Efficient Dissemination in Decentralized Social Networks

Giuliano Mega, Alberto Montresor, and Gian Pietro Picco
Department of Engineering and Information Science (DISI), University of Trento, Italy
Email: {mega, montresor, picco}@disi.unitn.it

***Abstract*—Online social networks (OSN) have attracted millions of users. This enormous success is not without problems; the centralized architectures of OSNs, storing the users' personal data, provides ample opportunity to privacy violation. These problems have raised the demand for open, decentralized alternatives. We tackle the research question: is it possible to build a decentralized OSN over a *social overlay*, i.e., an overlay network whose links among nodes *mirror* the social network relationships among the nodes' owners? This paper provides a stepping stone to the answer, by focusing on the key OSN functionality of disseminating profile updates. Our approach relies on gossip protocols. We show that mainstream gossip protocols are inefficient, due to the properties that characterize social networks. Therefore, we leverage these very same properties towards our goal, by appropriately modifying the forwarding rules of gossip protocols. Our evaluation, performed in simulation over a crawled real-world social network, shows that our protocols provide acceptable latency, foster load balancing across nodes, and tolerate churn.**

I. Introduction

Over the last decade, *on-line social networks* (OSNs) rose from anonymity to stardom. The meteoric growth rates of OSN sites like MySpaces and Facebook surpassed even the most optimistic predictions [1], reaching hundreds of millions of users worldwide. Unfortunately, this growth has only been rivaled by the controversy surrounding OSNs. Their centralized architectures place sensitive user data at the mercy of the companies, a situation that has led to a series of widely publicized incidents [2]. Apart from inducing distrust across the user base, these incidents have also provided momentum to the research and development of open, decentralized alternatives, providing the context and motivation for our work.

Among the features offered by OSNs, *profile-based communication* is arguably the most important, and the one we focus on in this paper. User *profiles* are personal web pages where users freely post content—e.g. text snippets, pictures, videos, and music [3]. In response to these postings other users, usually friends, post comments and other content. A *newsfeed* displays to users the most recent updates from their friends. Indeed, a recent study by Benevenuto et al. on OSN usage patterns [4] shows that $\sim$90% of server requests in Google's Orkut were about profile page content.

In a decentralized OSN, users run P2P clients (peers) on their hosts to browse the profiles of friends and post content. Peers form an overlay network with the purpose of collectively sharing and replicating content, serving it on behalf of offline users when needed. Current proposals to P2P OSNs [5]–[7] often rely on DHTs. These are, however, agnostic of social relationships: update routing and content storage is orthogonal to the social network, and results in unnecessarily long communication links and security issues.

An alternative is the *friend-to-friend* approach [8], where communication among nodes is enabled only if their owners know each other. Different nuances of this notion exist. Here, we choose the most radical one where peers are arranged in a *social overlay*, a one-to-one mapping mirroring the underlying social network. This choice provides several benefits:

- *Connect the right people*. People tend to talk to people they know. Recent studies [4] show that this is also true of OSNs, where about 78% of user interactions take part within one-hop neighborhoods. A social overlay enables short routes and constrains traffic to small network areas. Further, social networks are highly clustered [9], and are therefore likely to offer enough resilience to churn.
- *Improve locality*. People tend to connect to people that are alike, with geographical co-location playing a key role [10]. As a result, not only routes over social overlays are likely to be short, but also physically localized.
- *Provide peer incentives.* Unlike links in a DHT, links in a social overlay represent friendship. The hypothesis here is that friends are more likely to cooperate with other friends than with random strangers.
- *Improve privacy.* Provided that an authentication mechanism is in place [11], only the identity of the original poster needs to be checked, while privacy issues are mitigated since communication is restricted to friends.

Unlike traditional P2P applications such as file sharing, OSNs are quasi-interactive systems. Although real-time interaction is not required, profile updates should become rapidly available to friends currently online. In our proposal, this is achieved by relying on an efficient *profile update dissemination* protocol. This protocol pushes profile updates from friends, to friends, in a reliable, timely, and efficient fashion. Peers cache locally the updates to the content of their friends' profiles. Therefore, when a user browses a friend's profile, all data accessed is already locally available. After describing in Section II the system model we use as a reference, in Section III we further detail the problem, our approach, and the experimental framework we use for evaluation.

Several alternatives can in principle be used to deal with the problem. We focus on gossip protocols [12] because they were originally designed for update dissemination, albeit in a slightly different context, and because of their inherent simplicity, scalability, and tolerance to topology changes.

Unfortunately, gossip protocols are designed to operate on uniform random graphs, which have very different properties w.r.t. social networks. We assess quantitatively the negative impact of this assumption mismatch in Section IV.

The main contribution of this paper lies in Section V and VI where we describe and, respectively, evaluate a protocol for efficient dissemination over social networks. The protocol is a combination of known techniques and concepts—besides gossip, the use of message histories, and of a biased selection heuristic geared towards the particular properties of social networks. Nevertheless, their combination and application to an overlay mirroring the real-world social network is, to the best of our knowledge, original. Moreover, our evaluation shows that the approach is applicable in practice to the scale of friend neighborhoods found in real-world social networks.

Finally, Section VII concisely surveys related work, and Section VIII ends the paper with brief concluding remarks.

II. System Model

Definitions and notation. We model a *social network* as an undirected graph $G = (V, E)$, where V contains users and E is the friendship relation among them. For each user $u \in V$, function $f : V \rightarrow 2^V$ maps users to their set of friends, while f_2 maps users to their set of friends of friends:

$$f(u) = \{\ v \mid (u, v) \in E\ \}$$
$$f_2(u) = \{\ w \mid w \in f(v) \wedge v \in f(u)\ \}$$

Let $f^*(u) = f(u) \cup \{u\}$ be the *social neighborhood* of u; $f^*(u)$ induces a subgraph in the social network which is composed by the vertices in $f^*(u)$ and its interconnections. We refer to this subgraph as G_u.

Since we map nodes into users in the social network, we refer to users and nodes interchangeably. Sentences like *"a node u and its friends $f(u)$"*, then, are to be interpreted as *"a node controlled by a user u and the nodes controlled by its set of direct friends, $f(u)$"*. We also assume that the mapping between nodes and users is one-to-one (i.e., a user does not log in from more than one node at the same time).

Finally, let O be the set of all updates generated in the system, then $prof : O \rightarrow V$ is the function mapping an update to the owner of the profile page to which it was posted.

Dynamism. A social network is a dynamic concept: new friendship relationships may be formed or old ones severed. In practice, however, such changes are infrequent, at least w.r.t. the time scale of our problem of update dissemination. Therefore, for the purpose of this paper we consider the social network to be immutable. The dynamism of social overlays cannot be, however, disregarded in the same way. Users may start and stop their nodes at will: a particular user may not be available for extended periods of time. In P2P terminology, this phenomenon is known as *churn*. The social overlay undergoes frequent reconfiguration, and this must be taken into account in evaluating alternatives for update propagation. We analyse this aspect in the context of our solution in Section VI-B.

Realizing the social overlay. We assume that nodes able to discover the IP addresses of their currently online friends, to enable message exchange. The overlay management problem is outside the scope of this paper, but could be addressed by leveraging off of existing decentralized server infrastructures such as the widely available Jabber/XMPP network [21]: managing presence is already an integral part of what Jabber does, and we could simply piggyback on that.

Knowledge of the social network. We assume user u knows not only the set $f(u)$, but also $f_2(u)$; this knowledge makes it possible, given a friend $v \in f(u)$, to obtain the set of common friends $f(u) \cap f(v)$. This assumption is reasonable because in modern OSNs, the set of friends is already part of the profile and incremental updates (in the form of new friendship events involving your friends) are shown in the newsfeed.

Authentication, access control and privacy. We assume that each node u is identified by a pair of asymmetric keys $(Priv_u, Pub_u)$. The public keys of nodes in $f(u) \cup f_2(u)$ are acquired when learning new friendship relations. To guarantee authenticity, every update o generated by u must be signed with $Priv_u$. More complex access control mechanisms are possible [11], although outside of the scope of this paper. Even though privacy is guaranteed by the fact that updates are sent only to the intended destinations, it is always possible that, once decrypted, content is disclosed to unauthorized third parties. This is not exclusive to our approach, however, and could also happen in a centralized OSN. Finally, impersonation attacks are less likely due to personal knowledge of friends.

III. Problem, Approach, and Experimental Framework

In this section we state our problem, outline the proposed approach, and illustrate the metrics and experimental setup we use in the rest of the paper.

A. Problem Statement

As mentioned in Sec. I, we envision a decentralized OSN where updates to user profiles are proactively disseminated to all friends. Fig. 1 illustrates how it works. Users u, v, and w are such that $v, w \in f(u)$ but $v \notin f(w)$, i.e., u is friend with v and w, but v is not friend with w. In the interaction depicted:

1) u posts a picture to his local copy of his own profile;
2) the system disseminates the update to v and w, who update their local copies;
3) upon seeing the picture, v posts a comment to it. This goes into v's local copy of u's profile;
4) the system disseminates the comment to u and to w, who update their local copies.

Storing the content of all friends may seem overkill, but pretty much everything in the profile pages of modern-day

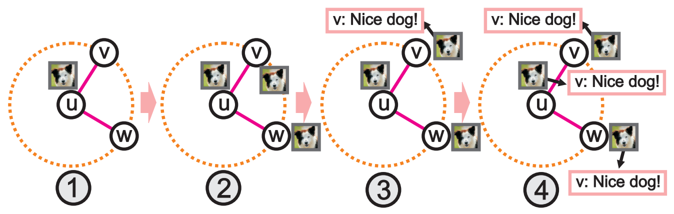


Fig. 1. User u posts a photo, user v posts a comment, user w just watches.

OSN—comments, links, thumbnails, small pictures—are *small objects* (e.g., under 60 KB, the size of current Facebook pictures). The only exception are movies and large collections of high-quality pictures: however, these are usually not part of profiles anyway, and are linked from services such as YouTube and Flickr. Their associated "metadata" (e.g., description, thumbnail, etc.) are instead small objects.

In this example, w sees a comment posted by v although the two are not friends, because the comment was posted to the profile of u, who is friend with w. This is the default behavior of OSNs. Note, however, that this would in principle enable w to, say, try to modify u's profile without asking for permission by disseminating an update that, say, changes u's name. The simple authentication mechanism suggested Sec. II would be enough to prevent these situations.

We envision interactions like the above to be supported by a *social newscasting* service consisting of two simple primitives:

1) the operation postToProfile($\text{PROFILE } v, \text{UPDATE } o$) enables u to post update o to the profile of $v = \mathit{prof}(o)$;
2) the callback newsReceived($\text{LIST}\langle\text{UPDATE}\rangle$ *list*) enables clients to retrieve new content upon arrival.

In the context of OSNs, realizing this social newscasting service essentially trades the problem of *fetching data* over the network with the problem of *disseminating updates* to all friends. The latter must be performed in a *timely* way, i.e., with a latency comparable to the one of today's centralized OSNs. In other words, for a given subgraph G_v, *any* node $u \in G_v$ may produce, at any point in time, some update o by calling postToProfile(v, o). The goal is to define a strategy to efficiently disseminate o from u to all nodes in $f^*(v)$.

B. Approach Outline

Owing to their wide recognition as robust tools for fast data dissemination, we chose to use *gossip protocols* [13] as the base for our social newscasting service. These protocols are typically run in *rounds*, in which each node selects a gossip partner using a *randomized selection heuristic* and exchanges data with it based on an *exchange strategy*. Rounds are *not* synchronized; rather, they simply play a "rate limitation" role.

Our social newscasting employs two gossip protocols. A *rumor mongering* protocol based on a *push* exchange strategy is used to disseminate updates quickly, possibly at the expense of guaranteed delivery (e.g., for users currently not online). This fast but unreliable dissemination is therefore complemented by an *anti-entropy* [13], *push-pull* protocol. This runs in the background at a slower pace w.r.t. rumor mongering and guarantees that all nodes that become and remain online for long enough eventually receive all updates. In a sense, anti-entropy serves as a "safety net", patching the message deliveries missed by rumor mongering.

In this paper, we focus *exclusively* on the first protocol (i.e., rumor mongering) because, as we show in Section IV, its application to social networks already requires a significant departure from the mainstream. Section V discusses in detail how we improve on the base rumor mongering by choosing selection strategies that explicitly take into account topological properties of social networks, such as centrality and clustering.

C. Experimental Framework

Unit experiments. Protocols are evaluated over a large number of isolated *unit experiments*, whose results are aggregated offline to obtain global figures. A unit experiment consists of: *i)* singling out a subgraph G_v; *ii)* having v publish an update by calling postToProfile(); *iii)* waiting until the push protocol terminates; *iv)* measuring the desired parameters.

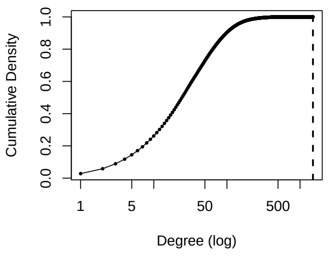
Each unit experiment e is associated with the dissemination of a single update generated by the profile owner, $\mathit{root}(e)$. Updates posted to profiles of friends would behave similarly and are thus excluded from the evaluation.

Performance metrics. Given a set E of unit experiments, we measure the following:

- *Residue*, i.e., the percentage of nodes who did not receive the update before the push protocol terminated. Let $\mathit{undelivered} : E \to \mathbb{N}$ be a function yielding the number of these nodes for a single unit experiment e. Then:
$$\mathit{residue}(E) = \frac{\sum_{e \in E} \mathit{undelivered}(e)}{\sum_{e \in E} |f(\mathit{root}(e))|}$$
- *Average and maximum latency*, measured in number of rounds, $t_{avg}(E)$ and $t_{max}(E)$. Defined as the number of rounds it takes for an update to reach a destination, minus the rounds for which that destination has been offline after the update was posted (measured relative to node uptime).
- *Absolute and average load*, in terms of messages sent and received. Formally, let $\ell_s : V \times E \to \mathbb{N}$ be the function yielding the number of messages sent by node $v \in V$ during a unit experiment e, and ℓ_r a similar function yielding the messages v received. The load for a given unit experiment is then $\ell(v, e) = \ell_s(v, e) + \ell_r(v, e)$. The absolute load over the entire set E of experiments at node v, and the average load, normalized over the size of the neighborhood of v, are then given as:
$$\mathit{load}(v) = \sum_{e \in E} \ell(v, e) \qquad \overline{\mathit{load}}(v) = \frac{\mathit{load}(v)}{|f^*(v)|}$$
It is useful to generalize this notion to sets of nodes. The average load incurred on a set of nodes $V' \subset V$ is
$$\overline{\mathit{load}}(V') = \frac{\sum_{v \in V'} \mathit{load}(v)}{\sum_{v \in V'} |f^*(v)|}$$
- *Duplicate ratio*, i.e., the ratio between the number of messages generated and delivered. For a unit experiment e, the former is
$$\mathit{generated}(e) = \frac{1}{2} \sum_{w \in f^*(v)} \ell(w, e)$$
and the latter is $\mathit{delivered}(e) = |f(v)| - \mathit{undelivered}(v)$. The duplicate ratio is expressed as:
$$\mathit{dup}(E) = \frac{\sum_{e \in E} \mathit{generated}(e)}{\sum_{e \in E} \mathit{delivered}(e)}$$

Vertices	72303
Edges	1508283
Avg. Degree	41
Max. Degree	1500
Min. Degree	1
Clust. coef.	0.34

(a) Summary.



(b) CDF of node degrees.

Fig. 2. Dataset characteristics.

- *Unit load balance*, i.e., how balanced the load was within a single unit experiment and therefore a single message dissemination. We capture it by using a dimensionless dispersion measure known as the *coefficient of variation* CV. For a given set of observations, CV is defined as $\frac{\sigma}{|\mu|}$, where σ stands for the sample standard deviation, and μ as the sample average. For a given unit experiment e, the coefficient of variation is computed over the set of load values $\{\ell(v_1, e), \cdots, \ell(v_n, e)\}$, where $v_i \in f^*(v)$, and this yields a relative measure of load balancing. The higher the CV, the less balanced is the load.

Setup. We use the PEERSIM [22] simulator, along with a social graph obtained from a popular OSN site through a snowball sampling procedure [23] over a single seed. Publicly-available friend connections were explored until we had a partial third level. The sample characteristics are summarized in Table 2a, and its cumulative degree distribution shown in Fig. 2b. The small average degree is due to the sampling procedure: since we had to stop crawling at some point, nodes at the outermost layer—the most numerous—are inevitably missing friends. Despite the small size w.r.t. real social networks, we argue this sample is adequate for simulation purposes. We chose to use a real, albeit smaller, network instead of a synthetic one because current models are still very limited [24].

Unless otherwise noted, we run 10 unit experiments per node, yielding 723,030 repetitions for each combination of protocols and parameters.

IV. A Fresh Look at Mainstream Techniques

An obvious question at this point is *"Can't we just re-use mainstream gossip protocols?"*. The answer to this question is negative, as we demonstrate quantitatively in this section, motivating the contribution described in the rest of the paper.

Demers' rumor mongering [13] is arguably one of the most well-known gossip push protocols in the literature. We consider the *feedback/coin* variant of Demers', where each node keeps a list of "hot rumors", i.e., updates its friends are more likely not to have. In a gossip round, each node u: *i)* selects a node v from its neighborhood *uniformly at random*; *ii)* sends all, or part of its list of hot rumors to v; *iii)* collects a *response vector* from v which tells which rumors v already knew and which it did not.

Then, for each item in the response vector *i)* if the rumor was not known to v, then nothing is done; *ii)* otherwise, the rumor is removed from the hot rumor list with probability p. Node v, in turn, adds to its hot rumor list the new rumors received from u. Further, whenever a new piece of content is posted locally it is immediately added to the poster's local hot rumor list and becomes eligible to dissemination.

We evaluated Demers' under our simulation framework, performing a single unit experiment per node. Aggregate results over the 72,303 unit experiments are shown in Table I. The row "Demers' residue" contains the values reported in [13], for reference. Fig. 4 shows the average load at each node as a function of node degree. Residues are orders-of-magnitude larger than expected, and average loads are large even for low degree nodes: one of the nodes with degree 1, for example, has an average load of 70. These issues arise from the fact that *social networks violate a fundamental assumption in gossip protocols, i.e., clustering in the network is approximately uniform.* Fig. 3a provides an example where:

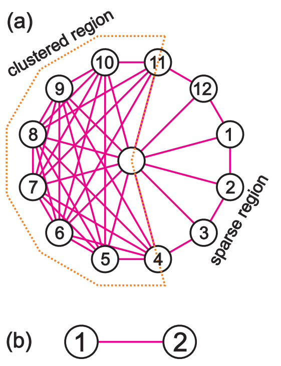


Fig. 3. Problematic neighborhoods for Demers' protocol. (a) Irregular clustering. (b) A small neighborhood.

1) Node u calls postToProfile().
2) As the clustered region is larger than the sparse one, it is going to be hit first with higher probability ($\frac{2}{3}$ over $\frac{1}{3}$).
3) Dissemination proceeds very quickly in the clustered region but also generates lots of duplicates.
4) Eventually, most (if not all) nodes in the clustered region get the update. The protocol, however, keeps selecting the clustered region with higher probability.
5) Every time a node selects an already-infected node it gives up on the rumor with probability p. Thus, the nodes in the clustered region (u included) are likely to give up too soon, as they "perceive" the message infection to have spread, when in fact many nodes may still await it.

As for load, a node generates $1/p$ duplicates before giving up on a rumor, as confirmed in Table I. For a neighborhood as in Fig. 3b, this means node 1 will generate around 10 messages

	$p = 0.4$	$p = 0.3$	$p = 0.2$	$p = 0.1$
Observed Residue	19%	14%	10.5%	6%
Demers' Residue	$\sim 3.7\%$	$\sim 1.1\%$	$\sim 0.01\%$	$--$
Dup. Ratio (avg.)	2.53	3.37	5.04	10.03
Avg. Load	5.66	7.35	10.6	20.3
t_{max}	59	68	79	124
t_{avg}	4.3	4.5	4.7	4.94

TABLE I
RESULTS FOR DEMERS' PROTOCOL.

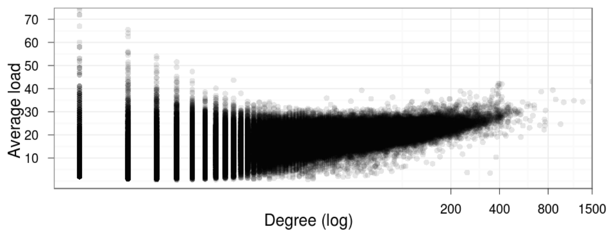


Fig. 4. Average loads by degree.

every time it publishes an update, and will receive 10 messages every time node 2 publishes an update. Since we run one unit experiment per neighbor, this amounts to an average expected load of 20, which is very high for such a small neighborhood. Other topologies might generate even larger averages.

V. Gossip in Social Overlays

Demers' rumor mongering is inefficient over social overlays as it is unable to cope with its graph properties: we propose a dissemination protocol that is aware of, and exploits them.

A simple way to solve Demers' residue issue is *flooding*: a node does not give up spreading an update until all of its friends receive it. Pure flooding, however, generates too much traffic. Therefore, we enhance it by piggybacking *histories* on each message, recording who received a given update. As nodes do not re-send an update to nodes known to have it, we reduce traffic significantly.

There is, however, another relevant effect of suppressing transmissions. Our flooding protocols are gossip-based, and thus employ a randomized neighbor selection heuristic, the simplest being selecting nodes uniformly at random. However, the latter is biased towards higher degree nodes. This, perhaps contrary to intuition, hurts performance instead of helping it. The reason is in Fig. 3a: if higher degree nodes are packed in a cluster, random selection tends to starve regions with lower clustering. Nevertheless, the progressive exclusion of nodes from highly clustered regions (which are the ones being selected first and thus entering the piggybacked histories first) helps us to eventually steer selection towards regions containing nodes not known to have received the update.

Flooding with message histories provides the base dissemination mechanism, which we improve with a pair of selection heuristics. Indeed, even if we eventually steer selection towards less favored regions, higher clustered regions are still heavily flooded with messages in the early stages of dissemination, when histories did not yet propagate. This not only causes more traffic, but also slows down the protocol.

We reap further improvements with a selection heuristic which steers away from high degree nodes from the very beginning, by picking nodes with a probability inversely proportional to their degree, favoring selection of lower-degree nodes. Finally, social neighborhoods are often divided into disjoint components, tied by a "central" node. The neighborhood in Fig. 5b, for example, has 4 such components. We can speed up dissemination significantly if the central node spreads the update by by hitting components in sequence, beginning with the one with the highest degree, down to the lowest degree. This way, the central node exploits its special position, keeping the neighborhood connected, to "parallelize" dissemination.

In the rest of this section we detail further these concepts.

A. *Flooding with Histories*

The baseline protocol FLOOD works as follows. When a node v learns about an update o belonging to the profile page of node $u \in f^*(v)$, it keeps sending one message per round to the common friends $f^*(v) \cap f^*(u)$ that may have not received o, stopping only once it knows that all of them have received it. In the variant with message histories, called HFLOOD, nodes piggyback histories in their messages, sharing their knowledge about nodes that have received o. If u and v do not share common friends, then v simply cannot help disseminating o.

Formally, let $K_{v,o} \subseteq f^*(u)$ contain the intended destinations of o known by v to have received o, and $E_{v,o} = (f^*(u) \cap f^*(v)) - K_{v,o}$ denote the common friends *eligible* for selection at v when considering update o. Then, at each round, node v:

1) selects $w \in E_{v,o}$ according to a selection heuristics;
2) sets $K_{v,o}$ to $K_{v,o} \cup \{w\}$;
3) sends o to w.

Whenever v receives an update o from a node z, it:

4) sets $K_{v,o}$ to $K_{v,o} \cup \{z\}$.

The variant of FLOOD that piggybacks histories in messages, called HFLOOD, substitutes steps (3) and (4) as follows:

3) sends a message $\langle o, K_{v,o} \rangle$ to w, where the *message history* of o known by v is piggybacked with o;
4) when v receives a message $\langle o, K_{z,o} \rangle$ from z, it sets $K_{v,o}$ to $K_{v,o} \cup K_{z,o}$.

In both cases, dissemination terminates when $E_{v,o} = \emptyset$.

Message histories are implemented using Bloom filters [25]. Since updates are disseminated over relatively small sets of nodes (41 nodes on average in our dataset), the overhead is small: Bloom filters incur around k bytes of overhead for a neighborhood of size k at a false positive rate of 1%.

B. *Selection Heuristics*

We consider three strategies for node v to send update o.

Random. In the RANDOM heuristic, node v selects a node from $E_{v,o}$ uniformly at random as in Demers' [13].

Anticentrality. The ANTICENTRALITY heuristic assigns to nodes in $E_{v,o}$ a selection probability which is inversely proportional to their degree in G_v, effectively "steering selection away" from nodes with higher degree as per the reasoning of Section V. We use G_v because what matters is the degree in the context of the neighborhood where update dissemination occurs, not the degree in the overall social network.

Fig. 5a shows how the algorithm which assigns selection probabilities in ANTICENTRALITY works in the particular

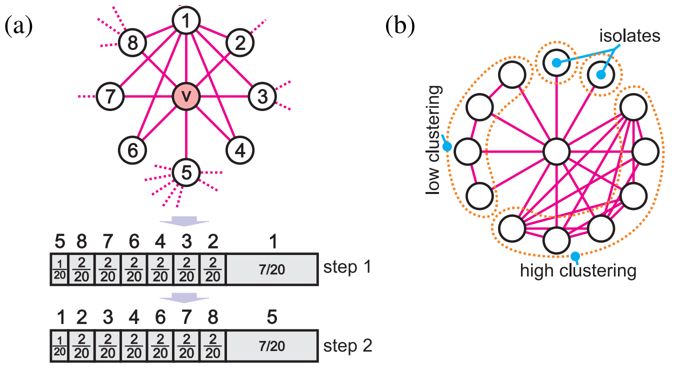


Fig. 5. (a) Probability assignment in ANTICENTRALITY. (b) Fragmented neighborhood.

case when v starts to disseminate an update over its own neighborhood. We first compute the sum of the degrees into G_v of all of v's neighbors (20). We then sort the neighbors by those degree values, and assign them proportional probabilities (step 1). Finally, we invert the assignments so that lower degree nodes get the high probabilities, and vice-versa (step 2).

Formally, let $d_v(w) = |f(w) \cap f(v)|$ (the degree of w into v's neighborhood), and $E_{o,v} = \{w_1, \cdots, w_n\}$. Suppose without loss of generality that the w_i are ordered such that $d_v(w_1) \leq \cdots \leq d_v(w_n)$. Then the probability $P_v(X = w_i)$ with which node v selects node $w_i \in E_{o,v}$ at some given round is expressed by:

$$P_v(X = w_i) = \frac{d_v(w_{n-i+1})}{\sum_{k=1}^{n} d_v(w_k)}$$

Fragmentation-aware heuristics. The *fragmentation* $\tau(v)$ of a node v is the number of connected components that remain in G_v if v is removed. Formally, let G_v^* be the subgraph obtained by removing v and all its links from G_v; let $C(G_v^*)$ be the set of connected components in G_v^*. Then, $\tau(G_v^*) = |C(G_v^*)|$.

Fig. 6 shows scatterplots of fragmentation vs. node degree in our dataset. Fragmentation varies widely at all neighborhood sizes, and larger neighborhoods tend to be proportionally less fragmented than smaller neighborhoods.

Fragmentation is an important structural metric for two reasons. First, $\tau(G_u) - 1$ represents the minimum number of messages that the node u at the center of a neighborhood (i.e. the profile page owner) must send if an update is to reach all neighbors, regardless of who published it. Second, it provides us with a simple way of improving latency, by noticing that u should hit as many different components as possible, avoiding selecting nodes inside of the same component more than once before all components have been hit.

This suggests two new heuristics, RANDCOMP and MAXCOMP, to be applied to the profile owner only. To diffuse an update o such that $prof(o) = u$, node u does the following:

1) if a component $C_i \in C(G_u^*)$ for which no node in C_i has yet received the update exists, then select a node $w \in C_i$ using ANTICENTRALITY;
2) otherwise, simply select $w \in f(u)$ using ANTICENTRALITY, completely ignoring the component structure.

The two heuristics differ in the way components are selected: RANDCOMP selects a random one, while MAXCOMP selects the largest among the candidate components.

Note that computing the actual connected components is inexpensive. We assume from Sec. II that a node u knows

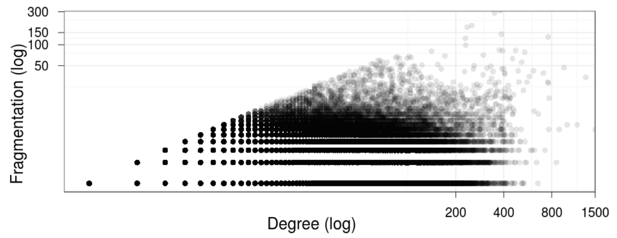


Fig. 6. Fragmentation τ for our network crawl.

both $f(u)$ and $f_2(u)$. This information is all u needs to locally reconstruct its 1-hop neighborhood graph, which is enough to compute its connected components.

A similar reasoning applies under churn. If we rely on the assumption made in Section II that a node u knows, with reasonable accuracy, which of its 1-hop social neighbors are on-line at any given point in time, then u is able to locally estimate the shape of its 1-hop neighborhood by excluding the nodes it thinks are off-line, and (locally) compute the connected components based on the estimate instead.

VI. EVALUATION

In this section we evaluate our protocols, first under the assumption of a static network, and then under churn.

A. Static Network

Here, we assume 100% availability. Under this assumption, all protocols yield residue zero for practical purposes, so we do not discuss it further.

Baseline. We use *direct mailing* [13] as the baseline protocol in our comparisons. In direct mailing, the node posting the update is responsible for contacting receivers directly. As we discuss in Section VII, this simple technique is used in some P2P OSNs. To enable comparison, we consider a round-based variant in which contacts are performed in rounds, one after the other. Although direct mailing could be run "in parallel", this would be equivalent to setting its round length to zero, which is something we can also do for our protocols. This transformation, therefore, incurs no loss of generality.

Progressive plots. Due to the nature of our experiments, to effects of the crawling procedure (e.g., border nodes all have degree 1), and to social networks themselves [24], displaying whole-network results is misleading as lower-degree nodes heavily bias figures like average latency (e.g., experiments rooted in a node with one neighbor always have average latency 1, regardless of the strategy). We therefore choose to plot these results in a way we call "progressive trimming".

We focus on the degree of the root node, and sweep through the set of values one at a time, in our case over the $[1, 1500]$ interval, the degree range of our crawl. Let $\delta : V \to \mathbb{R}$ be the function providing such value for a given node. For each value k, we compute the aggregate statistics (e.g. average latency) over the set of unit experiments $e \in E$ for which $\delta(root(e)) \geq k$, where E is the set of all unit experiments. We call such plot a *progressive plot*. Note that the value we would get for computing the statistics over the whole network corresponds to the first point in these graphs.

Graphs like this must be generated with care. For the statistics we compute, they work as if we were progressively removing terms from a weighted average, so we must make sure we are not removing the "relevant" terms too soon. The point is we *know* that the bias induced by lower size neighborhoods decreases the relevance of results, both because those are *much* more numerous, and because they do not afford much variability. As an example, think again about average latency: neighborhoods of size 1, 2, or 3 are likely to show

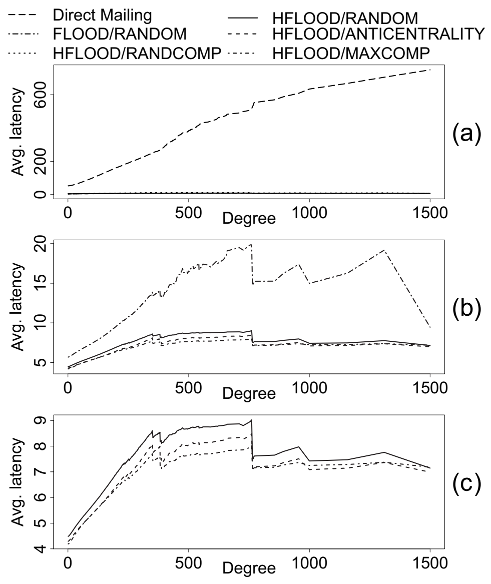


Fig. 7. Latency.

very similar average latencies. Yet, their numerosity brings the average value down, making it similar over all protocols. If we show average latency by progressively excluding lower degree neighborhoods, however, the differences between the protocols become more evident.

Latency. Fig. 7a shows the progressive plot of the average latency t_{avg} of direct mailing compared to our variants over HFLOOD. t_{avg} values for direct mailing can be computed analytically: for a neighborhood of size n, we have $t_{avg} = \frac{n-1}{2}$. This means direct mailing does not really scale, and that can be seen from the plot: t_{avg} values grow very large, while for our protocols they seem to remain almost constant in comparison.

Note that the reason why we do not see a straight line for direct mailing in the plot is that we are doing a progressive plot by degree, not simply plotting t_{avg} by degree. The shape of the plots is heavily influenced by the skewed degree distribution of the network, and that is why the curve is irregular.

Fig. 7b shows zoomed plots for HFLOOD variants and FLOOD. We can clearly see the performance benefits of using histories: HFLOOD variants are faster than FLOOD across the entire plot, with latencies being up to three times smaller at some points of the graph (i.e. for some subsets of nodes). This is mainly due to the "anti-starvation" effect of histories we described in Sec. V.

As for the HFLOOD variants, the graph also shows the effectiveness of sorting components by size (MAXCOMP) when compared to picking them at random (RANDCOMP). Not only MAXCOMP performs better than RANDCOMP, but the latter actually performs *worse* than non-fragmentation-aware heuristics such as RANDOM.

Note that these graphs are rather "bumpy". This is a result of fragmentation: since it is the main (but not the only) bottleneck for diffusion speed in our protocols, average latency tends to correlate highly to it, particularly in those heuristics which are not fragmentation-aware. That is, in fact, the reason why bumps are located roughly at the same points in the graphs, and why the graphs for the best performing protocols seem to resemble "flattened versions" of the worst-performing ones.

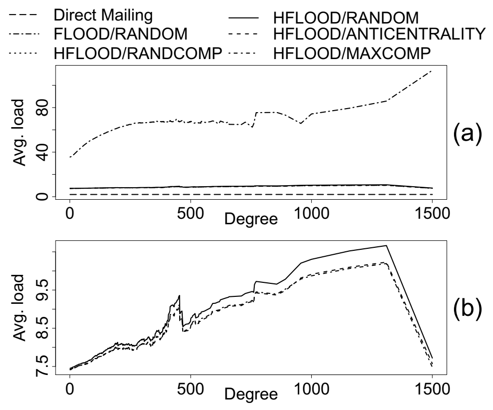


Fig. 8. Average load.

Finally, Fig. 7c zooms into the three best-performing HFLOOD variations, and confirms both the effectiveness of ANTICENTRALITY—which performs better than RANDOM across the board—and MAXCOMP, which performs better than ANTICENTRALITY, except at the end. The reason, again, is fragmentation. From Fig. 6 we can see that the large neighborhoods which get included at end of the graph are not very fragmented, in which case MAXCOMP performance tends to align with that of ANTICENTRALITY.

Load. Fig. 8a and 8b show the progressive plot for average load over all nodes, as we trim by degree. The use of histories provide large savings in load, with HFLOOD generating on average 4.8 times less messages than FLOOD. Direct mailing is the cheapest of all protocols, since it generates no duplicates: for a neighborhood of size n, each node processes on average $\frac{2n}{n+1}$ messages. As for our approaches, there is no significant difference in load values, with MAXCOMP and ANTICENTRALITY performing slightly better than RANDOM.

An interesting point is that average load seems to grow with the size of the neighborhood. To get a closer look, we do a scatterplot in Fig. 9a of average node load as a function of degree. Each point represents how much a node v pays, on average, whenever an update emerges over $f^*(v)$. We can see that the load indeed grows fast. Fortunately, this increase comes with the size of neighborhoods, not of the network. Further, even at its maximum, the value is not too high, particularly if we consider the rate at which users post updates on OSN sites. The user with the most posts in Twitter, according to the Twitaholic website [26], posts one tweet a minute; the average user is likely to post much less. An average Facebook user posts 3 pieces of content *a day* [27].

Direct mailing is economical, but if we look at how many messages each update originator must actually push into the network, on average, to disseminate its update, we get the scatterplot in Fig. 9b (shown for MAXCOMP). This brings us to another nice property of our protocol: it balances the load among those interested in receiving an update, shifting it away from the poster. Fig. 9c shows a scatterplot of the coefficients of variation (Sec. III-C) for messages sent and received by HFLOOD with MAXCOMP, as well as for direct mailing (where the coefficients of variation for messages sent and received are the same). Our approach clearly provides better balance.

Finally, we show in Fig. 10 the progressive plots of how many message copies, on average, a single post entails. Note that this gives us a detailed account of how much redundancy our protocols generate: since direct mailing produces zero duplicates, its curve serves as a reference as well. We generate, on average, 3.79 times more traffic than direct mailing, but this value can become as high as 6.8 for some neighborhoods; if direct mailing is implemented in its naïve form. If we were to implement it by using the point-to-point, DHT routing primitive provided in Graffi et al. [6], for example, this would change. While the overhead of our protocol depends on single neighborhoods and should remain stable as the system grows, the overhead for routing over a DHT grows with the size of the network, even if slowly. If we assume 100 million on-line users ($\frac{1}{5}^{th}$ of the Facebook userbase) and a Pastry DHT with settings as in [28], then $\log_{16} 10^8 = 6.64$, and we would get the curve marked as "Direct Mailing/DHT" in the graph, in which the savings afforded by direct mailing disappear.

Our solution, therefore, incurs acceptable overhead and performance, effectively enabling the use of social overlays as dissemination media in static networks.

B. Impact of Churn

We evaluated HFLOOD with MAXCOMP and direct mailing under a simple churn model wherein we associate an on/off, discrete time stochastic process $Z_v(t)$ to each node $v \in V$, such that $Z_v(t) = 1$ if node v is alive at time t, or 0 otherwise. These processes can be modelled as a two-state Markov chain, with transition probabilities given by $p_{1,1} = p_{on}$, $p_{0,0} = p_{off}$, $p_{1,0} = 1 - p_{on}$, and $p_{0,1} = 1 - p_{off}$, where $p_{i,j}$ is the probability that the chain transitions into state j at time $t+1$ given that it was at state i at time t. Let X_v^{on} be the random variable defining the session length for a node v, then:

$$\mathbb{E}(X_v^{on}) = \sum_{i=1}^{\infty} i p_{on}^i (1 - p_{on}) = \frac{1}{1 - p_{on}} - 1$$

We can similarly define and derive X_v^{off} and $\mathbb{E}(X_v^{off})$. We evaluate protocols under different p_{on} values, so as to get average session lengths of 0.5, 2, 4, and 6 hours. For the inter-session lengths we use a single p_{off} value for a 1 hour average. Asymptotic availability settings are similar to those used by Yao et. al. [29], from where our churn model derives. We use 1 second as the round duration for all protocols.

Churn introduces a new problem. Recall from Sec. V that a node v stops disseminating an update o only when it knows that all of its neighbors received it. If churn is pessimistic, it might be that v has to wait for a very long time (possibly forever) for such condition to be satisfied. To remedy this situation, we introduce a *timeout parameter* t_{out} in our protocol: if a node v cannot contact any node that has not yet received o for more than t_{out} seconds, it stops disseminating o. For the purposes of our simulations, we use a fixed t_{out} value of 30.

Since churn simulations are much more expensive, we had to constrain our evaluation to a single unit experiment per node. We are still evaluating 72,303 unit experiments for each combination of parameters and protocols.

Residue. We argue that, contrary to intuition, residue is not a really important metric for comparing our push protocols, even under churn. The reason is that session lengths under the churn model are so large when compared to the average experiment duration that, for the purpose of a single unit experiment—focusing on a single update dissemination—it is as if the network remained static. Since over a static network the number of nodes reachable from the update originator is the same regardless of the protocol, the residue is also the same. This is confirmed in the progressive residue plot of Fig. 11a: although graphs become noisy at larger degree values as we trim out more and more unit experiments from the average residue computation, the overall point stands, with values almost converging up until around 400.

A way to quantify if the dynamism left in the network actually produces a measurable impact on residue is by "correcting" it. We define the *corrected residue* of a unit experiment e associated to update o to be the percentage of nodes *with uptime larger than zero over* e which did not receive o by the time e finishes. Fig. 11b shows a progressive plot of corrected residues for HFLOOD and direct mailing under the various churn settings we tried. HFLOOD produces

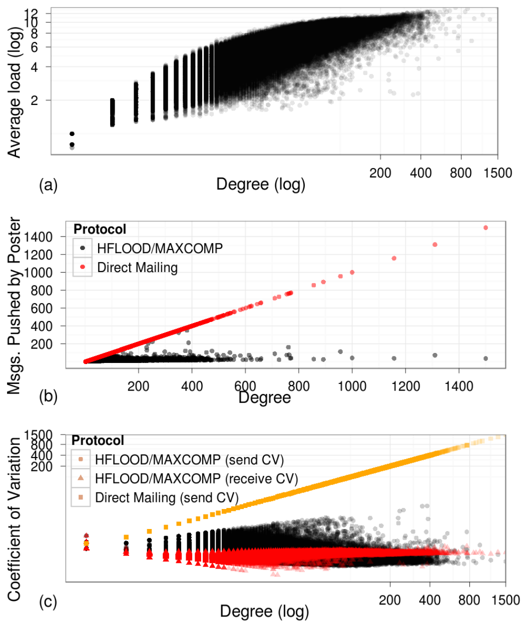


Fig. 9. **(a)** average load, **(b)** messages posted by $root(e)$, **(c)** coefficient of variation for loads within experiments.

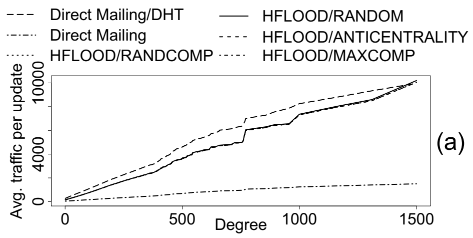


Fig. 10. Average number of messages generated per update.

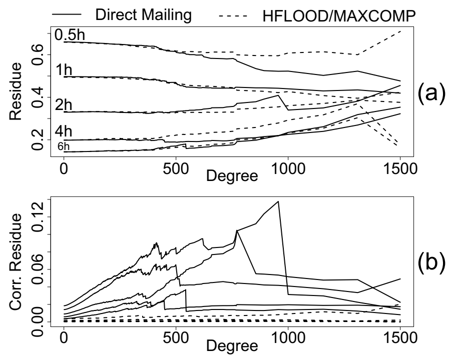


Fig. 11. **(a)** residue and **(b)** *corrected residue* for HFLOOD and direct mailing.

less corrected residue than direct mailing *over all settings*. Other churn and timeout settings are likely to exacerbate these effects, and that is something we intend to investigate next.

Latency and Load. Churn transforms the underlying social network by removing nodes from it. Therefore, we need to assess whether these transformed networks create adverse effects, be it on load (by causing certain nodes to grow in importance) or latency (by increasing the lengths of dissemination paths). Fig. 12a shows progressive plots for latency over all average session length settings, trimmed by degree, comparing HFLOOD and direct mailing. The overall point of this graph is: churn is effectively shrinking the neighborhoods, and this translates into a considerable improvement to direct mailing, but our protocol remains faster.

We zoom into the progressive latency plots for our protocol in Fig. 12b, and compare its latency figures with what we had before churn. Even if the initial averages under churn are smaller, numbers grow larger at some points, likely due to some unlucky or unforeseen structural changes. Performance remains however generally consistent, with increases only at the lowest availability levels.

Finally, Fig. 12c, shows progressive plots for load for direct mailing and our protocol, as well as for our protocol without churn. The curves for our protocol are similar, resembling translated and slightly flattened versions of the static load curve on the top. By and large, the roles played by the nodes in dissemination in terms of relative importance over their neighborhoods remain similar, with curves flattening as average uptime plummets. This is to be expected: the lower the availability, the less nodes in the network, and the more fragmented the remaining neighborhoods become. As fragmentation increases, our protocol smoothly converges into direct mailing, both in terms of latency and load.

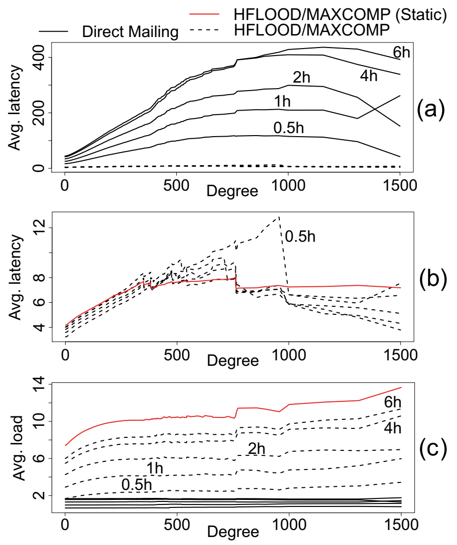


Fig. 12. **(a), (b)** average latency and **(c)** load of HFLOOD and direct mailing.

VII. Related Work

P2P OSNs and publish/subscribe. The few current proposals for P2P OSNs [5]–[7] assume that nodes are mapped onto individual users. Content exchange among friends requires that the nodes under their control are able to message each other over the network. Whenever multicast communication is required (e.g., for profile updates), these proposals must resort to *direct mailing* [13], where a node unicasts individually each destination. As we show in Sec. VI, our protocols provide significant advantages over direct mailing, and hence could find broader application in existing proposals as well.

Dissemination of updates can be seen as a pub/sub problem wherein users are publishers and their friends, subscribers. An alternative to using social overlays would then be the use of topic-connected overlays [14]: while not respecting social constraints in general, these overlays would at least allow information to be disseminated without leaving the circle of subscribers (i.e., updates can be disseminated from a user to its friends using only those friends). Unfortunately, building these overlays in a decentralized fashion is still an open problem. Recent solutions [14], [15] still suffer from limitations that are significant in our context (e.g., the inability to guarantee a single topic-connected overlay connecting all neighbors)

Quasar [16] is an overlay-independent pub/sub system for social networks. Nodes use attenuated Bloom filters to create “gravity wells” for topics of interest. Publishing amounts to performing parallel random walks on the overlay, with walkers being “pulled” into and then “expelled” from gravity wells. It relates to our approach in that random walks and push gossip protocols are similar, and because it piggybacks histories of previously visited subscribers into walkers. However, the latter is done in Quasar to avoid loops, while we do it to increase efficiency under differing clustering conditions.

GoDisco [17] is a hierarchical topic-based pub/sub system which exploits social communities to route messages and, as in our approach, uses social overlays. The authors share our view on their potential benefits, but are most interested in homophily: nodes with similar interests tend to cluster, which means that social overlays might provide an efficient dissemination medium for their pub/sub system. Like our approach, GoDisco embodies topology awareness in its routing protocol: it uses social triads [18] to counter duplicates. We instead rely on message histories, which we believe provides more robustness, and embody different topological awareness techniques to speed up dissemination.

Gossip protocols. The ANTICENTRALITY selection heuristic we put forth on Section III-B is similar to directional gossip [19], particularly in that nodes with reduced connectivity are given priority. Apart from the details of computing weights, the main difference w.r.t. our approach is that whereas directional gossip assigns a "thresholding weight" above which it switches from gossip to flooding, our heuristic biases selection based on such weights instead. The rationale is that if all nodes have similar characteristics our protocol behaves like uniform gossip; otherwise it adapts, subject to the particular topological conditions of the neighborhood in which it disseminates.

Our protocols can be related to *biased gossip* approaches like BEEP [20]. Like our approach, BEEP favours the selection of nodes on a per-user, per-news, and per-dissemination-hop basis. The inputs and rationale for biasing are however different: BEEP heuristically disseminates news to nodes that might find them interesting, while adapting the fan-out so that popular news spread and unpopular one die. Our biasing is instead geared towards reducing duplicates and latency, and compensating for the non-uniformity of social graphs.

Finally, while our protocols share their general operation with traditional gossip [13], they are particular in that nodes are only allowed to talk to friends. This constrains the protocol to a set of arbitrary, richly varying graphs—the social neighborhoods. General reliability results [12] are then not expected to hold and, given the complexity and variability of these graphs, extensive empirical evaluation is required instead.

VIII. Conclusions and Future Work

This paper has discussed the use of push-based gossip protocols for the dissemination of updates over social networks. The problem is of interest not only because it fits our vision for a P2P OSN based on *social overlays*, but also because it might find broader application into existing P2P OSN proposals which, up until now, relied either explicitly or implicitly on the use of direct mailing and DHTs.

We have shown the caveats of applying gossip protocols to social networks by quantitatively demonstrating the extent to which classical protocols such as Demers' rumor mongering become inefficient under their widely non-uniform clustering characteristics. We then introduced a novel gossip protocol capable of adapting to (and leveraging off) such non-uniformity. This protocol is based on three key principles: the use of *message histories*, an *anticentrality* selection heuristic, and *fragmentation awareness*. We have shown through simulations that these principles yield benefits, given that our protocol significantly improves over mainstream gossip protocols and direct mailing. Finally, we have shown that our protocol performs acceptably under various churn conditions.

Future work involves an improved evaluation of the protocols, including more realistic workloads, an evaluation of the anti-entropy mechanism, and an integrated analysis encompassing overlay maintenance, as well as the implementation of a prototype and an evaluation of the protocols under a distributed testbed (e.g., PlanetLab).

References

[1] J. Rothschild, "High Performance at Massive Scale – Lessons learned at Facebook," http://cns.ucsd.edu/lecturearchive09.shtml#Roth.

[2] D. Boyd and E. Hargittai, "Facebook privacy settings: Who cares?" *First Monday*, vol. 15, no. 8, 2010.

[3] D. Boyd and N. B. Ellison, "Social network sites: Definition, history, and scholarship," *Jrnl. of Computer-Mediated Comm.*, vol. 13, 2007.

[4] F. Benevenuto, T. Rodrigues, M. Cha, and V. Almeida, "Characterizing user behavior in online social networks," in *Proc. Internet Measurement Conf. (IMC'09)*, 2009.

[5] S. Buchegger, D. Schiöberg, L. hung Vu, and A. Datta, "PeerSoN: P2P social networking – early experiences and insights," *Proc. Workshop on Social Network Systems (SNS'09)*, 2009.

[6] K. Graffi, C. Gross, P. Mukherjee, A. Kovacevic, and R. Steinmetz, "LifeSocial.KOM: a P2P-Based platform for secure online social networks," in *Proc. Conf. P2P Computing (P2P'10)*, 2010.

[7] L. A. Cutillo, R. Molva, and T. Strufe, "Safebook: A privacy-preserving online social network leveraging on real-life trust," *IEEE Communications*, vol. 47, no. 12, pp. 94–101, Dec. 2009.

[8] W. Galuba, "Friend-to-Friend computing: Building the social web at the internet edges," EPFL, Tech. Rep. LSIR-REPORT-2009-003, 2009.

[9] D. Watts and S. Strogatz, "Collective dynamics of small-world networks," *Nature*, vol. 393, no. 6684, pp. 440–442, 1998.

[10] M. McPherson, L. Smith-Lovin, and J. M. Cook, "Birds of a feather: Homophily in social networks," *Annual Review of Sociology*, vol. 27, no. 1, pp. 415–444, 2001.

[11] K. Graffi, P. Mukherjee, B. Menges, D. Hartung, A. Kovacevic, and R. Steinmetz, "Practical security in p2p-based social networks," in *Proc. Conf. Local Computer Networks (LCN'09)*, 2009.

[12] A.-M. Kermarrec, L. Massouli, and A. J. Ganesh, "Probabilistic reliable dissemination in large-scale systems," *IEEE Trans. Parallel Distributed Computing*, vol. 14, no. 3, pp. 248–258, 2003.

[13] A. Demers *et al.*, "Epidemic algorithms for replicated database maintenance," in *Proc. Intl. Conf. on Principles of Distributed Comp.*, 1987.

[14] F. Rahimian, S. Girdzijauskas, A. Hossein Payberah, and S. Haridi, "Vitis: A gossip-based hybrid overlay for internet-scale publish/subscribe enabling rendezvous routing in unstructured overlay networks," in *Proc. Intl. Parallel & Distributed Processing Symp. (IPDPS'11)*, 2011.

[15] S. Girdzijauskas, G. Chockler, Y. Vigfusson, Y. Tock, and R. Melamed, "Magnet: practical subscription clustering for internet-scale publish/subscribe," in *Proc. Conf. Distributed Event-Based Sys.*, 2010.

[16] B. Wong and S. Guha, "Quasar: A probabilistic publish-subscribe system for social networks," in *Proc. Intl. Workshop on P2P Sys.*, 2008.

[17] A. Datta and R. Sharma, "Godisco: Selective gossip based dissemination of information in social community based overlays," in *Proc. Intl. Conf. Distributed Computing and Networking (ICDCN'11)*, 2011.

[18] A. Rapoport, "Spread of information through a population with sociostructural bias: I. assumption of transitivity." *Bulletin of Mathematical Biophysics 15*, pp. 523–533, 1953.

[19] M.-J. Lin and K. Marzullo, "Directional gossip: Gossip in a wide area network," in *Proc. European Dependable Computing Conf.*, 1999.

[20] A. Boutet, D. Frey, R. Guerraoui, and A.-M. Kermarrec, "WhatsUp: news from, for, through everyone," in *Proc. Conf. P2P Computing*, 2010.

[21] P. Saint-Andre, "Extensible Messaging and Presence Protocol (XMPP): Core," RFC 6120, IETF, Mar. 2011. [Online]. Available: http://www.ietf.org/rfc/rfc6120.txt

[22] A. Montresor and M. Jelasity, "Peersim: A scalable p2p simulator," in *Proc. Conf. P2P Computing (P2P'09)*, 2009.

[23] S. H. Lee, P. Kim, and H. Jeong, "Statistical properties of sampled networks," *Phys. Rev. E*, vol. 73, no. 1, pp. 1–7, Jan. 2006. [Online]. Available: http://link.aps.org/doi/10.1103/PhysRevE.73.016102

[24] A. Sala *et al.*, "Brief announcement: Revisiting the power-law degree distribution for social graph analysis," in *Proc. Conf. on Principles of Distributed Computing (PODC'10)*, 2010.

[25] A. Broder and M. Mitzenmacher, "Network applications of bloom filters: A survey," *Internet Mathematics*, vol. 1, pp. 636—646, 2002.

[26] "Twitaholic website." http://www.twitaholic.com, [visited Apr 2011].

[27] Facebook Inc., "Facebook statistics," http://www.facebook.com/press/info.php?statistics, [visited Apr 2011].

[28] A. Rowstron and P. Druschel, "Pastry: Scalable, decentralized object location, and routing for Large-Scale Peer-to-Peer systems," in *Proc. Conf. on Middleware (Middleware'01)*, 2001.

[29] Z. Yao, D. Leonard, X. Wang, and D. Loguinov, "Modeling heterogeneous user churn and local resilience of unstructured P2P networks," in *Proc. Intl. Conf. Network Protocols (ICNP'06)*, 2006.