# Balanced Content Replication in Peer-to-Peer Online Social Networks

Mohammad A Khan
Department of Computer Science
New Jersey Institute of Technology
Email: mak43@njit.edu

Hillol Debnath
Department of Computer Science
New Jersey Institute of Technology
Email: hd43@njit.edu

Cristian Borcea
Department of Computer Science
New Jersey Institute of Technology
Email: borcea@njit.edu

***Abstract*—This paper presents an effective content replication scheme for peer-to-peer online social networks (P2P-OSN). The topology of P2P-OSN is defined by the social network of the participants. P2P-OSN allow people to share content and run applications with their 1-hop friends in decentralized fashion, while denying access to their potentially private data to other users. Content replication in these networks is difficult because users can place replicas only at their 1-hop friends, and this could substantially skew the storage availability in the network.**

**Our main contribution is a distributed replication method that prevents the skewness of available replication storage across the network and improves replication fairness/success without relying on global knowledge of the social network. We developed a new centrality metric, EasyRank, which is calculated at each peer and finds the underlying connectivity structure responsible for introducing the skewness of storage availability. Our distributed replica placement algorithm places the replicas after ranking the potential storage peers using their EasyRank scores and currently available storage. We evaluated our solution with social graphs from Facebook and Google+ having more than 4,900 vertices and 720K edges. The evaluation is done for both stable and emerging social networks. The results show that EasyRank-based replication achieves the fairest storage allocation and maintains the most balanced storage availability among the tested methods. Thus, it provides the highest replication success rate.**

***Index Terms*—Replication, online social networks, peer-to-peer.**

## I. Introduction

Peer-to-peer online social networks (P2P-OSN) are P2P networks used to share socially generated content and execute OSN applications over the shared content [1], [2], [3], [4], [5]. The topology of these networks is defined by the social network graph of the participating users. P2P-OSN allows people to run collaborative applications with their friends in decentralized fashion and, at the same time, make unauthorized access to their potentially private information harder. If designed properly, they could enable similar OSN applications with those currently provided by centralized platforms.

This paper addresses content replication in P2P-OSN. We assume that users are willing to collaborate for content replication with their 1-hop social connections (friends) in the network, but not with any other users. Replication has three benefits in P2P-OSN: (1) higher content availability in the face of typical P2P churn, (2) higher data locality, as many times OSN applications will find the friends' data stored locally on the user's computer, (3) faster execution, as OSN applications could be sped up by running in parallel over different chunks of the replicated data.

The problem is: how to replicate content efficiently in P2P-OSN without disclosing the global network topology (i.e., social graph) to any peer? The requirement of not disclosing the social graph is crucial from a privacy point of view. Other researchers have studied the content placement problem extensively for centralized social networks [6], [7]. However, these solutions use the entire social graph for placing content and thus do not work for P2P-OSN. Furthermore, they were designed for data center environments which are very different from P2P environments regarding churn, latency, etc.

As peers use only 1-hop storage and cannot see the entire social graph, naive solutions would give more replication storage to peers with more neighbors, but their local storage would quickly be allocated. Subsequently, users with few friends would remain without options to ensure the desired replication factor as neighbors with many friends would be already full. An effective content replication in P2P-OSN should allocate storage carefully and maintain uniform replication storage availability across the network over time. In the ideal case, the ratio of available local storage to total network storage will be the same for every peer at any time. This property ensures that peers with many neighbors do not get overloaded, while peers with few neighbors can replicate content with the same success rate as peers with many neighbors.

The main contribution of this paper is a replication method that prevents skewness in the availability of replication storage across P2P-OSN without relying on global knowledge of the network topology. The distributed replica placement algorithm ranks the neighboring peers based on a centrality metric and their currently available storage. The centrality metric provides insights about the network topology and helps measure the impact of replica placement decisions. Specifically, the centrality metric assigns a score to each peer based on its structural position in the network. Similarly, a score based on the currently available storage is assigned to each peer. The replicas for each piece of content are stored at the neighboring peers according to their ranks until all the desired replicas are stored or until no more peers are available.

We define a new centrality metric for each peer, EasyRank, which is a function of the number of friends of its 1-hop neighbors. EasyRank assigns relatively higher scores to peers whose neighbors have few friends. These are the peers who

finish their storage soon and introduce skewness in storage availability. Therefore, the EasyRank-based peer ranking has the potential to reduce the skewness in storage availability and improve replication fairness/success.

We have used simulations to evaluate our solution with social graphs extracted from Facebook and Google+ [8]. We evaluated three common scenarios: (1) stable social networks and constant storage, (2) stable social networks, but users are adding storage over time, and (3) emerging social networks — new peers are joining, and new social connections are forming over time. We have compared our replication scheme with: (1) several methods using the same algorithm but other centrality metrics, (2) a method that first chooses the peers with enough available storage and then selects one of them randomly. The results show that the EasyRank-based replication algorithm maintains the most uniform/balanced replication storage availability over the peers, leading to higher replication success rates and lower failed replication rates.

The rest of the paper is structured as follows. Section II overviews the problem and the main ideas of our solution. Section III discusses the related work. The principles of centrality-based replication and the EasyRank centrality metric are described in Section IV. Section V presents the replica placement algorithm for the EasyRank-based replication. Section VI presents the evaluation and analysis of the results. The paper concludes in Section VII.

## II. Overview

Since replication requires peers to share storage with other users, we assume that users are willing to collaborate with their 1-hop social connections, but not with any other users. There are two reasons behind this assumption which limit the number of peers available for replication. First, most sharing in social networks is done with 1-hop connections, and thus, it makes sense to replicate data at these peers. Replication acts as an incentive for sharing because it improves application response time: the friends' content is already stored locally due to replication. Second, users might be unwilling to replicate some of their content on peers belonging to unknown people due to privacy. Replicating encrypted content could use all the peers in the network, but it slows down the applications and could be difficult to manage.

We assume that applications running over P2P-OSN decide their desired replication factor, and the replication factor, $r_f$ satisfies the following condition: $r_f <= neighbors\_count+1$. A peer always stores one replica of its content locally. For example, applications may set the replication factor as follows:

$$r_f = 1 - \log(1 - desired_{availability}) / \log(1 - avg_{peer\_up})$$

Here, $desired_{availability}$ is the required availability of the content (decided by the application or the user), and $avg_{peer\_up}$ is the average uptime of the 1-hop peers.

We can demonstrate why replication is hard in P2P-OSN using the social network in Figure 1. Suppose, C wants to store one replica for a piece of content. Intuitively, it should store the replica on B or F instead of D because D is the only

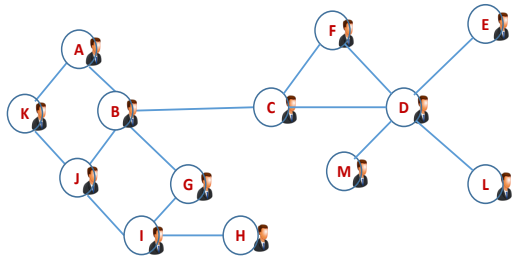



Fig. 1: A social graph and its corresponding P2P-OSN

storage option for peers M, L, E. But C does not know about M, L, E or their connectivity status because no peer can see the social graph beyond 1-hop neighbors. To solve this type of problem, an effective replication method should consider the structural properties of the peers in the network despite not having access to the global social graph.

To design such a method, we propose to compute a localized network centrality metric, EasyRank, for each peer; the scores obtained from this metric are used to rank the peers with respect to their position in the social graph. As it will be explained in section V, the ranking effectively differentiate peers who have relatively limited storage space. Our replication method also considers the current available storage at peers, which can also be a source of imbalance. Broadly, our distributed replica placement works as follows:

- Upon any local changes, each peer updates its 1-hop neighbors with its EasyRank score and available storage.
- Upon receiving updates, each peer calculates a score for each 1-hop neighbor using the reported values for EasyRank and available storage.
- Each peer ranks its neighbors according to the compound score.
- When replication is needed, replicas are placed on the neighbor peers according to their rank. Higher-ranked peers are chosen sequentially (starting from the top-ranked), with a small probability of skipping peers [9].

## III. Related Work

Social relationships have been exploited for fair storage sharing in Friendstore [10] and Social-Cloud [11]. FriendStore provides backup services among socially-acquainted groups of peers. Social-Cloud uses Facebook as a trading platform to share storage among friends. These works maintain fairness in storage sharing/usage at personal level. On the other hand, our goal is to achieve social fairness at the network level. Therefore, we have to carefully place replicas to ensure that everyone can receive storage as long as any storage is available in the network. This model is expected to enable novel OSN services, which will eventually encourage users to donate more storage. Another replica placement system for general P2P networks is Farsite [12], where peers can access any other peer in the network. Our solution, on the other hand, targets P2P-OSN where peers can access only their 1-hop neighbors.

Replica placement for centralized social networks has been extensively investigated. In [6], the goal is to minimize the number of replicas, while ensuring that all social network operations are disk-local for users. S-Clone [7] has a slightly different goal: reduce the number of replicas given a fixed amount of storage. These works use the global social graph to find suitable partitions of the social graph before placing replicas. Since P2P-OSN have no central authority which can see the entire graph, these solutions are not feasible for us.

Several projects [2], [4] investigated operations such as profile and privacy management for P2P-based social network platforms. The primary focus of these projects is to maintain the availability of social profile meta-data and lower the access latency. These solutions are suitable for handling small amounts of data, but not for large amounts of user-generated content, which is the focus of this paper.

The research in [1] used network centrality to efficiently manage the social graph using P2P technology. The social graph is partitioned into small communities, which are then assigned to the peers. This work analyzes the entire social graph before mapping it to the peers and uses centrality to efficiently store the graph. Our work uses centrality to replicate content in a balanced and privacy-aware manner.

Several works investigated timely propagation of new content or profile changes in OSN. SocialCDN [3] discusses distributed caching mechanisms to efficiently distribute updates for decentralized OSN. Mobiclique [13] and Contrail [14] optimize the social update propagation for mobile environments. These works differ from ours in their main focus.

Storing content over any trusted peer in a P2P network, not just 1-hop neighbors, has been discussed in PrPl [15], Prometheus [5], Diaspora [16], Confidant [17], and Vis-a-Vis [18]. These projects did not focus on efficient content placing and replication based on the social graph.

The use of network centrality for replica placement in P2P networks has been discussed in [19]. This work uses peer ranks to store replicas for content distribution networks (CDN) and video-on-demand (VoD). However, P2P-OSN have different topologies and storage access rules compared to the networks used for CDN and VoD. Therefore, these solutions do not work in our context.

## IV. Balanced Replication with Centrality

Our idea is to use a network centrality metric to quantify the structural properties of the peers in P2P-OSN for efficient replication in order to avoid storage allocation skweness. This skewness is introduced mainly for peers with many neighbors, when these neighbors are not well connected. These peers become replication hotspots in the network, and all their storage could be quickly allocated. For example, in Figure 1, the only replication storage available for peers E, M, L is D's storage. If F and C do not treat D differently than their other neighbors (C or B), there is a chance that E, L, M will find no replication storage available very soon.

We want to detect this type of network structure and assign centrality scores to peers based on the severity of the potential for introducing skewness. Furthermore, peers should be able to calculate the metric locally using simple functions. The calculation should not require multi-hop access to the social graph or distributed iterations, which may fail due to churn in the network.

1: ▷ Periodically exchange friends counts information with 1-hop neighbors only
2: $r_i \leftarrow 0$
3: **for** all neighbors $j$ **do**
4: $r_i \leftarrow r_i + \frac{1}{sqrt(deg_j)}$
5: **end for**
6: $r_i \leftarrow sqrt(deg_i) + r_i$
7: ▷ Normalize the metrics
8: $easyrank \leftarrow 1.0 - \frac{1.0}{sqrt(1+r_i^2)}$

Fig. 2: EasyRank algorithm

This section presents first our EasyRank centrality metric, wich satisfy these properties. Then, we explain why existing centrality metrics do not work well for balanced replication in P2P-OSN, while EasyRank does. Finally, we illustrate centrality-based balanced replication with several examples.

### A. EasyRank

Each time a peer P wants to use storage for replication, it competes with other friends of the targeted storage owner. As the social graph is not revealed to the peers, P cannot find out any information about the replication storage availability of these other friends and does not even know some of them (uncommon friends).

EasyRank calculates the centrality metrics of peers using two pieces of information: (1) the number of peers which compete for the same storage, and (2) the network structure of these peers. Then, it normalizes the metrics for a meaningful comparison of non-neighboring peers. For example, in Figure 1, when C wants to store a replica, it ranks the potential candidates B, F, and D. But B does not even know that D or F exists. Therefore, the centrality scores calculated at B must be comparable to the centrality scores calculated at D or F.

Figure 2 shows the EasyRank algorithm. To calculate EasyRank, peers share their friend count with 1-hop neighbors. Peers never share any other information about the social connections. Afterward, peers exchange the EasyRank values with their 1-hop neighbors only.

Lines 2-5 calculate the inverse of square root of the number of friends of neighbors. These neighbors compete for the same replication storage. We take the square root to minimize the impact of large numbers on the metric. We take the inverse to assign larger values to peers having relatively less-connected friends. Line 6 adds the square root of the degree of the peer. This value is not inverted as it directly indicates the number of competitors. Finally, in line 8, we normalize the metrics within (0,1) range.

### B. Why Existing Centrality Metrics Are Not Sufficient?

**Degree centrality** is the node degree of a peer, i.e., the number of friends that the owner of the peer has. The number of friends captures to some extent the immediate neighbor connections, and this metric can be computed easily. However, it is inadequate for discovering social structure beyond the

friend count, such as how the friends are connected with others in the network. For example, even though B and D have the same degree centrality in Figure 1, D's friends only option to get storage for replication is D, while B's friends have multiple options. This suggests that B and D should be treated differently.

**Eigenvector centrality** assigns higher scores to peers having connections with more central peers. The eigenvector centrality for peer $i$, $c_i$, is calculated in the following way:

$$c_i = \frac{1}{\lambda} \sum_j A_{ij} c_j$$

In the equation, $A$ is the adjacency matrix, $\lambda$ is the highest eigenvalue, and $c_j$ is the eigenvector centrality of peer $j$. The eigenvector centrality of a peer is proportional to the eigenvector centrality of the neighbors. Therefore, the eigenvector centrality captures information of the network structure which spans beyond 1-hop. For example, in Figure 1, unlike degree centrality, eigenvector centrality will treat B and D differently (Table I shows the exact values).

A significant problem with eigenvector centrality is that its calculation needs multiple iterations to converge to reasonably accurate values. This calculation requires exchanging intermediate values around the network, which leads to high latency. In addition, churn in P2P networks may prevent reasonable convergence and stability of the calculated values. Furthermore, new peer joins or edge creations will force recalculation of the old values.

**Pagerank** is a variation of eigenvector centrality. While eigenvector centrality assigns higher scores to all the neighbors of a high centrality peer, pagerank adjusts the scores according to the number of neighbors. The pagerank for peer $i$, $c_i$, is calculated using the following equation:

$$c_i = (1 - d) + d * \sum_j \frac{c_j}{deg_j}$$

Here, $d$ is a damping factor, $deg_j$ is the degree centrality of peer $j$, and $c_j$ is the pagerank of peer $j$. For our problem, pagerank captures better the contribution of the higher centrality peers over lower centrality peers, as the contribution of each peer is scaled down with the number of neighbors. Similar to eigenvector centrality, pagerank captures information around a peer with respect to the whole network. It also requires multiple iterations to calculate the correct value and suffers from the same problems with eigenvector centrality.

**Bonacich's power centrality** assigns higher scores to the peers that are connected with powerless peers. According to this metric, if a peer has many friends which are not well connected, the peer is more central. This is one of the expected feature for replication in P2P-OSN because it captures the dependency among peers. It can be calculated using following formula:

$$c_i = \sum_j (\alpha + \beta c_j) * A_{ij}$$

Here, $\alpha$ is a normalizing constant, $\beta$ indicates the importance of the neighbor's centrality, $A$ is the adjacency matrix, $c_j$ is the Bonacich centrality of peer $j$. The calculation of this metric also requires multiple iterations. Thus, it suffers from the same problems with eigenvector centrality and pagerank.

Therefore, these centrality metrics capture information of the network structure beyond 1-hop. However, distributed algorithms to calculate all these metrics (except degree centrality) require precise coordination among the peers over a lengthy series of iterative steps. The intermediate exchanged values have potential to reveal critical social information. In addition, churn in P2P networks may prevent reasonable convergence and stability of the calculated values. Furthermore, joining of new peers or edge creations will force recalculation of the old values.

EasyRank has the following benefits over these metrics: (i) lightweight computation, as there is no need for distributed iterations in the algorithm; thus, the metric can be calculated in constant time, and churn has no effect on the calculation; (ii) it captures the structural information needed for balanced replication. Peers whose neighbors do not have many neighbors receive higher scores; (iii) newly joined peers can compute a score immediately; and (iv) new peers or connections only force the recalculation at the affected peers. Therefore, EasyRank is expected to work well within the constraints of P2P-OSN.

### *C. Examples of Replication Using Network Centrality*

To demonstrate the usage of network centrality for replication storage allocation in P2P-OSN, we present a few examples. These examples are based on the scenario from Figure 1. We analyze the impact of different centrality metrics on storage allocation fairness, replica placement, and replication success rate.

*1) Fairness in Storage Allocation:* One important requirement for storage allocation in P2P-OSN is fairness: peers should receive a similar amount of storage independent of how many neighbors/friends they have. This minimizes the skew in available storage in the long run and leads to better global space utilization. Table I shows the storage allocation for the peers in Figure 1 using allocation policies based on the centrality metrics previously described. We assume that each peer initially donated 1GB of storage. Each peer divides the storage among its neighbors/friends proportional with the centrality metric of the respective column. Each value in the table indicates how much storage each peer gets if all the peers apply the policy specified by the column. The bottom three rows of Table I show the 25-th and 75-th percentiles, and, the standard deviation of the storage acquired by the peers.

We see that EasyRank is the most successful in fairly distributing the storage space irrespective of the number of neighbors the peers have. The highlighted lines in Table I show the situation of the peers having few neighbors (L and M) and many neighbors (B and D). The results demonstrate that EasyRank provides the highest amount of storage to the peers with few neighbors (M and L). It provides a lower amount

TABLE I: Storage space allocated to peers when everyone donates 1GB

| Peer | Degree | EasyRank | Bonacich | PageRank |
|---|---|---|---|---|
| A | 0.35 | 0.47 | 0.02 | 0.37 |
| B | 2.61 | 2.11 | 2.49 | 2.56 |
| C | 0.99 | 0.94 | 1.35 | 0.91 |
| D | 4.04 | 3.91 | 3.70 | 4.10 |
| E | 0.13 | 0.17 | 0.10 | 0.14 |
| F | 0.41 | 0.50 | 0.72 | 0.43 |
| G | 0.48 | 0.53 | 0.78 | 0.49 |
| H | 0.17 | 0.25 | 0.30 | 0.22 |
| I | 1.65 | 1.82 | 2.19 | 1.68 |
| J | 1.03 | 0.94 | 0.95 | 0.97 |
| K | 0.88 | 0.99 | 0.16 | 0.85 |
| L | 0.13 | 0.17 | 0.10 | 0.14 |
| M | 0.13 | 0.17 | 0.10 | 0.14 |
| 25% | 0.17 | 0.25 | 0.1 | 0.22 |
| 75% | 1.03 | 0.99 | 1.35 | 0.97 |
| SD | 1.16 | 1.07 | 1.15 | 1.17 |

TABLE II: Centrality metrics for the example network

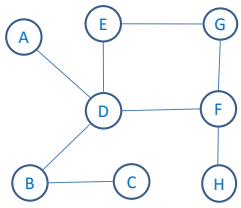
| Peer | Degree | Bonacich | EasyRank | EgVector | PgRank |
|---|---|---|---|---|---|
| A | 0.40 | 0.003 | 0.62 | 0.57 | 0.06 |
| B | 1.00 | 0.09 | 0.78 | 1.00 | 0.14 |
| D | 1.00 | 0.09 | 0.85 | 0.43 | 0.17 |
| F | 0.40 | 0.12 | 0.62 | 0.33 | 0.07 |
| C | 0.60 | 0.09 | 0.71 | 0.58 | 0.09 |
| G | 0.40 | 0.13 | 0.62 | 0.47 | 0.06 |
| H | 0.20 | 0.09 | 0.46 | 0.14 | 0.04 |
| I | 0.60 | 0.14 | 0.76 | 0.43 | 0.09 |
| J | 0.60 | 0.08 | 0.72 | 0.71 | 0.08 |
| M | 0.20 | 0.04 | 0.43 | 0.14 | 0.04 |
| K | 0.60 | 0.01 | 0.72 | 0.74 | 0.08 |
| E | 0.20 | 0.04 | 0.43 | 0.14 | 0.04 |
| L | 0.20 | 0.04 | 0.43 | 0.14 | 0.04 |

of storage to the peers having many friends, compared to pagerank or degree centrality based policies. Finally, the lowest standard deviation indicates that EasyRank-based storage distribution is the most balanced. Therefore, if the peers order their storage distribution based on EasyRank, the allocation is expected to be fair/balanced.

*2) Replica Placement Using Network Centrality:* Table II shows the centrality values for the peers in Figure 1. Suppose, C wants to store two replicas. It has three neighbors B, D and F (highlighted in the table). It is clear that B and F should be chosen more often than D. However, to place two replicas, a random scheme will choose D for at least one replica with probability 0.56. Centrality based schemes, on the other hand, can completely avoid D.

Suppose replica placement is done based on the lowest centrality score. The first replica will be stored at F by all replication schemes (other than Bonacich), because F has the lowest score. Intuitively, this is the correct choice because B and D have many neighbors, which may need to store replicas.

The problem becomes more complex when C needs to store the second replica. If we check the highlighted rows of Table II, we see that B will be chosen by EasyRank and PageRank, while D will be chosen by EigenVector and Bonacich. Therefore, we see that centrality-based storage allocation can achieve the most intuitive solution which is difficult for a random scheme. However, as the example demonstrates, some centrality-based metrics do a better job than others.




| Scheme 1 | | |
|---|---|---|
| From | To | Size |
| D | E,F | 40 |
| D | E,F | 40 |
| D | E,F | 20 |
| B | D | 100 |

| Scheme 2 | | |
|---|---|---|
| From | To | Size |
| D | A,E | 40 |
| D | A,B | 40 |
| D | A,F | 20 |
| B | C | 100 |

| Peer | A | B | C | D | E | F | G | H |
|---|---|---|---|---|---|---|---|---|
| EasyRank | 0.44 | 0.67 | 0.49 | 0.80 | 0.64 | 0.72 | 0.65 | 0.46 |

Storage resource left after scheme 1 and Scheme 2

| Peer | A | B | C | D | E | F | G | H | |
|---|---|---|---|---|---|---|---|---|---|
| Storage Left (Sch 1) | 100 | 100 | 100 | 0 | 0 | 0 | 100 | 100 | 38% peers (A,G,H) cannot use the system any more |
| Storage Left (Sch 2) | 0 | 80 | 0 | 100 | 60 | 80 | 100 | 100 | 100% peers can still use the system |

Fig. 3: An example of a replication scheme introducing skew in storage availability

*3) Replication Using Network Centrality and Available Space:* Although centrality based ranking can lead to better choices for replica placement, the content generation rate should be included into the ranking process. Certain content generation patterns may change the placement choices in future.

Suppose, each peer in figure 3 donates 100MB of storage. Peer B sets the replication factor to 2, and D sets the replication factor to 1. B generates a content of size 100MB, and D generates three content items of sizes 40MB, 40MB, and 20MB. To demonstrate how skewness can hamper the system lifetime as well as its usability, we will place replicas using scheme 1 and scheme 2 from Figure 3.

It is clear from Figure 3 that, if we use scheme 1 to place the replicas, peers A, G, and H will have no available replication storage at all while 50% of the global storage is still unused. The system is not usable anymore for peers A, G, and, H.

On the other hand, scheme 2, which uses ranking based on both centrality and currently available storage, makes replication storage availability more uniform over the network. Scheme 2 combines the EasyRank scores with the available storage at peers in a new metric. The metric is defined as the sum of the EasyRank score of a peer and half of the available storage percentage at this peer. In this case, the centrality has a higher weight than the available storage. In the beginning, D will rank its neighbors before storing the first two replicas. D will calculate the scores of the neighbors which are: A=0.44, B=0.67, E=0.64, F=0.72. D will select A, E and then

```
1: Requires:
2: Centrality values calculated using algorithms
   shown in figure 2
3: Do the following for each replica:
4: Initialize:
5: pot_peers_list ← empty                          ▷ vector of (peerid,rank) tuples
6: cent_list ← empty                               ▷ vector of neighbors' centralities
7: storage_list ← empty                            ▷ vector of neighbors' available storage
8: skip_thr ← THR                                  ▷ pre-defined value for stable balancing
9: w_c, w_s ← WEIGHT                               ▷ weights for centrality and storage values
10: pindex ← 0                                     ▷ index to the pot_peers_list
11: repl_placed ← 0
12: repl_to_place ← number_of_replicas()           ▷ application specific
13: for all neighbors n do
14:     pot_peer.peerid ← n.id
15:     pot_peer.rank ← 0
16:     pot_peers_list.add(pot_peer)
17: end for
18:                ▷ Populate lists with neighbors' centrality and storage availability
19: cent_list ← centrality(neighbors)              ▷ get centralities of all neighbors
20: storage_list ← empty_storage_percent(neighbors)  ▷ get available
    storage of all neighbors
21: for each peer p in pot_peers_list do
22:     p.rank ← w_c * cent_list.get(p.peerid) + w_s *
    storage_list.get(p.peerid)
23: end for
24: sort(pot_peers_list, decreasing)
                ▷ sort the peers in decreasing order of the ranks(p.rank)
25: skip_probability ← random()
26: pcount ← pot_peers_list.length()
27: first_repl ← TRUE
28: while repl_placed < repl_to_place do
29:     if ((first_repl == FALSE)&&((pcount − pindex) >
    repl_to_place)&&(skip_probability < skip_thr)) then
30:         pindex ← pindex + 1
31:         continue                                   ▷ while loop
32:     end if
33:     first_repl ← FALSE
34:     is_stored ← store_data(pot_peers_list.get(pindex))   ▷ store the
    replica in the corresponding peer
35:     if is_stored == true then
36:         pindex ← pindex + 1
37:         repl_to_place ← repl_to_place − 1
38:         repl_placed ← repl_placed + 1
39:     else
40:         pindex ← pindex + 1
41:     end if
42: end while
```

Fig. 4: Replica placement algorithm

will update the scores: A=0.64, B=0.67, E=0.84, F=0.72. The second set of replicas will be stored at A and B (i.e., lowest scores). Next, the updated scores become A=0.84, B=0.87, E=0.84, F=0.72. Now, A and F will be chosen. Afterward, B will calculate the scores of its neighbors: C=0.49, D=0.80. Therefore, C will be selected for storing B's replica. Scheme 2 selects peers which are obvious choices if the peers could see the entire social graph. Therefore, our selection process based on network centrality and the available storage can find the rational choices without revealing the global social graph.

## V. Replica Placement Algorithm

Our algorithm for replica placement uses the following ranking function for the peers:

$R_{ind} = W_{cent} * centrality + W_{st} * storage$

Here, $centrality$ is the centrality score of the peer, and its value is between 0 and 1. The $storage$ is the portion of available storage, and its value is between 0 and 1. Based on extensive experimentation, we chose $W_{cent} = 0.7$ and $W_{st} = 0.3$ for the implementation of our method (centrality has a higher weight than the available storage). Each peer computes the ranking scores of its neighbors before placing replicas for new content. The peers are sorted in increasing order of the ranks. Therefore, peers with low centrality or high amounts of empty storage are at the top of the list. Peers are chosen from the top with a low probability of skipping the current peer and using the next peer in the list. This skipping of comparatively good peers is done to obtain a better global solution [9].

The algorithm is presented in Figure 4. This algorithm requires the EasyRank centrality scores. Lines 5-12 initialize the required data structures(e.g., $pot\_peers$ is the collection of potential peers for the current placement, and $skip\_thr$ is the threshold for skipping a more favorable peer). Lines 13-17 set the list for potential peers. If a peer is not available at the moment due to churn, it is not added to the list. Lines 19-20 asks the potential peers about their EasyRank scores and current available storage. Lines 21-23 calculate the peer ranks, and line 24 sorts the peers based on their ranks. The loop in lines 28-42 runs until all the required replicas are placed or there are no more peers left to consider.

The second condition in the if statement at line 29 makes sure that random skipping will not lead to storing fewer replicas; the third condition skips a favorable peer with probability $skip\_thr$ ($skip\_thr$ is constant and set to 0.1 in our implementation). The *complexity of the algorithm* is linear to the number of neighbors (lines 28-42).

## VI. Experimental Evaluation

We evaluated our scheme through simulation using two social networks collected from Facebook and Google+ [8]. The Google+ network has 4,903 vertices, 72,3321 edges, a density of 0.06, a diameter of 5, and a clustering coefficient 0.24. The Facebook network has 1,034 vertices, 26,749 edges, a density of 0.05, a diameter of 9, and a clustering coefficient of 0.5.

We built a new simulator that fits our problem and allowed us to quickly evaluate the replication methods. We could not evaluate larger graphs due to memory limitations of our simulator. However, we are confident that our solution will work for larger or other types of social networks for two reasons. First, we evaluated it for several other publicly available social graph and found similar results. We could not include these results due to space limitations. Second, the responsibilities of the peers vary based on friend count, not on network size.

We compared our method with two other types of methods: (1) the same algorithm using different centrality-based rankings, and (2) a random method that first selects the peers which have enough storage and then chooses a peer in a random fashion to place the replica.

### A. Evaluation Scenarios and Metrics

We compared the replication schemes for three different scenarios: (1) stable social networks where the graphs do not change and the amount of storage remains the same (i.e., steady state); (2) stable social networks where the graphs do not change, but people are donating additional storage over

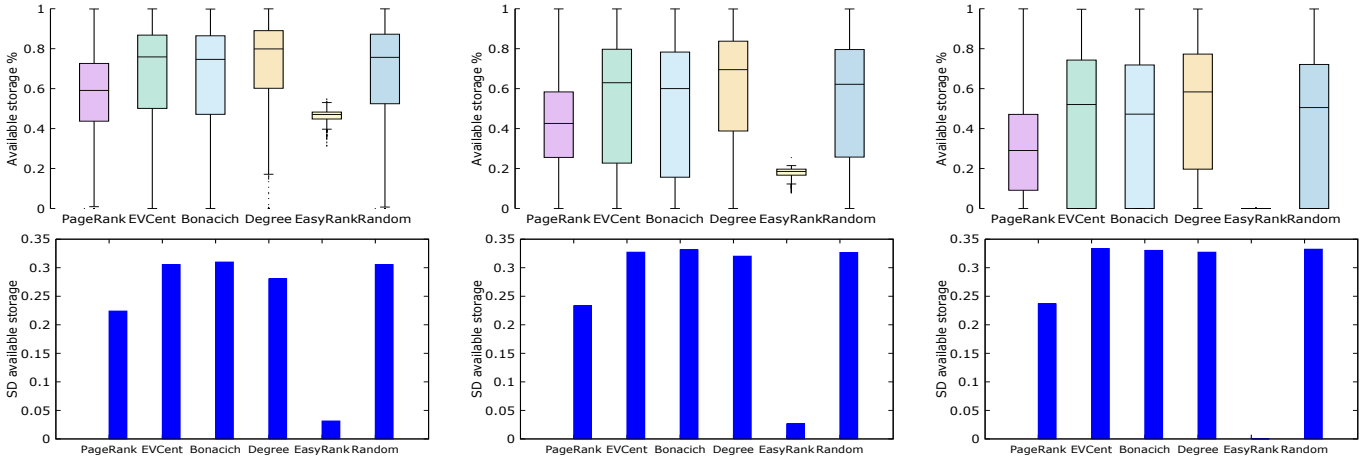



Fig. 5: BoxPlot and standard deviation of the fraction of available replication storage left at peers for the stable Facebook graph: global average available storage 40% (left), 20% (middle), 1% (right)

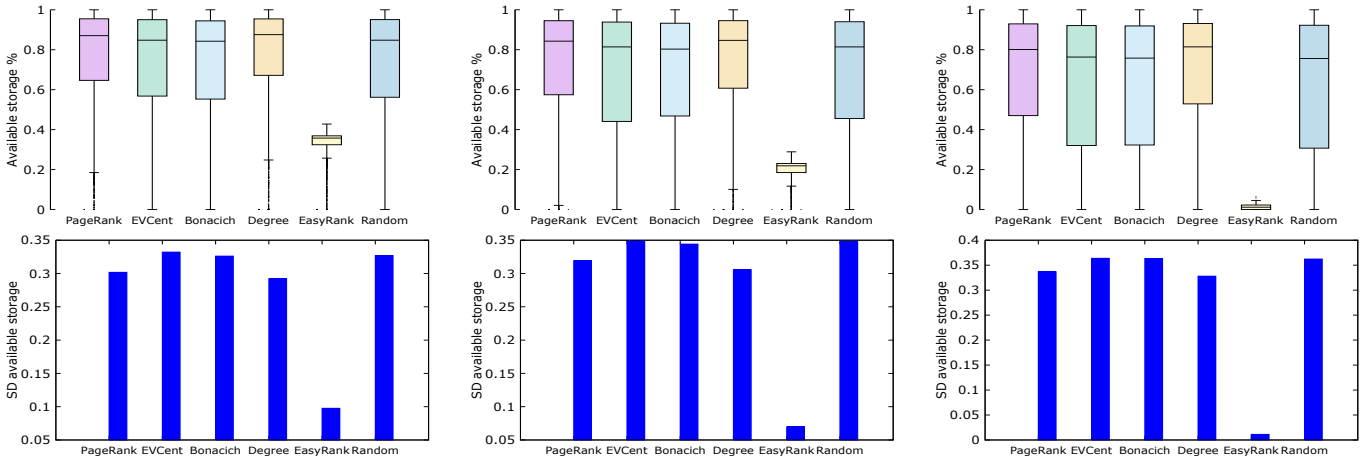



Fig. 6: BoxPlot and standard deviation of the fraction of available replication storage left at peers for the stable Google+ graph: global average available storage 40% (left), 20% (middle), 1% (right)

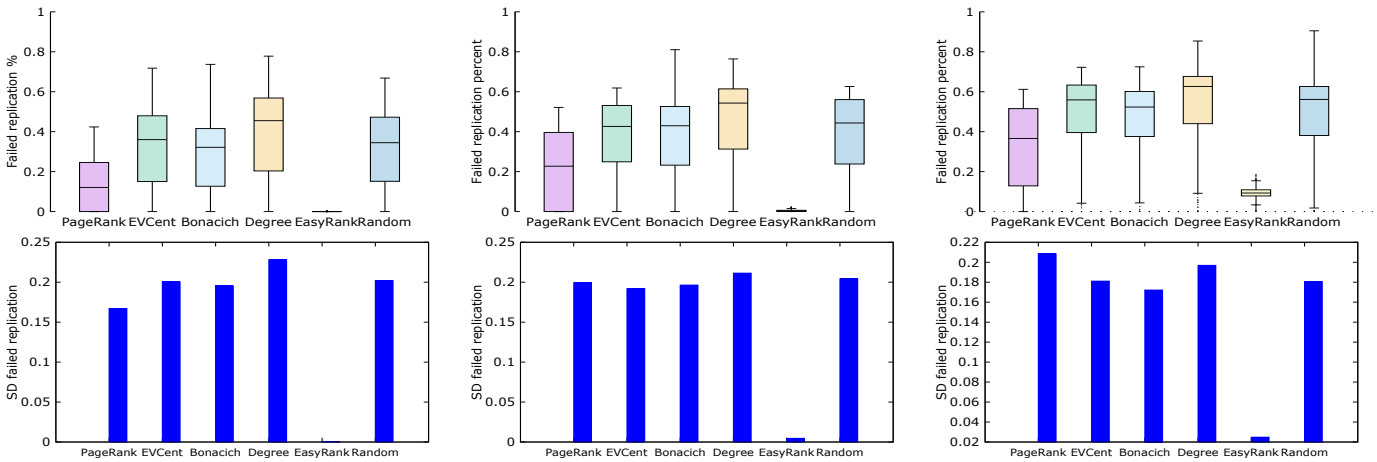



Fig. 7: BoxPlot and standard deviation of the fraction of completely failed replication attempts for the stable Google+ graph: global average available storage 40% (left), 20% (middle), 1% (right)

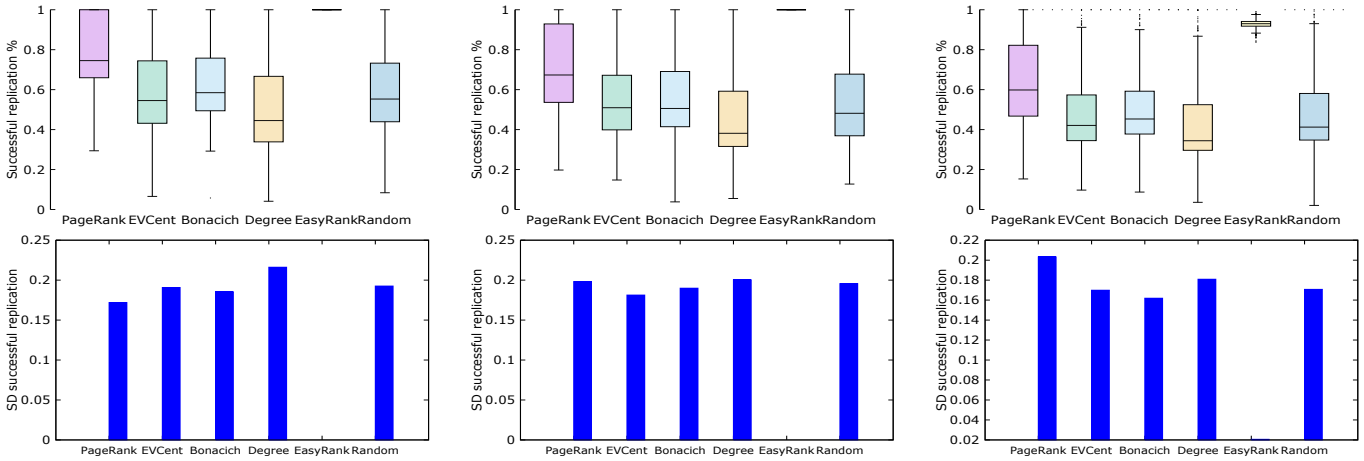



Fig. 8: BoxPlot for the fraction of successful replication attempts for the stable Google+ graph: global average available storage 40% (left), 20% (middle), 1% (right)

TABLE III: Simulation parameters

| Parameter | Value |
|---|---|
| Storage donation | 5-20 GB (from a uniform distribution) |
| Content generation 1 | Pareto ($\alpha$=0.3,$\beta$ = 1.0) with prob=0.1 |
| Content generation 2 | LogNormal (mean=10.0,var=5.0) with prob=0.9 |
| Maximum generated content size | 100 MB |
| Churn | Exponential (mean=0.2) |
| Content generation rate | Top 20% every 6 hour, Rest every 12-36 hour |
| Additional storage donation | 5-20GB |
| Forest fire burning probability | 0.2 |
| Peers removed from FB Graph | 400 |
| Peers removed from GP Graph | 2000 |
| Replica count | 1-3 |
| $W_{cent}$ | 0.7 |
| $W_{st}$ | 0.3 |

time; and (3) emerging social networks, where people are joining the system and new connections are created over time.

For each scenario, we measured the spread and dispersion of the: (1) available replication storage across the network, (2) percentage of failed replication when peers could not store even a single replica of the content, and (3) percentage of successful replication when peers could store all intended replicas. If the values of these three metrics are clustered around the median and have low standard deviation, the replication scheme is balanced/fair and can prevent skewness of storage allocation.

### B. Simulation Setup and Parameters

Table III shows the simulation setup and parameters. For each result, we executed 20 simulation runs. Normally, in social networks, 80% of the content is generated by 20% of the users [20]. Hence, we randomly selected 20% peers, which generate more content than the others. The file sizes are selected from two distributions as shown in the Table III. We generated this mix to represent mostly images, documents, and some other large files. The choice of distributions comes from [21].

After every 24 hours, the peers decide whether to shut down with probability 0.1. Churn duration is generated from an exponential distribution with a mean of 4 hours. This is somewhat lower than usual P2P systems, but we expect the churn in P2P-OSN to be low as the network provides social incentives to keep the peers on [22]. The number of replicas a peer requests (in addition to its own copy) varies from 1 to 3 and comes from a uniform distribution.

### C. Results for Stable Social Networks

The first set of experiments analyze the steady state operations when the social graph and the amount of storage at peers do not change over time. We ran the simulations until the global average available storage space dropped to 1%. We took the snapshot of the metrics (available storage, success rates, and failure rates) when global available storage was 40%, 20%, and 1%.

**Availability of replication storage.** Figures 5 and 6 show the boxplot and standard deviation of the percentage of available storage at the peers for the Facebook and Google+ graphs, respectively. The boxplots show the median, 25$^{th}$ and 75$^{th}$ percentiles, and the overall spread of the available replication storage across the networks. The histograms show the standard deviation of the available replication storage.

The results demonstrate that the EasyRank-based method could maintain the most uniform level of available storage across the network. Peers irrespective of position in the network receive similar availability of replication storage. For all the other methods, the available storage is scattered over long ranges which introduce skewness. Therefore, we conclude that the EasyRank-based scheme provides the most fair and balanced solution.

We also observe that the EasyRank-based scheme can best maintain the uniform availability over the lifetime of the system when the percentage of available storage gradually drops from 40% to 20% and then to 1%. The impact of maintaining uniformity leads to higher replication success rates, as shown in next two experiments.

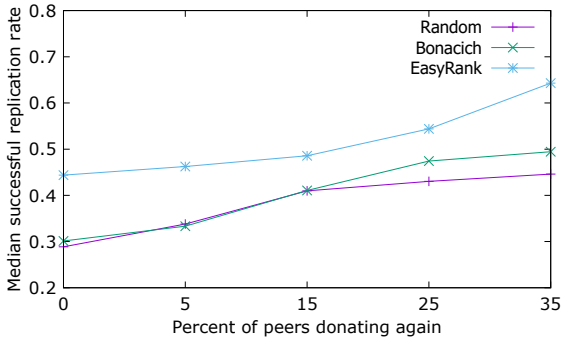



Fig. 9: Percentage of successful replication requests when new storage is added

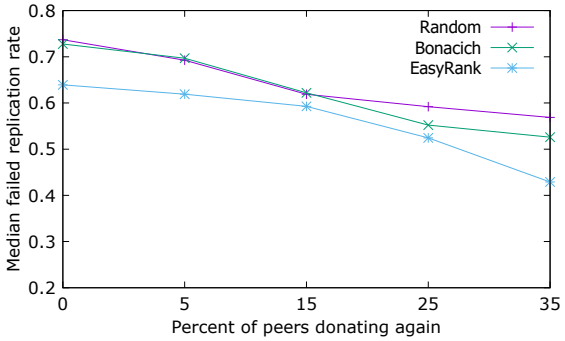



Fig. 10: Percentage of failed replication requests when new storage is added

From now on, due to space limitations, we will show results only for the Google+ network, since the results for the Facebook network are similar.

**Replication failure rate.** The impact of non-uniform/imbalanced available replication storage consists of many failed attempts to store even a single replica of the content. Figure 7 shows the fraction of failed replication attempts for the stable Google+ network. We observe that the median failure rate for the EasyRank-based scheme is at most 7% over the lifetime of the system, while the median failure rate for the other schemes is as high as 60%.

**Replication success rate.** Figure 8 shows the fraction of successful replication requests (i.e., the desired replication factor is satisfied). We again observe the effect of efficient storage allocation for the EasyRank-based method: as long as storage is available, peers could store all the desired replicas. The success rate is close to 95% until the available replication storage drops to 1%. The comparison schemes could achieve at best a success rate of 60%.

### *D. Results for Stable Networks with Storage Addition*

To assess the performance when storage is dynamically added during the network lifetime, we randomly select a number of peers that add storage after their initial contributed storage (5-20 GB per) is 95% allocated. The rest of the peers do not add space. For this experiment, the percentage of peers contributing additional storage is varied from 5% to 35% of the total number of peers. These peers donate extra 5-20GB of storage.

The goal of this experiment is to find out how sensitive the replication schemes are to storage addition. A good scheme is expected to be able to readily use the newly donated storage and should exhibit comparatively lower replication failure rates and higher replication success rate.

Figures 9 and 10 show the median values of the fraction of successful replication requests and the fraction of failed replication requests, respectively. We only plot the curves for EasyRank, Bonacich, and random schemes. Eigenvector and pagerank schemes show similar results to Bonacich. The results demonstrate that EasyRank always performs better than the other schemes and exhibits steeper improvements if more than 30% peers contribute storage to the system. The results are worse than the ones for the stable networks because there is a large storage imbalance between the peers adding storage (5%-35%) and those not adding (65%-95%). The result of this imbalance is that many peers fill up before the average available storage across all the peers becomes 1%. This phenomenon does not happen in the previous experiments. Thus, more replication requests will completely fail or will not be able to store the desired number of replicas.

### *E. Results for Emerging Social Networks*

This experiment evaluates the performance while new vertices and new edges are being added to the network. We used the forest fire method [23] to generate back in time social graphs. Using this method, we removed 2000 vertices (and corresponding edges) and additional random edges from the networks. We recorded the exact order of removing the edges and vertices. We started the simulation with the back in time social graph and kept adding back vertices and edges in the same order they were removed, one per hour.

Figure 11 shows the results for the emerging social networks. We only considered the degree centrality, EasyRank, and random schemes. Since the graph is changing, there is no simple way to find the correct centrality values for the other schemes. We ran the simulation until the global average available storage dropped to 5%. The results demonstrate that EasyRank achieves the most balanced/uniform usage, the lowest failure rate, and the highest success rate.

## VII. Conclusion

This paper presented an efficient replication method for storing replicas of user generated content in P2P-OSN, which uses EasyRank, a new network centrality metric. We designed our solution to be fair and balanced: peers can use the system irrespective of the number of friends they have. We evaluated the EasyRank-based replication method with real social graphs and compared it against other centrality-based replication mehtods and a random replication method. The results demonstrate that our solution allocates the storage in a fair and balanced way, which allows for an increased

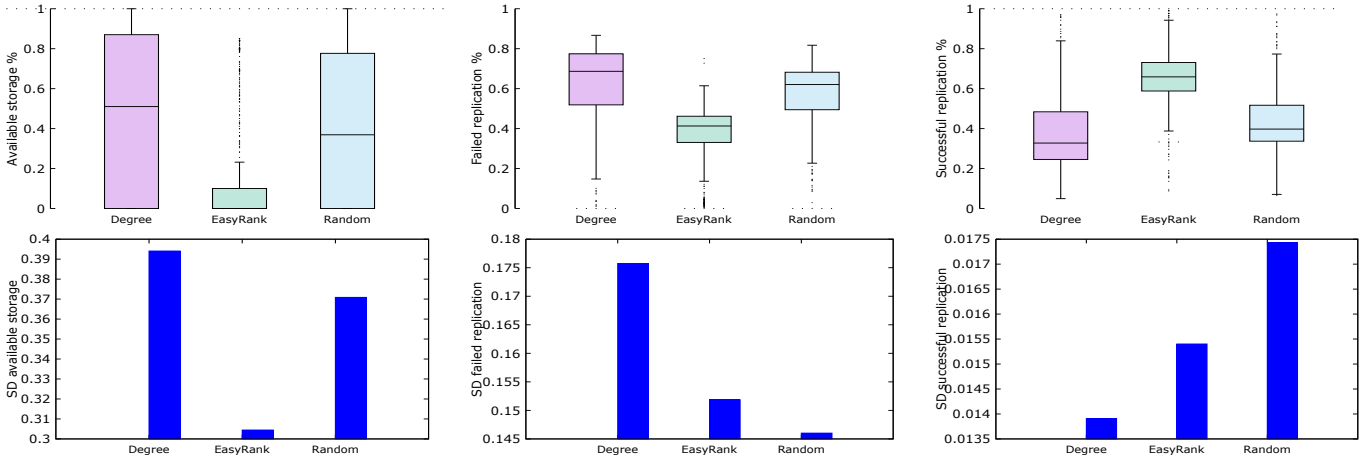



Fig. 11: Boxplot and Standard deviation of available replication storage (left), failed replication(middle), and successful replication (right) for emerging social network

lifetime of the system. The peers can find replication storage at neighbors as long as there is global storage available.

## ACKNOWLEDGMENT

This research was supported by the National Science Foundation (NSF) under Grants No. CNS 1409523 and DGE 1565478, and the National Security Agency (NSA) under Grant H98230-15-1-0274. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of NSF and NSA. The United States Government is authorized to reproduce and distribute reprints notwithstanding any copyright notice herein.

## REFERENCES

[1] N. Kourtellis and A. Iamnitchi, "Leveraging peer centrality in the designof socially-informed peer-to-peer systems," *Parallel and Distributed Systems, IEEE Transactions on*, vol. 25, no. 9, pp. 2364–2374, 2014.

[2] R. Narendula, T. G. Papaioannou, and K. Aberer, "A decentralized online social network with efficient user-driven replication," in *Privacy, Security, Risk and Trust (PASSAT), 2012 International Conference on and 2012 International Confernece on Social Computing (SocialCom)*. IEEE, 2012, pp. 166–175.

[3] L. Han, M. Punceva, B. Nath, S. Muthukrishnan, and L. Iftode, "Socialcdn: Caching techniques for distributed social networks," in *Peer-to-Peer Computing (P2P), 2012 IEEE 12th International Conference on*. IEEE, 2012, pp. 191–202.

[4] R. Narendula, T. G. Papaioannou, and K. Aberer, "Privacy-aware and highly-available osn profiles," in *Enabling Technologies: Infrastructures for Collaborative Enterprises (WETICE), 2010 19th IEEE International Workshop on*. IEEE, 2010, pp. 211–216.

[5] N. Kourtellis, J. Finnis, P. Anderson, J. Blackburn, C. Borcea, and A. Iamnitchi, "Prometheus: User-controlled p2p social data management for socially-aware applications," in *Middleware 2010*. Springer, 2010, pp. 212–231.

[6] J. M. Pujol, V. Erramilli, G. Siganos, X. Yang, N. Laoutaris, P. Chhabra, and P. Rodriguez, "The little engine (s) that could: scaling online social networks," in *ACM SIGCOMM Computer Communication Review*, vol. 40, no. 4. ACM, 2010, pp. 375–386.

[7] D. A. Tran, K. Nguyen, and C. Pham, "S-clone: Socially-aware data replication for social networks," *Computer Networks*, vol. 56, no. 7, pp. 2001–2013, 2012.

[8] L. Jure. (2016) Stanford large network dataset collection. [Online]. Available: http://snap.stanford.edu/data/index.html#socnets

[9] S. Kirkpatrick, D. G. Jr., and M. P. Vecchi, "Optimization by simmulated annealing," *science*, vol. 220, no. 4598, pp. 671–680, 1983.

[10] D. N. Tran, F. Chiang, and J. Li, "Friendstore: cooperative online backup using trusted nodes," in *Proceedings of the 1st Workshop on Social Network Systems*. ACM, 2008, pp. 37–42.

[11] K. Chard, S. Caton, O. Rana, and K. Bubendorfer, "Social cloud: Cloud computing in social networks," in *Cloud Computing (CLOUD), 2010 IEEE 3rd International Conference on*. IEEE, 2010, pp. 99–106.

[12] A. et al., "Farsite: Federated, available, and reliable storage for an incompletely trusted environment," *ACM SIGOPS Operating Systems Review*, vol. 36, no. SI, pp. 1–14, 2002.

[13] A.-K. Pietiläinen, E. Oliver, J. LeBrun, G. Varghese, and C. Diot, "Mobiclique: middleware for mobile social networking," in *Proceedings of the 2nd ACM workshop on Online social networks*. ACM, 2009, pp. 49–54.

[14] P. Stuedi, I. Mohomed, M. Balakrishnan, Z. M. Mao, V. Ramasubramanian, D. Terry, and T. Wobber, "Contrail: Enabling decentralized social networks on smartphones," in *Middleware 2011*. Springer, 2011, pp. 41–60.

[15] 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 *Proceedings of the 1st ACM Workshop on Mobile Cloud Computing & Services: Social Networks and Beyond*. ACM, 2010, p. 8.

[16] (2016) Join diaspora. [Online]. Available: https://joindiaspora.com/

[17] D. Liu, A. Shakimov, R. Cáceres, A. Varshavsky, and L. P. Cox, "Confidant: Protecting osn data without locking it up," in *Middleware 2011*. Springer, 2011, pp. 61–80.

[18] A. Shakimov, H. Lim, R. Cáceres, L. P. Cox, K. Li, D. Liu, and A. Varshavsky, "Vis-a-vis: Privacy-preserving online social networking via virtual individual servers," in *Communication Systems and Networks (COMSNETS), 2011 Third International Conference on*. IEEE, 2011, pp. 1–10.

[19] S. Sodsee, "Placing files on the nodes of peer-to-peer systems," Ph.D. dissertation, FernUniversität in Hagen, 2012.

[20] L. Guo, E. Tan, S. Chen, X. Zhang, and Y. E. Zhao, "Analyzing patterns of user content generation in online social networks," in *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2009, pp. 369–378.

[21] M. Mitzenmacher and B. Tworetzky, "New models and methods for file size distributions," in *Proceedings of the Annual Allerton Conference on Communication Control and Computing*, vol. 41, no. 1. The University; 1998, 2003, pp. 603–612.

[22] J. Li and F. Dabek, "F2f: Reliable storage in open networks." in *IPTPS*, 2006.

[23] J. Leskovec and C. Faloutsos, "Sampling from large graphs," in *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2006, pp. 631–636.