|s}/N_{c|s}$ is the probability that when the query reaches $N_{c|s}$, it contacts $N_{c|s}$ before $N_s$. Thus,

$$H(N, N_s, N_c, N_{c|s}) = H(N, N_s) + \frac{N_{c|s}}{N_{c|s}}H(N_c, N_{c|s}),$$

$$≤ H(N, N_s) + \frac{N_{c|s}}{N_{c|s}}H(N_c, N_{c|s})$$

by (2) and (3),

$$= H(N, N_s) \left( H(N, N_{c|s}) + \frac{N_{c|s}}{N_{c|s}} \right)$$

$$≤ H(N, N_s),$$

by (2).

Note that a “pathwise” argument used in the previous theorem will not work as some long paths $h$ may so greatly reduce the remaining population of peers in the group $N_{c|s}$ from which to choose that restricting search to $N_{c|s}$ (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 query involving \( h \) hops is

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

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

\[
\hat{H}(N, N_s) = \sum_{h=1}^{\infty} h \hat{P}_h(N, N_s) = \frac{N - 1}{N_s}.
\]

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

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

**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], they 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., uniform sampling. In general, however, the topology of the graph may have a significant effect on the conditions under which search by constrained random walk will be faster on average than that by free random walk. The following theorem illustrates this for an idealized case.

**Theorem 2.** For a symmetric random walk on the circle, if \( N_s \subset N_c \) or

\[
N_c \setminus s < \frac{1}{2} (N - N_s) \equiv \frac{1}{2} N_{\pi},
\]

then constrained (to \( N_c \) once contacted) search is faster than unconstrained/free search (for \( N_s \)) for every starting point in \( N_{\pi} \).

**Proof:** Here as above, the case where \( N_s \subset N_c \) \( (\Rightarrow N_c \cap N_s = \emptyset) \) is obvious as we are simply removing the possibility of excursions back to \( N_{\pi} \) (once \( N_c \) is contacted if the starting point is \( \notin N_c \)).

Let \( S := N_s \).

Without loss of generality, we will enumerate the nodes in \( N_{\pi} \) as \( 1, 2, 3, \ldots, N - S \). Thus, nodes enumerated 0 or \( N - S + 1 \) are both elements of the (assumed connected) set \( N_c \). For a symmetric random walk \( X \), let \( T^X_k \) be the first time \( n \geq 0 \)

such that \( X_n = k \). The expected first contact time of \( N_s \) starting from state \( i \in \{0, 1, \ldots, N - S + 1\} \) is:

\[
u_i := E(T^X_0 \wedge T^X_{N - S + 1} \mid X(0) = i).
\]

For an unconstrained symmetric random walk, the forward equation governing \( u_i \) is simply

\[
u_i = \frac{1}{2} u_{i-1} + \frac{1}{2} u_{i+1} + 1 \quad \text{for } 0 < i < N - S + 1,
\]

with boundary conditions \( u_0 = 0 = u_{N - S + 1} \) [19]. Thus,

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

Now suppose the random walk is constrained to the connected set \( N_c \) whose nodes are, without loss of generality, enumerated \( K, K + 1, \ldots \), where \( 0 < K < N - S + 1 \). Constraining search to \( N_c \) amounts to making impossible the transition from \( K \) to \( K - 1 \). Let \( Y \) be a thus constrained random walk in \( N_c \). So, the expected contact time of \( N_c \) under constrained search, \( r_i \), satisfies: if \( K \leq i \leq N - S \) then

\[
r_i = E(T^Y_{N - S + 1} \mid Y(0) = i) =: y_i,
\]

else if \( 0 < i < K \) then

\[
r_i = E(T^X_0 \wedge T^X_K \mid X(0) = i) + P(T^X_K < T^X_0 \mid X(0) = i) y_K.
\]

Now, by similar forward equation arguments,

\[
P(T^X_K < T^X_0 \mid X(0) = i) = i/K \quad \text{for } 0 \leq i \leq K, \text{ and}
\]

\[
y_i = (N - S + 1 - K)^2 - (i - K)^2 \quad \text{for } K \leq i \leq N - S + 1.
\]

Thus,

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

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

\[
u_i > r_i \Leftrightarrow \frac{1}{2} (N - S + 1) < K,
\]

where, by definition, \( K < N - S + 1 \).

C. Time for creation of social groups

In this subsection, we note how to compute the mean “formation times” of social groups for the example of “complete graph" search dynamics. Let \( x(t) \) be the number of peers at “time" \( t \) that possess certain attributes under consideration (belong to a specific subgroup), i.e., after \( t \) queries for it. A query corresponds to a unit of discrete-time. Taking the case where there are \( x(t) = n \) informed peers, there are \( N - n \) peers for the querier to choose from, i.e., except the querier himself.

So, by Lemma 1 (attributed to Karp and stated and proved in the Appendix), the expected number of queries to inform a subgroup \( s \), for \( N_s < N \), is

\[
E(T_{N_s} \mid x(0) = 1) \leq \int_1^{N_s} \frac{1}{H(N, N - z)} \, dz,
\]

Alternatively, if the nodes of \( N_c \) are enumerated \( \ldots, K - 1, K \), the condition for smaller search time when constrained would have been \( 0 < K < \frac{1}{2} (N - S + 1) \).

---

\(^3\)Nodes enumerated 0 and \( N - S + 1 \) are in fact the same case in the same where \( S = 1 \).
where $H(N, N - z)$ is the number of hops required to inform $z$ peers about subgroup $s$. 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 $E(T_{N_s} \mid x(0) = n)$, one can use a forward equation argument [19] to get for $1 \leq n < N_s$,

$$g(n) \equiv E(T_{N_s} \mid x(0) = n) = \sum_{h=1}^{N_s-n} (g(n + h) + 1)P_h(N, n),$$

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

$$g(N_s - 2) = g(N_s - 1) + P_1(N, N_s - 2)$$

$$g(N_s - 3) = g(N_s - 2) + P_1(N, N_s - 3) + g(N_s - 1) + P_2(N, N_s - 3)$$

These equations were derived with an 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. Some through simple semantic proximity like in peer-to-peer file sharing systems as demonstrated in [13], [27], and some not. In social networking sites where a person who might be a member of community for UC Berkeley Computer Science might also be a member of community UC Berkeley alumni. 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 500 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). For every 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 community and that too for 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 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 plus/minus 0.0729.

In the first set of experiments as depicted in Figure 1, the identities of the query resolver and the queried community are added in the querier’s primary finger table after every successful query. Note that, this changes the topology of the network. In this set of experiments, search using RPF with group-to-group caching performed marginally 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. Overall it was observed that the peer’s primary finger table saturated faster than the rate at which its group-to-group correlation cache could learn about group correlations as a result of which RPF with peer-to-group caching performed almost similar to RPF without caching.

In the second and third set of experiments depicted in Figures 2 and 3 respectively, it was observed that search using RPF with group-to-group caching performed better than RPF without caching. In the second set of experiments, identities of resolving peer and queried community are added in the querier’s primary finger table only after the querier launches three queries for that community. In other words, even though the query was resolved twice for that community, the resolved identities were not added in the querier’s primary finger table.

In the same way, this information was only added after 5 queries in the third set of experiments. The graphs depict the hop counts only when those identities were added in the
queer’s finger table. In these set of experiments, as the size of the social networks increased, hop counts also increased. Also, for the third set of simulation results as shown in Figure 3, RPF with group-to-group caching performed much better than RPF without caching by as much as 30% whereas this performance gain was around 20% for the second set. 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.

VI. CONCLUSIONS

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.

REFERENCES


APPENDIX: A LEMMA ATTRIBUTED TO KARP [2]

Consider a process $X_0, X_1, \ldots$ with $EX_0 < \infty$. Assume that $X_k \geq X_{k+1} \geq 0$ a.s. for all $k$. Let $F_k \equiv \sigma(X_0, \ldots, X_k)$, and

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

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

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

and

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

then

$$ET^k \leq E \int_{X_T}^{X_0} dz \int_{\mu(z)}^{\mu_k(z)} dz.$$ 

Proof:

$$ET^k = E \sum_{k=0}^{T-1} 1 = E \sum_{k=0}^{\infty} 1(X_k > 1) = E \sum_{k=0}^{\infty} 1(X_k > 1) \left( \frac{X_k - X_{k+1}}{\mu_k(X_k)} \right) \bigg| F_k$$

$$= E \left( 1(X_k > 1) \frac{X_k - X_{k+1}}{\mu_k(X_k)} \bigg| F_k \right) = E \sum_{k=0}^{\infty} \left( 1(X_k > 1) \frac{X_k - X_{k+1}}{\mu_k(X_k)} \bigg| F_k \right)$$

$$= E \sum_{k=0}^{\infty} 1(X_k > 1) \frac{X_k - X_{k+1}}{\mu_k(X_k)}$$

$$= E \sum_{k=0}^{T-1} \frac{X_k - X_{k+1}}{\mu_k(X_k)}$$

$$= E \left( \sum_{k=0}^{T-1} \frac{X_k - X_{k+1}}{\mu_k(X_k)} \right) = E \left( \int_{X_{k+1}}^{X_k} dz \int_{\mu(z)}^{\mu_k(z)} dz \right).$$

$$\square$$