SocialCDN: Caching Techniques for Distributed Social Networks

Lu Han*, Magdalena Punceva†, Badri Nath*, S. Muthukrishnan*, Liviu Iftode*

Department of Computer Science.
Rutgers, The State University of New Jersey
* {*luhan, badri, muthu, iftode*}*@cs.rutgers.edu*, † *magdalena.punceva@gmail.com*

***Abstract*—Distributed online social networks (DOSN) have been proposed as an alternative to centralized Online Social Networks (OSN). In contrast to centralized OSN, DOSNs do not have central repository of all user data, neither impose control regarding how users data will be accessed. Therefore, users can keep control of their private data and are not at the mercy of the social network providers. However, one of the main problems in DOSNs is how to efficiently disseminate social updates among peers. In our previous work, we proposed Social Caches for social updates dissemination in DOSN. However, the selection of social caches requires knowledge about the entire social graph. In this paper, we propose four fully distributed social cache selection algorithms, and evaluate their performance on five well known graphs. Using simulations we show that these algorithms perform almost as good as the centralized best known approximation algorithm would do. These distributed caching techniques can be used as a basis for various applications such as those that represent fusions of social and vehicular networks.**

I. Introduction

Popular Online Social Networks (OSN), such as Facebook, Twitter, and Google+ have changed the way people communicate and share information. This revolution in human interaction through social media has brought to the forefront the issues of ownership and control of user generated data. In the case of centralized "Online Social Networking" sites, once personal content is stored in an OSN, users give up the control of their own data. Distributed Online Social Networks (DOSN), such as Diaspora [1], PeerSoN [2], and PrPI [3], have been recently proposed as an antidote to centralized OSNs. DOSN is a P2P infrastructure that supports the features of OSNs in a distributed way. DOSNs enable users to host and organize their personal profiles and social connections while retaining full control over their own data. More precisely, within DOSNs, users can manage their data, control with whom to share the data, and determine which third parties can access their data for online advertisement. An exhaustive list of the existing DOSNs is maintained on the Wiki page [4].

The obvious advantages of DOSNs over centralized OSNs are counter-balanced by several challenges in deploying the DOSNs. Critical among them is the need for a scalable *social update dissemination* service. A Social Update is defined as any social content that users share with their friends, such as changes to profile information, wall postings, pictures, videos, status updates, links, messages, tweets, etc. Sending and browsing social updates represent a significant portion of the activities in OSNs, and contribute to the majority of the network traffic. According to a recent Facebook survey [5], on an average day, 15% of Facebook users update personal status; 22% comment on others' posts or status; 20% comment on others' photos; 26% "Like" other users' contents; 10% send another user a private message. Social network sites generate more network traffic stream than most of the other websites. 28% of the Internet traffic is coming from Social Media [6], ranking as the second source of the Internet traffic after the search engines.

While a CDN [7] can effectively reduce the load on a centralized server, the feasibility of the DOSNs critically depends on the efficiency of social updates delivery and on measures taken to support good data availability. The former can be achieved through caching in the social network, while the latter can be achieved through redundancy.

In a previous paper, Social Butterfly [8], we proposed *Social Caches*, which are nodes selected to act as local bridges for their friends in order to reduce the number of connections necessary to collect the social updates in DOSNs. However, the solution presented in [8]was not fully distributed, as it required knowledge about the entire social graph topology.

In this paper, we propose SocialCDN, an efficient social content distribution system based on a distributed social cache selection mechanism that does not require global knowledge of the underlying social graph. Within the context of SocialCDN, we propose and analyze four distributed cache selection algorithms: *Randomized* algorithm, *Triads Elimination* algorithm, *Span Elimination* algorithm, and *Social Score* algorithm. The Randomized algorithm, used as a baseline for comparison, elects caches based on a uniform probability. In the two elimination algorithms (Triad Elimination and Span Elimination algorithms), every pair of nodes select a cache in the local neighborhood in a greedy manner and afterwards minimize the total number of caches. These algorithms are applicable to any arbitrary graph. The *Social Score* algorithm is designed specifically for social graphs and takes into account measures such as centrality of a node in its local network. A node decides whether it will become a social cache or not based on its local properties. We evaluate the performance of these four cache selection algorithms on five different graphs that can be grouped into three categories: *Social, Semi-Social, and Non-Social Network*, and discover that the *Span*

Elimination algorithm outperforms the others and is the closest to the centralized Approximate NDS (Neighbor-Dominating Set) algorithm [8], which we use as a lower bound. Besides the quantitative differences that we extensively analyze through simulation, we also discuss the qualitative differences of the proposed algorithms.

We discuss related work in Section II. Section III introduces SocialCDN model and the approach, and Section IV clarifies the notations. Section V presents the four distributed social cache selection algorithms. In Section VI, we discuss the properties and measurements of the five social graphs we used for evaluation. The evaluation on the five graphs is presented in Section VII. Section VIII presents the mechanism to maintain selected caches. Section IX discusses our future directions, and Section X concludes our work.

II. Related Work

The current deployments of DOSNs can be divided into three categories: *(i). Federation*, which requires social network providers to agree upon standards of operation in a collective fashion. Federated social networks enable users to share their social contents in one OSN with friends from other OSNs. *(ii). OSNs over unstructured P2P underlay*, which have users' personal data distributed among multiple servers, and utilize a lookup server for bootstrapping functionalities [9], [10] and [3]. *(iii). OSNs over structured P2P underlay*, which utilize DHT underlays such as My3 [11] and Peerson [2].

The approaches presented in [10] and [11] are the closest to our approach. In [10], the authors propose a P2P OSN that assumes that a user's updates will reach another user if there is a path in the overlay network between the two users. Sending content through several links makes the system more vulnerable to failures and requires stronger incentives for intermediary peers to store and transfer the content. In our SocialCDN approach, content can reach the other party through a common friend, which means within at most two-hops. My3 [11] is a P2P based OSN that relies on users' geographical locations and online time statistics to provide availability. It uses the original Dominating Set (DS) problem to minimize the number of replicas. Our work does not consider availability. Instead, we use the Neighbor Dominating Set (NDS) problem to minimize the number of selected cache nodes in order to improve social update dissemination efficiency. NDS is sufficiently different from the DS problem to render the known DS approximation algorithms unusable in the NDS case.

There exists other data dissemination schema such as gossip protocol [12], epidemic routing [13] and probabilistic routing based on prediction [14]. Giuliano Mega, et al. [12] propose a modified gossip protocol for data dissemination in DOSN. They utilize vertex centrality and clustering to select where to send the gossip message. However, gossip protocol and epidemic routing schema rely on flooding technique, which do not help in reducing network traffic. Prediction based methods do not fit for social updates distribution, since publishing and browsing in OSNs, although usually follow certain daily patterns, are very hard to predict.

III. Model and Approach

In this section, we first present the underlying models on which the SocialCDN is relying. Then, the second half of the section presents the SocialCDN approach for social updates dissemination in distributed OSN.

A. Model

SocialCDN works by making the following assumptions about the underlying communication network, social graph, and trust.

- **Network and Communication:** For simplicity, we assume a static network and a standard synchronous round-based distributed communication model. In particular, we assume that once selected the cache nodes are always available.
- **Social Graph**: SocialCDN assumes that each node knows its immediate friends, and its two-hop friends. Knowing two-hop neighbors is a common feature in today's OSNs, which allows users to see friends-of-friends' comments, and to expand their networks of friends. We also assume that social links in the social graph are equal; no edge weights to represent social tie strength, distance, delays or communication loads.
- **Trust and Altruism**: SocialCDN assumes friends are trust-able and altruistic. In other words, they do not misbehave and are willing to cache content for their friends without compensation. Furthermore, we assume that the social cache altruism applies to friends only, for whom personal bandwidth and storage can be sacrificed. Reputation and economic models are outside the scope of the paper.

With all these simplifying assumptions, our problem is still an NP-complete optimization problem.

B. Approach

Within a pure DOSN, dissemination of the social updates among friends necessitates $O(n^2)$ (n is the number of friends) network connections to be established, which can significantly degrade the user-perceived performance compared to a centralized DSN. SocialCDN proposes social caches to reduce the total number of P2P connections necessary for social updates dissemination within a DOSN. Nodes push their social updates to, and fetch those of their friends from the social caches they are connected to. If the number of selected caches is significantly less than the number of friends, then the use of social caches will significantly reduce the total connections when compared to a fully P2P pull/push approach. This is why, in this paper, the goal is to minimize the number social caches using fully distributed algorithms.

Typically, users in OSNs can be producers of social updates, as well as consumers of social updates produced by their friends. These properties of OSN require the selected social caches to be friends with both nodes. Therefore, the following

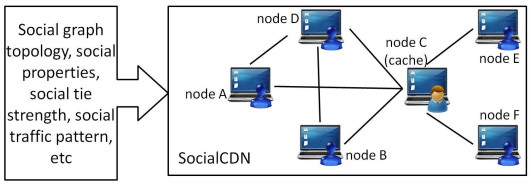


Fig. 1. An example of socialCDN network.

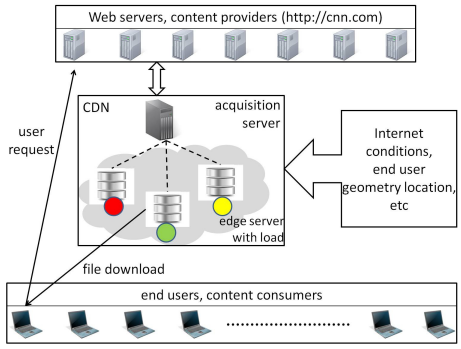


Fig. 2. An example architecture of Content Delivery Network (CDN).

constraint holds when selecting social caches: *social caches are a subset of vertices in the graph, whereas a vertex is either a social cache or connected to a social cache, whereas any pair of friends must have at least one common friend who is a social cache, if none of them is a social cache.*

Although both SocialCDN and CDN rely on caching schema for content delivery, the selection methods they employ are completely different. CDN technology selects the edge server depending on geographic locations of the users, edge server traffic loads, and network conditions such as bandwidth, as shown in Figure 2. SocialCDN decides the placement of social caches based on social graph topology, social properties, social tie strength, social traffic pattern, etc., as shown in Figure 1. The distributed algorithms used for selecting social caches are discussed in Section V.

IV. Notations

We use $G = (V, E)$ to represent an undirected graph, where V is the set of vertices, and E is the set of edges. The following terms, *social network user*, *vertex* and *node* are interchangeably used to represent a vertex in the graph.

We also define the following notations:

- $deg(v)$ to be the degree of node v.
- $N(v)$ to represent the set of immediate neighbors of v.
- The set of *edges covered by node* v, S_v, is the subset of E and is composed of any $e = (\alpha, \beta) \in E$ iff $v \in (N(\alpha) \cap N(\beta)) \cup \{\alpha, \beta\}$. An edge e can be covered by multiple nodes depending on the topology of the graph.
- $size(S_v)$ denotes the number of edges in S_v.
- $T(v)$ is the number of *Transitive Triads* a node v is part of. One of the basic unit of social network theory is *Dyad*, which is a pair of parties who may or may not share a social relation. A *Triad* is a set of three parties and consists of three dyads. A triad is *transitive* if when there is a tie (social relationship) between party A and party B, and between B and a third party C, then there is also a tie between A and C.
- $CN(u, v)$ or $CN(e)$ denotes the set of common neighbors of edge e, where u and v are endpoints of e.

During the execution of a cache selection procedure:

- A node v belongs to one of the following categories:
  i). *Black*: v is selected as a social cache;
  ii). *Grey*: every edge in S_v is *covered* by social caches but v is not selected as a social cache;
  iii). *White*: v is not a social cache and there is at least one edge in S_v that has not been covered.
- The edges covered by a social cache are *green* edges, others (uncovered edges) are *red* edges.
- $span(v)$ is the number of *red* edges in S_v. At the beginning of a selection procedure, $span(v) = size(S_v)$, but decreases as algorithm executes, and will be 0 when the algorithm terminates.

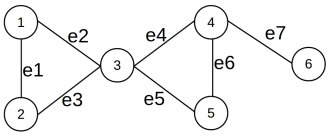


Fig. 3. An example graph to illustrate the cache selection algorithms.

V. Distributed Cache Selection Algorithms

In this section, we present four distributed algorithms to solve the following social cache selection problem: *find the smallest set of cache nodes such that each edge is connected by at least one social cache if none of its endpoints is a social cache.* Due to the similarity to the *Dominating Set* problem, it is also referred to as a *Neighbor-Dominating Set* problem [8] and is defined as:

> " The *Neighbor-Dominating Set* of graph $G = (V, E)$ is the set $S \subseteq V$ of vertices such that for each edge $(u, v) \in E$, there exists a $w \in S$ satisfying $w \in (N(u) \cap N(v)) \cup \{u, v\}$. Given a graph $G = (V, E)$, find a *Neighbor-Dominating Set* of smallest size."

A. Randomized Algorithm

We use the Randomized algorithm as a baseline for evaluation and comparison with the other three. The algorithm works by letting node v to elect itself as social cache with a threshold probability θ. More precisely, each node applies the distributed algorithm in the following steps:

a. calculate $span(v)$,

b. if $span(v)$ == 0, (the edges in S_v are all marked as *green*), node v makes itself as *grey* and quits the loop.

c. if $span(v) > 0$, randomly generate a number $p(v) \in [0, 1]$. If $p(v) > \theta$, node v elects itself as a social cache, marks itself as *black*, and marks all edges in S_v as *green*, informs its neighbors about its election and quits the loop.

We use the graph in Figure 3 as an example to explain all the distributed social cache selection methods. Given the

TABLE I
$T(v)$, $size(S_v)$, $ss(v)$, AND $ss\_prob(v)$ OF EACH NODE FOR THE GRAPH IN FIGURE 3

Node	1	2	3	4	5	6
Transitive Triads	1	1	2	1	1	0
$size(S_v)$	3	3	6	4	3	1
Social Score	0.0	0.0	16/3	12/3	0.0	1.0
Social Score Prob	0	0	13/16	9/12	0	0

social graph in Figure 3, we assume node i generates a random number $p(i)$ in each iteration, and consider the following scenario: $p(3) > \theta$, and $p(i) < \theta$ for $i = 1, 2, 4, 5, 6$. In this case, during the first iteration, node 3 elects itself as a social cache, and marks edges $\{e1, e2, e3, e4, e5, e6\}$ as *green*. Next, randomly electing a node from $\{4, 6\}$ will cover the whole graph. It is clear that the performance of this method is determined by the predefined θ. The evaluation will be discussed in Section VII-A.

B. Triad Elimination Algorithm

The Triad Elimination algorithm and the Span Elimination algorithm presented in Section V-C have two phases: the *selection* phase, and the *elimination* phase. During the *selection* phase, a social cache is selected for every edge based on the number of *transitivity triads* a node is a part of, or the *span* of a node, respectively. During the *elimination* phase, the redundant caches are being reduced as much as possible. Both algorithms terminate within a **constant** number of rounds, i.e., two rounds.

In the *cache selection* phase, each node v calculates $T(v)$ as the number of transitive triads it is part of. It is relatively easy to figure out this number of triads once the node knows its two-hop neighbors. Next, for each edge $e = (u, v)$, u and v exchange their $T(v)$ and $T(u)$, and select the one with higher T to be the *Temporary Social Cache* for the edge, $TSC(e)$. In case $T(u) = T(v)$, the choice is made randomly.

The selection phase enables all edges to be covered, i.e., *green*, however, the number of caches is not optimal. For example, in a graph with three nodes A, B and C forming a transitive triad, it is possible to select all three nodes as caches in the worst case, as $T(v) = 1$ for each of them. In fact, choosing one node as cache is the optimal for this graph.

The *elimination* phase reduces the redundancy by utilizing the fact that *every common neighbor of an edge can also be a cache for that edge besides the two endpoint nodes*. The temporary cache for each edge contacts all the common neighbors of $e = (u, v)$, checks how many times each of them has been selected during the *selection* phase, and chooses the one that has been selected the most number of times as the final social cache for that edge. More precisely, the temporary cache $TSC(e)$ for each edge $e = (u, v)$ compares the number of times it has been selected $freq(TSC(e))$ with $freq(w)$ for every $w \in CN(u, v)$. Node $n \in \{u, v\} \cup (N(u) \cap N(v))$ with the highest $freq(n)$ will be selected as a cache for that edge. Node n marks itself as *black*, marks edges in S_n as *green*, informs its friends about its selection, and terminates the algorithm.

TABLE II
MEASUREMENTS OF THE TWO ELIMINATION ALGORITHMS. $TSC(e)$ ARE DIFFERENT, BUT THE FINAL SELECTION ARE THE SAME.

Edge	e1	e2	e3	e4	e5	e6	e7
$CN(e)$	{3}	{2}	{1}	{5}	{4}	{3}	{}
Triad Elimination Method							
$TSC(e)$	1	3	3	3	3	4	4
Cache selected	3	3	3	3	3	3	4
Span Elimination Method							
$TSC(e)$	3	3	3	3	3	3	4
Cache selected	3	3	3	3	3	3	4

Given the graph shown in Figure 3, the number of transitive triads for each node is listed in Table I, and the selected $TSC(e)$ and the common neighbors $CN(e)$ are listed in Table II. Since $T(1) = T(2)$, we select node 1 (randomly) as TSC for edge $e1$. A similar situation happens for edge $e6$, where node 4 is selected as temporary social cache. Further, node 3 is selected as a TSC for edges $e2, e3, e4$, and $e5$, since $T(3) > T(1), T(2), T(4), T(5)$. Finally, $T(4) = 1 > T(6) = 0$, and node 4 is selected for edge $e7$. During the elimination phase, node 3 is selected for edges $\{e1, e2, e3, e4, e5, e6\}$, and node 4 is selected as for edge $e7$ based on frequency. The results are listed in Table II.

C. Span Elimination Algorithm

Similar to the Triad Elimination algorithm, initially, each node v calculates S_v, the set of edges that it covers. During the *selection* phase, nodes u and v of each edge $e = (u, v)$ exchange $size(S_u)$ and $size(S_v)$, and select the node with the higher value to be a temporary cache. The selected node further contacts the common neighbors of edge e, compares $size(S_n)$ with every node $n \in \{u, v\} \cup (N(u) \cap N(v))$, and selects the node w that has the largest $size(S_w)$ as $TSC(e)$.

The *elimination* phase is similar to the one in Triad Elimination algorithm. The $TSC(e)$ for each edge e contacts every node $w \in CN(u, v)$, compares the $freq(TSC(e))$ with $freq(w)$, and selects node $n \in \{u, v\} \cup (N(u) \cap N(v))$ that has the highest $freq(n)$. Node n marks itself as *black*, marks edges in S_n as *green*, informs its neighbors, and terminates the algorithm.

For the graph shown in Figure 3, Table I lists $size(S_v)$ for each node v. Nodes 1 and 2 cover 3 edges each, and their common neighbor is $CN(1, 2) = 3$. Since S_3 = 6 edges, node 3 is selected as $TSC(e1)$, as well as for edges $e2, e3, e4$, and $e5$. Since $S_4 = 4 > S_6 = 1$, therefore node 4 is selected for edge $e7$. During the elimination phase, node 3 is selected as social cache for edges $\{e1, e2, e3, e4, e5, e6\}$ since it has the highest selection frequency. Node 4 remains to be the cache for edge $e7$. The results are shown in Table II. Note that the $TSC(e)$ selected by *Span Elimination* algorithm are the same as the final caches, which indicates that Phase 1 alone is efficient in cache selection.

D. Social Score Algorithm

The *Social Score* algorithm elects social caches based on a node's *Social Score Probability*, $ss\_prob(v)$, which is calculated according to the Equation 1. The $ss(v)$ in the formula

Algorithm 1 Social Score Algorithm - Stage 1

```
calculate ss_prob(v)
if ss_prob(v) > ρ then
    mark itself as black (*social cache*)
    mark edges in S_v as green
    inform every node in N(v)
end if
```

Algorithm 2 Social Score Algorithm - Stage 2

```
while span(v) > 0 do
    calculate ratio(v)
    if (ratio(v) > γ) then
        mark v as black (*social cache*)
        make red edges in S_v as green
    else
        γ− = RATIO_STEPSIZE
        recalculate span(v)
    end if
end while
```

is the *Social Score* [8] of node v to measure the centrality of a node in its local network.

$$ss\_prob(v) = \begin{cases} 1 - 1/ss(v) & \text{if ss(v)} \geq 1 \\ 0 & \text{if ss(v)} < 1 \end{cases} \quad (1)$$

Social Score of a node is a combination of Clustering Coefficient $cc(v)$ [15], Egocentric Betweenness Centrality $ebc(v)$ [16], as well as the vertex degree and is defined by Equation 2.

$$ss(v) = [(1 - cc(v)) + ebc(v)] * deg(v) \quad (2)$$

Clustering coefficient quantifies how well connected are the neighbors of a vertex in a graph, and is defined as in Equation 3:

$$C_i = \frac{2T(i)}{deg(i)(deg(i) - 1)} \quad (3)$$

where $T(i)$ is the number of transitive triads node i is part of. An egocentric network is a "local" network consisting of a node and its immediate neighbors. Betweenness centrality measures the influence a node has over the spread of information through the network, and is defined by Equation 4:

$$BC(i) = \sum_{s \neq t \neq i} \frac{\sigma_{st}(i)}{\sigma_{st}} \quad (4)$$

where σ_{st} is the total number of shortest paths from node s to t, and $\sigma_{st}(i)$ is the number of those paths that pass through i. Egocentric Betweenness Centrality is the betweenness centrality of a vertex in its egocentric network. Given the assumption that each node v knows its two hop neighbors, these measurements can be locally calculated. Therefore, $ss(v)$ and $ss\_prob(v)$ can be calculated using local information only.

The *Social Score* algorithm executes in two stages by utilizing two predefined variables, the *threshold probability*, as ρ, and *ratio threshold*, as γ. First, each node v calculates its $ss(v)$ and $ss\_prob(v)$, and elects itself as a social cache if $ss\_prob(v) > \rho$, marks itself as *black*, marks edges in S_v as *green*, and informs its neighbors as presented in Algorithm 1. Next, each node that has not been elected executes the following steps in iterations locally. In each loop, node v recalculates its $span(v)$ and color. If every edge in S_v is *green* and thus $span(v)$ = 0, node v marks itself as *grey* and exits the loop. Any node v with $span(v) > 0$ calculates a *ratio*, $ratio(v)$, which is defined in Equation 5. If $ratio(v) > \gamma$, node v elects itself as social cache, marks itself as *black*, marks *red* edges in S_v to *green*, and notifies its neighbors as shown in Algorithm 2.

$$ratio(v) = span(v)/size(S_v) \quad (5)$$

In the second stage, $span(v)$ either decreases or remains the same after each iteration. If node v is elected, $span(v)$ becomes zero. If an edge in S_v is marked as *green* by another cache, $span(v)$ decreases. Otherwise, $span(v)$ remains the same. $ratio(v)$ changes with $span(v)$ according to Equation 5, and eventually the algorithm stops once $ratio(v) \leq \gamma$ for node v. Therefore, γ needs to be decreased by stepsize $RATIO\_STEPSIZE$ after each iteration to cover the entire graph, which is shown in Algorithm 2.

For the graph shown in Figure 3, the social scores and the corresponding probabilities are listed in Table I. If we set the ρ to be 0.75, nodes 3 and 4 will be elected as social caches during the first stage. In the second stage, nodes 1, 2, 5, 6 re-calculate their $span$, which are now equal to zero. They mark themselves as *grey* and the algorithm terminates.

E. Time Complexity

The time complexity of the algorithms is measured by the total number of communication steps. Both Triad Elimination and Span Elimination algorithms terminate in two rounds. In the first round (the *selection* phase), each pair of connected nodes u, v exchange $T(u)$, $T(v)$ or $size(S_u)$, $size(S_v)$ values respectively, to determine the temporary social caches. Note that even for Span Elimination algorithm, since every node u exchanges $size(S_u)$ with every neighbor, the information is sufficient to select a temporary cache. In the second round, each temporary social cache exchanges how many times it has been selected with all common neighbors of u and v to make a final decision. The Social Score algorithm terminates in: $1 + 1/RATIO\_STEPSIZE$ rounds. In the first round, each node decides if it has been elected as a social cache by comparing its social score with ρ, and informs its friends about the election. Next, any node that has not been elected executes Algorithm 2, where γ is first set to 1, and decreases by $RATIO\_STEPSIZE$ in each iteration until the algorithm terminates. The time complexity in terms of rounds and communication complexity (messages) for each algorithm are listed in Table III.

VI. Graph and Social Properties

In this section, we will present graph and social characteristics of the datasets that we use for evaluating the

TABLE III
TIME COMPLEXITY IN ROUNDS AND COMMUNICATION COMPLEXITY IN MESSAGES FOR EACH METHODS.

Algorithm	Time Complexity	Communication Complexity
Triads	2	$4\|E\|$
Span	2	$4\|E\|$
Social Score	$1+\frac{1}{RATIO\_STEPSIZE}$	$\|E\|+\frac{\|E\|}{RATIO\_STEPSIZE}$

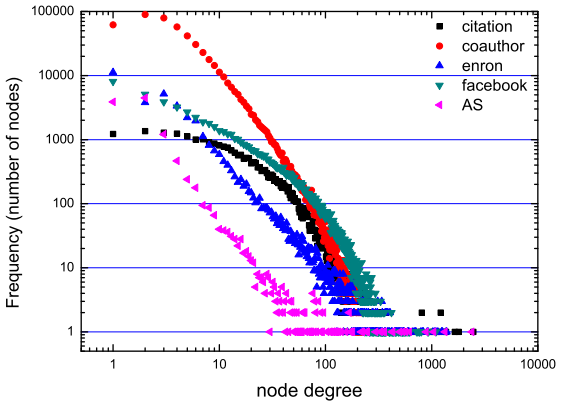


Fig. 4. The log-log plot of node degree distributions of the five graphs. The x axis represents the degree, and y axis represents the frequency.

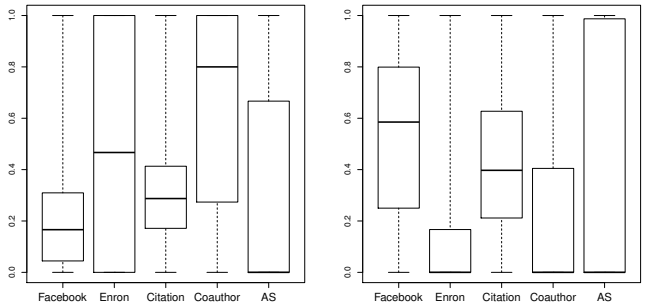
cache selection methods. Furthermore, since cache selection algorithms utilize diverse social properties, we will discuss them for each graph.

A. Dataset Description

To evaluate the proposed algorithms, we choose five widely used graphs, namely, Facebook graph [17], Enron email graph [18], Coauthor graph [19], Citation graph [20], and Autonomous Systems networks graph [21]. These graphs are considered as un-directed graphs and fit into three categories: *Social Graph, Semi-Social Graph*, and *Non-Social Graph*.

Facebook, as one of the most popular OSNs, is a typical *Social Graph*. Enron graph represents social connections but only when one sends an email to another during the data collection period. Facebook graph represents *cumulative* social connections from the day user registers with Facebook until the data is collected; while Enron graph only illustrates *periodical* social connections during the crawling duration for the dataset. Therefore, we consider the Enron graph as a *Semi-Social* graph. Coauthor and Citation graphs are also *Semi-Social Graphs*: the Coauthor graph shows how authors collaborate to produce papers, while Citation graph shows how papers cite each other. The Autonomous Systems (AS) graph shows how routers comprising the Internet are organized, and forms a *Non-Social Graph*. The statistics about vertices, edges, and node degrees are listed in Table IV.

Figure 4 is a log-log scale plot of node degree distributions for the five datasets. The x axis represents node degree, and the y axis is the number of nodes having that degree. The Coauthor, Enron and AS graphs exhibit characteristics of a power law distribution.



(a) Clustering Coefficient. (b) Egocentric Betweenness Centrality.

Fig. 5. Boxplot of Clustering Coefficient and Egocentric Betweenness Centrality for the five graphs.

B. Social Properties

Table IV lists percentage of nodes that are part of at least one transitive triad in each graph. This measurement affects the performance of the *Triad Elimination* method. Coauthor graph has the highest value, and we believe the reasons are posited as being twofold. First, research papers are likely to be composed by authors from the same lab/institute, and the same group of authors tend to collaborate to produce papers. Second, Coauthor graph is crawled from the DBLP conferences, and authors from the area tend to submit papers to these conferences over the years. Facebook and Enron graphs also have high percentage of nodes involved in a transitive triad. This is because of the social network principle: "your friend's friends are more likely to be your friends". Citation and Oregon graphs have less percentage of nodes involved in a transitive triad compared with the other three graphs.

Table IV also lists statistics about number of edges covered by a node, $size(S_v)$. $size(S_v)$ affects two cache selection algorithms: the Randomized algorithm and the Span Elimination algorithm. For any node v, $size(S_v) \leq deg(v)*(deg(v)+1)/2$. Therefore, no surprise that on average $size(S_v)$ correlates with $deg(v)$. Citation graph has the highest value for the $max(size(S_v))$ as well as $max(deg(v))$.

The Social Score algorithm utilizes Clustering Coefficient, Egocentric Betweenness Centrality and degree of a vertex as input parameters. The boxplot of the Clustering Coefficient for each graph is shown as in Figure 5(a). With lower quantile and the median being equal to the minimum, the plot for AS graph shows that at least half of the users have zero clustering coefficient, and neighbors of a node tend to be poorly connected. The plot for the Coauthor graph has upper quantile same as the maximum and median around 0.8. We believe that this is due to coauthors constructing highly connected local communities for collaboration. The boxplot of Enron graph is evenly distributed among [0,1]. The boxplots for Facebook and Citation graphs show similar layouts.

The boxplot of Egocentric Betweenness Centrality is in Figure 5(b). The plots for Facebook and Citation graphs are again similar, with Facebook graph having a higher median. The plot illustrates that on an average a vertex in Citation graph connects more non-connected vertices than a vertex in

TABLE IV
STATISTICS AND PROPERTIES OF THE FIVE GRAPHS.

Metrics		**Graph**				
		Facebook	*Enron*	*Citation*	*Coauthor*	*AS*
Number of Edges		817090	183831	705084	3742140	23409
Number of Vertex		63731	36692	27400	511164	11174
Degree	max	1098	1383	2468	597	2389
	min	1	1	1	1	
	avg	25.64	10.02	25.70	7.32	4.19
	median	10	3	15	4	2
% of nodes in transitive triads		77%	67%	40%	87%	41%
Number of edges covered, $size(S_v)$	max	20189	18770	35995	9661	6027
	min	1	1	1	1	1
	avg	190.47	69.46	187.60	30.16	9.53

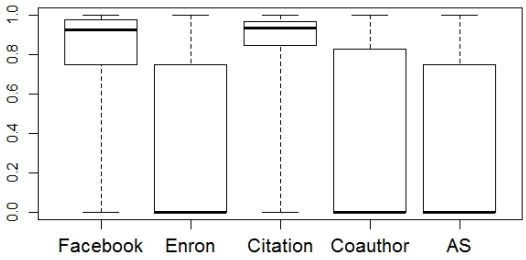


Fig. 7. Boxplot of the social score probability for each graph.

Facebook graph. The median and lower quantile are equal to the minimum value in the Enron, Coauthor and the AS graphs. This demonstrates that at least half of the vertices are not centered in their egocentric graphs, hence, are unlikely to be selected as caches. The evaluation results in Section VII-C verify this by showing that the number of social caches selected is less than half of vertices no matter which method is used.

We also calculate the *social score*, and the *social score probability* for each node in each graph. The social score distribution plotted in a 3D coordinate system formed by degree, clustering coefficient, and egocentric betweenness centrality for each graph is presented in Figure 6, with x axis as degree, y axis as clustering coefficient, and z axis as egocentric betweenness centrality. The dots are the social scores, which is calculated based purely on local information.

Figure 7 shows boxplots of the social score probability for each graph. Enron, Coauthor, and AS graphs have the medians and lower quantiles equal to the minimum (0). For Facebook and Citation graphs, the boxplots have the medians greater than 0.9 and lower quantiles larger than 0.7. This means that the percentage of vertices selected as social caches in Facebook and Citation graphs is higher than the other three graphs.

Facebook and Citation graphs have similar graph and social properties in terms of average node degree, percentage of nodes in transitive triads, average number of edges covered ($size(S_v)$), clustering coefficient and egocentric betweenness centrality distributions, as well as social score probability. Since we utilized these social properties in various cache selection methods, we believe that this similarity will translate into similar performance of the corresponding methods.

VII. EVALUATION

We evaluate four distributed social cache selection algorithms using the five graphs that include both "Social Graph" to "Non-Social Graph" as discussed in Section VI in order to answer the following questions:

- Which algorithm performs best in terms of number of caches selected?
- Do the discussed social properties affect the algorithm performance, and how?
- Do graph categories, e.g. social graph or non-social graph, affect the selection of caches?

We will first present some results regarding Randomized algorithm and Social Score algorithm, and then compare all the four algorithms.

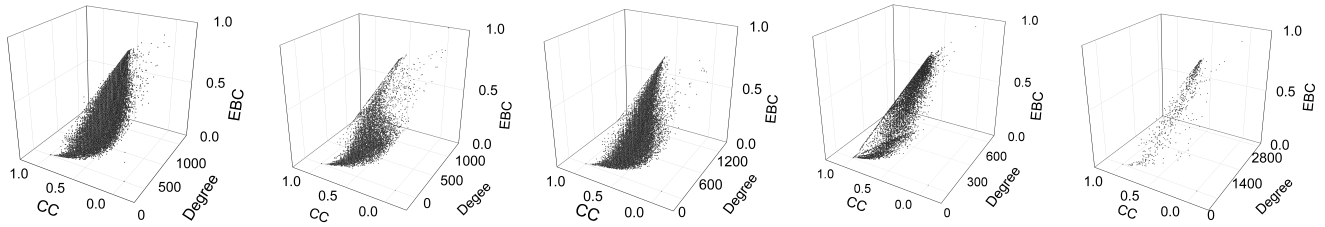
A. Randomized Algorithm

To evaluate the Randomized algorithm, we vary the probability threshold θ, and run the algorithm 10 times for any given θ on each graph. Figure 8 presents fraction of nodes elected as social caches with error bars (y axis) when varying θ (x axis) for the five graphs. [1] The minimum value of θ we test is 0.90, since the fraction of nodes elected shows clear pattern of stability for every graph. Specifically, as θ decreases, fraction of elected nodes remains almost the same for Facebook and Citation graphs, but increase slightly for Enron and Coauthor graphs. As for AS graph, the results zigzag as θ decreases, and we believe it is due to the following two reasons: *(i).* the Randomized method is simply based on the randomly generated numbers, which makes it unpredictable; *(ii).* the topology of AS graph differs from the other four graphs as it is a "Non-Social" graph.

B. Social Score Algorithm

The performance of the Social Score algorithm is determined by two key parameters: the social score probability threshold ρ and the ratio threshold γ. Therefore, we perform a set of experiments to answer the following questions: *(i)* What is the fraction of nodes elected and fraction of edges marked as *green* after stage 1 (Algorithm 1) when varying the ρ? *(ii)*

[1] In this section, we will compare the performance of different algorithms given the five graphs. Since each graph has different $|E|$ and $|V|$, we use *fraction* of nodes (edges) for comparison purpose.



(a) Facebook (b) Enron (c) Citation (d) Coauthor (e) AS

Fig. 6. Plots of social score of the five datasets in the 3D coordinate system composed of deg, cc, and ebc as x, y, and z axis.

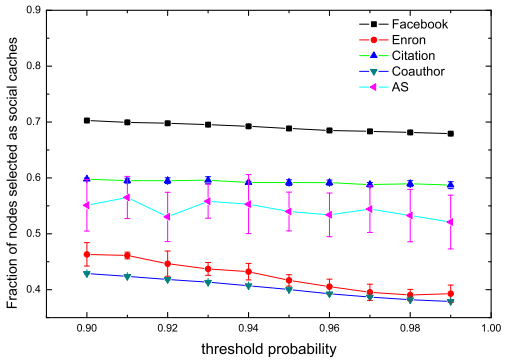


Fig. 8. Fraction of nodes elected as social caches with error bars (y axis) by randomized algorithm when varying the θ (x axis).

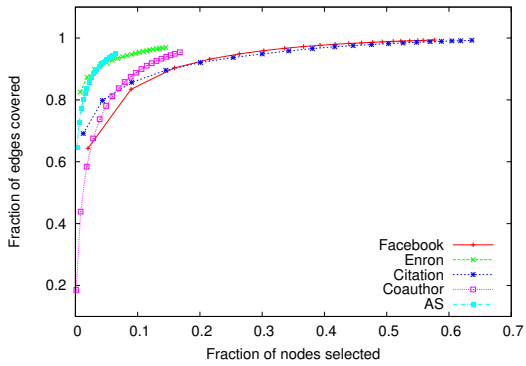


Fig. 9. Fraction of nodes elected (x axis) v.s. fraction of edges marked as green (y axix) after the first phase of the algorithm when vary the ρ. For each line, the left most symbol represents ρ = 0.995, and the following symbols represent ρ decreases by 0.05 to 0.9.

What is the number of social caches elected by the algorithm when varying both ρ and γ? *(iii)* How does the algorithm converge?

First, we observe the stage 1 of the algorithm, where nodes elect themselves based on the social score probability. Figure 9 plots the fraction of edges marked as *green* (y axis) depending on the fraction of nodes elected as social caches (x axis). For each line (that corresponds to one of the five graphs) the leftmost symbol represents ρ = 0.995, the following symbols represent ρ decreased by 0.05 and the rightmost symbol corresponds to ρ = 0.9. Thus we increase the fraction of nodes selected (x-axis) by decreasing the threshold probability. As we decrease the ρ, the fraction of social caches elected, and the fraction of edges marked as *green* are both increasing. As

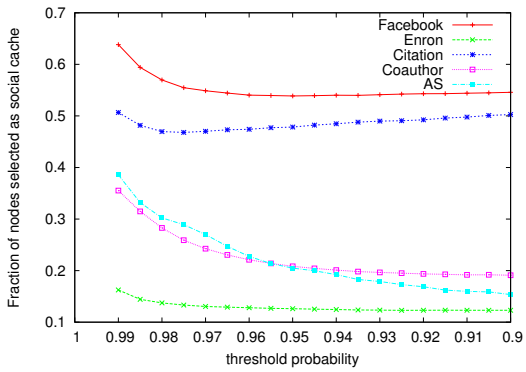


Fig. 10. Fraction of vertices elected as social cache (y axis) when varying the ρ (x axis) from 0.99 to 0.9 by step size 0.05.

we discussed in Figure 7, more than 50% of vertices in both Facebook and Citation graphs have social score probability greater than 0.9. Therefore, above 60% of nodes are elected as social caches for these two graphs when ρ reaches 0.9. For the other three graphs, only less than 20% nodes are elected after the first phase of the algorithm. For each graph, the fraction of edges that are covered after the first phase approaches 100%.

Second, we study number of caches elected when the algorithm finishes. The results are shown in Figure 10. The x axis represents ρ ranging from 0.99 to 0.9, and the y axis is fraction of nodes elected as social caches. Decreasing ρ enables stage 1 of the algorithm to elect more social caches, hence, covers larger portion of the edges in the graph. For Facebook and Citation graphs, fraction of nodes elected as social caches reaches a minimum (optimal value) at ρ = 0.97. Reaching an optimal value is observed for the Enron graph as well. This can be explained by the fact that for the particular probability threshold optimal number, more edges become green during the first phase already. For the other two graphs, Coauthor and AS, the number of caches decreases when probability decreases.

Finally, we discuss how the algorithm converges when varying γ for different ρ. We use fraction of *green* edges in the graph as measurement to evaluate the convergence rate. Figure 11 shows convergence of the algorithm via fraction of *green* edges (y axis), and the x axis represents γ. The lines with different colors in each subgraph, from bottom to top, show γ varying from 0.9 to 0.99 with stepsize equals to 0.05. The convergence rates for all graphs are similar. In the AS graph, there is a spike when the γ decreases from 0.6 to 0.4. We

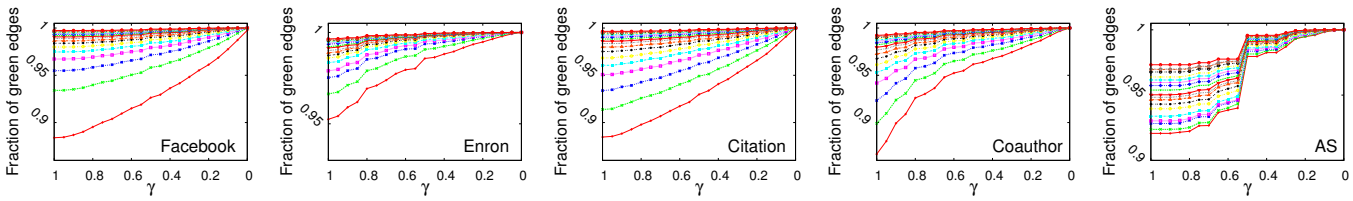


Fig. 11. Convergence of the Social Score algorithm in the means of fraction of green edges when γ decreases(x axis). The different lines in each subgraph, from bottom to top, shows ρ increasing from 0.9 to 0.99.

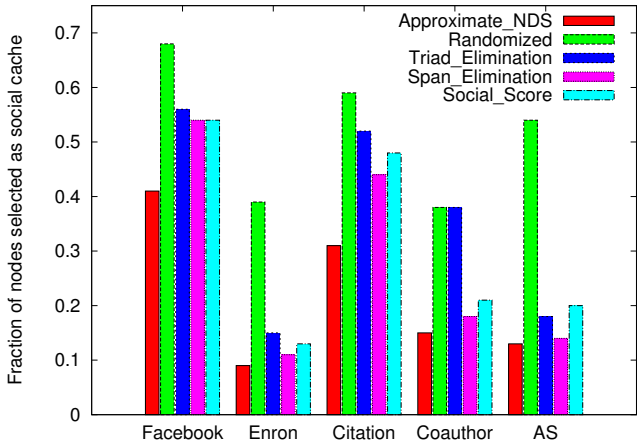


Fig. 12. Fraction of nodes selected as social caches for each graph.

believe that this is because the number of nodes with degree equal to 1 is non-negligible; these nodes will be elected as social cache only when the ratio threshold decreases to 0.5. Indeed, a large portion, 3866 out 11174 vertices in the AS graph, have degree equal to 1.

C. Comparison of Algorithms

We evaluate the performance of the four proposed algorithms by comparing the fraction of nodes selected as social caches. Specifically, we compare them with the *Approximate NDS* algorithm introduced in [8], which is a centralized cache selection method that has an $O(\log m)$ approximation to the cache selection problem (Neighbor-Dominating Set), where m is number of edges in the graph.

Table V lists the fraction of nodes selected as social caches when different methods are being applied on each graph (number of caches selected are also listed in parentheses). Specifically, we choose $\theta = 0.99$ for the Randomized method, and choose $\rho = 0.95$ for the Social Score method for the comparison. The results for Triads and Span elimination methods are averages of ten runs to reduce possible bias. The standard deviation is also listed. The fraction of nodes selected as social caches for each graph is also plotted in Figure 12. From Table V and Figure 12, we observe that Facebook and Citation graphs select 20% more than the other three graphs no matter which cache selection method is used, which confirms our initial guess in Section VI-B. We believe that the social properties and measurements discussed in Section VI-B are the key influencers for social caches selection and placement.

Figure 12 shows that all the three algorithms Span Elimination, Triad Elimination, and Social Score have similar good performance, yet about 15% or lower than the centralized best known approximation algorithm, Approximate NDS. Span Elimination algorithm performs best in all cases, although occasionally (Facebook data), the performance of Social Score is comparable. We believe this is because Social Score algorithm is specifically designed for *Social Graphs*, such as Facebook. Span elimination algorithm outperforms the Triad elimination, mainly because it takes into account the span of a node besides the number of triads around a node.

VIII. Cache Maintenance

As OSN friends join and leave over time, the underlying social graph changes as well, requiring a re-election of social caches. In this section, we discuss the *Least Cache Re-election* (LCR) mechanism, which maintains the proper set of caches while reducing the re-election overhead as much as possible. The mechanism is inspired by the Least Cluster Change approach [22] proposed for Mobile Ad-hoc networks.

When a node enters or leaves a social network or when new connections are created or existing ones removed, a procedure of *friending* or *de-friending* takes place. With respect to friending, three scenarios are of interest for our method: a cache node friends with a non-cache, a non-cache friends with a non-cache, and a cache friends with a cache. In order to reduce the maintenance overhead, the LCR does not perform re-elections when a non-cache node friends with a cache, or when a cache friends with a cache. Although these actions may result in redundant caches, the existing set of caches will still be a Neighbor-Dominating set.

When a non-cache node friends with another non-cache node, a cache re-election occurs to ensure that a cache node is available for this newly formed connection. In this case, the two non-cache nodes u and v will exchange their friend lists and calculate their common neighbor set $CN(u, v)$. If there exists at least one cache in $CN(u, v)$, no re-election is needed. Otherwise, a new cache node needs to be selected between u and v according to the cache selection algorithm used.

Similarly, we consider the following scenarios when de-friending occurs: a cache node de-friends with a cache, a cache de-friends with a non-cache, and a non-cache de-friends with a non-cache. In the situation where a cache de-friends with a cache, and a non-cache de-friends with a non-cache, no re-election is needed. In the scenario where a cache de-friends with a non-cache, the re-election is needed to ensure that every

TABLE V
FRACTION OF NODES SELECTED AS SOCIAL CACHE BY DIFFERENT ALGORITHMS (NUMBER OF CACHES SELECTED ARE LISTED IN PARENTHESES)

Algorithms	**Graph**				
	Facebook	*Enron*	*Citation*	*Coauthor*	*AS*
Number of Vertex	63731	36692	27400	511164	11174
Centralized Apprx NDS	0.41(26288)	0.09(3370)	0.31(8517)	0.15(79672)	0.13(1505)
Randomized Algorithm (p = 0.99)	0.68(43291)	0.39(14416)	0.59(16061)	0.38(193399)	0.54(6079)
Triads Elimination	0.56(35913.6)/(29.42)	0.15(5410.7)/(36.19)	0.52(14355)/(33.56)	0.38(194527.9)/(32.85)	0.18(1998.3)/(12.88)
Span Elimination	0.54(34096.7)/(6.88)	0.11(3929.1)/(7.62)	0.44(12116.0)/(6.55)	0.18(92089.3)/(21.87)	0.14(1585.1)/(3.31)
Social Score ($p = 0.95$)	0.54(34331)	0.13(4627)	0.48(13107)	0.21(106530)	0.21(2290)

friend of the non-cache can get its social updates. We adopt a simple approach by letting the non-cache node notifies its neighbors about the de-friending to initiate a cache re-election. Note that, in case of a cache miss, a node can always go to the original node to get the latest social updates. We believe the "Least Cache Re-election" mechanism is a good trade-off between cache availability and maintenance overhead.

IX. DISCUSSION

In this section, we discuss several aspects that we did not cover in this paper but plan to address as future work.

Network Dynamics and Availability: The performance of SocialCDN is directly influenced by the network dynamics and the content availability. In Section VIII, we described possible directions of how to handle nodes' joins and leaves as well as unexpected failure situations. Detailed experimental evaluation with realistic nodes' reliability data will be one of our future directions. We also plan to explore availability in SocialCDN in the future.

Load Balancing: How to balance the network traffic associated with the cache nodes is another important issue. As part of future work, we plan to formulate a new optimization problem by adding an additional constraint related to the load in terms of number of connections or traffic per cache node.

Privacy: SocialCDN assumes that users know their immediate friends and friends-of-friends. We did not investigate nodes' misbehaving scenarios and Sybil attack in this paper, which are the other directions for our future work.

X. CONCLUSION

In this paper, we presented SocialCDN, a novel social content dissemination system for Distributed Online Social Networks based on Social Caches. By caching the social updates on social caches, SocialCDN enables efficient data dissemination among social buddies through fewer network connections. We propose four distributed cache selection algorithms for SocialCDN based on different social properties, Randomized, Triad Elimination, Span Elimination, and Social Score algorithm. Empirical evaluations on five well known graphs show that Span Elimination algorithm has the least time complexity in term of communication steps, and selects least number of social caches for any given graph.

XI. ACKNOWLEDGEMENT

We thank our shepherd Venugopalan Ramasubramanian for his valuable suggestions, and the anonymous reviewers for their insightful comments. This work is supported in part by the NSF grant CNS-1111811.

REFERENCES

[1] "Diaspora." [Online]. Available: http://joindiaspora.com
[2] S. Buchegger, D. Schiöberg, L. H. Vu, and A. Datta, "Peerson: P2p social networking - early experiences and insights," in *Proc. of the Second ACM Workshop on Social Network Systems (SNS'09)*, 2009.
[3] S.-W. Seong, J. Seo, M. Nasielski, D. Sengupta, S. Hangal, S. K. Teh, R. Chu, B. Dodson, and M. S. Lam, "Prpl: a decentralized social networking infrastructure," in *Proc. Workshop on Mobile Cloud Computing Services: Social Networks and Beyond*, 2010.
[4] "Distributed online social networks list." [Online]. Available: http://en.wikipedia.org/wiki/Distributed_social_network
[5] K. N. Hampton, L. S. Goulet, L. Raine, and K. Purcell, "Social networking sites and our lives," *Pew Internet and American Life Project*.
[6] "Network traffic distribution." [Online]. Available: http://www.greenhostit.com/green-blog/98-blogging/338-blogging-for-traffic
[7] J. Dilley, B. Maggs, J. Parikh, H. Prokop, and B. Weihl, "Globally distributed content delivery," *IEEE Internet Computing*, 2002.
[8] L. Han, B. Nath, L. Iftode, and S. Muthukrishnan, "Social butterfly: Social caches for distributed social networks," in *SocialCom*, 2011.
[9] O. Schneider, "Trust-aware social networking: A distributed storage system based on social trust and geographic proximity," 2009.
[10] A. Olteanu and P. Guillaume, "Towards Robust and Scalable Peer-to-Peer social networks," in *SNS'12*.
[11] R. Narendula, T. G. Papaioannou, and K. Aberer, "My3: A highly-available P2P-based online social network," in *Proc. of the 11th IEEE International Conf. on Peer-to-Peer Computing (IEEE P2P'11)*, 2011.
[12] G. Mega, A. Montresor, and G. P. Picco, "Efficient dissemination in decentralized social networks," in *Proc. of Conf. on Peer-to-Peer Computing (P2P'11)*, Aug. 2011.
[13] A. Vahdat and D. Becker, "Epidemic routing for partially connected ad hoc networks," *Technical Report CS-200006, Duke University.*, 2000.
[14] A. Lindgren, A. Doria, and O. Schelén, "Probabilistic routing in intermittently connected networks," *SIGMOBILE Mob. Comput. Commun. Rev.*, vol. 7, no. 3, pp. 19–20, Jul. 2003.
[15] P. W. Holland and S. Leinhardt, "Transitivity in structural models of small groups," *Small Group Research*, vol. 2, pp. 107–124, 1971.
[16] P. Marsden, "Egocentric and sociocentric measures of network centrality," *Social Networks*, vol. 24, no. 4, pp. 407–422, 2002.
[17] B. Viswanath, A. Mislove, M. Cha, and K. P. Gummadi, "On the evolution of user interaction in facebook," in *Proc. of Workshop on Online social networks (WOSN'09)*, 2009.
[18] B. Klimt and Y. Yang, "Introducing the enron corpus." in *In First Conference on Email and Anti-Spam (CEAS'04)*, 2004.
[19] "The dblp computer science bibliography coauthor graph." [Online]. Available: http://www.sommer.jp/graphs
[20] "The kdd competition, citation graph." [Online]. Available: http://www.sommer.jp/graphs
[21] J. Leskovec, J. Kleinberg, and C. Faloutsos, "Graphs over time: densification laws, shrinking diameters and possible explanations," in *Proc. of Conf. on Knowledge discovery in data mining (KDD'05)*, 2005.
[22] C. chuan Chiang and M. Gerla, "Routing and multicast in multihop, mobile wireless networks," in *Proc. in Multihop, Mobile Wireless Networks (ICUPC '97)*, 1997.