Cloud-assisted Dissemination in Social Overlays

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

***Abstract*—Decentralized social networks are an emerging solution to the privacy issues plaguing mainstream centralized architectures. *Social overlays*—overlay networks mirroring the social relationships among node owners—are particularly intriguing, as they limit communication within one's friend circle. Previous work investigated efficient protocols for P2P dissemination in social overlays, but also showed that the churn induced by users, combined with the topology constraints posed by these overlays, may yield unacceptable latency. In this paper, we combine P2P dissemination on the social overlay with occasional access to the cloud. When updates from a friend are not received for a long time, the cloud serves as an external channel to verify their presence. The outcome is disseminated in a P2P fashion, quenching cloud access from other nodes and speeding dissemination of existing updates. We show that our protocol performs close to centralized architectures and incurs only modest monetary costs.**

I. Introduction

Online social networks (OSNs) play a key role in the way we communicate over the Internet, having attracted billions of users worldwide. Unfortunately, this popularity is accompanied by concerns about privacy [3]. The root of the problem lies in the centralized architecture of mainstream OSNs, which requires users to surrender control of sensitive and personal data. As a consequence, interest in decentralized alternatives [8] has greatly increased in the last few years.

In this context, peer-to-peer (P2P) OSNs appear as an interesting option, as they allow decentralization to be achieved while allowing end users to participate with existing resources. P2P has, however, two main drawbacks: *i)* open membership and decentralization creates important privacy and security issues [17]; and *ii)* unpredictable and skewed peer availability [19] leads to performance and reliability issues. In [13], we have advocated the use of *social overlays* (SOs)—overlay networks that *mirror* and underlying social network—as a way to either solve or greatly mitigate the former. We proposed a simple system that could provide arguably the most important functionality of OSNs: the ability to browse profile pages of friends, and post updates to them. Timely *dissemination of profile updates to friends* emerged as the key problem to be solved. Such proposal is in stark contrast with most of the literature on P2P OSNs (e.g. [4], [5], [12]), which relies on distributed hash tables (DHTs) for dissemination and storage.

Yet, SOs had limitations of their own: indeed, we show in [14] that SOs cannot cope well with peer churn, mostly because they cannot be reconfigured to compensate for missing nodes, leading to transient partitions and update dissemination times larger than 3 hours for 1% of the receivers—unacceptable for a world-scale service like today's Facebook.

The problem of any purely decentralized dissemination protocol for SOs is that the latter does not provide enough "options" to propagate updates in the presence of churn. Our idea is simple and yet effective: create an out-of-band channel that can "patch" connectivity if and when needed, by leveraging the persistence and ubiquity of cloud services. Our scheme works as follows. The bulk of update dissemination is still carried out in a P2P fashion on the SO. Specifically, we reuse the gossip-based protocol described in [13]. In addition, each node u is associated to a *profile store* hosted on the cloud. Updates to u's profile are first written to the store, then disseminated over the SO. Therefore, the profile store of u contains an always-available, consistent copy of u's profile.

The availability of the profile store is key in overcoming the aforementioned delays in the propagation of profile updates. When a node v, friend of u, has not heard any update from u for a predefined time interval, it assumes that the update has been delayed, and verifies if this is the case by polling the profile store. In principle, this naïve solution is enough to overcome the limitations above. However, cloud access has a monetary cost, and we show that this solution has a poor performance/money tradeoff. We improve over this baseline by disseminating the outcome of polling the profile store back on the SO. This has the beneficial effect of quenching cloud accesses from other nodes (i.e., saving money) and speeding update dissemination. Results show that, compared to fully decentralized solutions, our hybrid one reduces maximum delays from hours to minutes, average delays from minutes to seconds, and incurs only a small cost to users.

While our approach inevitably reveals some information to the cloud provider—a limitation shared with other proposals [6]—we argue that it is still a significant departure from the *status quo* of centralized systems, in that the service agreement of the latter implies that user information is fully surrendered, while the service of cloud providers promises data confidentiality. Data is in any case stored encrypted, providing an extra level of safety against prying eyes.

II. System Model

We represent the social overlay as an undirected graph $G = (V_G, E_G)$. We define the *ego network* G_v of a node $v \in V_G$ as the subgraph of G composed of v, the friends of v, and the edges among them. To simplify exposition, we use the words "user" and "node" interchangeably, assuming there is a one-to-one mapping between nodes and users. Every user u is associated to a unique profile page $\mathbf{pp}(u)$ that might contain textual posts, pictures, comments from friends, among others. Such pieces of content are usually small objects under $150\ kB$, the size of a Facebook picture. While users also share larger objects such as videos, we argue that most of them are not part

978-1-4799-0521-8/13/$31.00 ©2013 IEEE

of profile pages themselves, but rather *linked* from third-party services such as YouTube or Flickr.

Two nodes are able to communicate as long as they are online at the same time. We further do not model network transmission latencies since, as we will later see, these are relatively small w.r.t. to other sources of delay.

We assume clocks to be loosely synchronized, within the order of minutes. This can be easily achieved by NTP.

At any point in time, the nodes of the P2P network are either logged in (online) or out (offline). To drive this online/offline behavior, we adopt the well-known availability model of Yao et al. [19]. In the particular instance of the model we adopt, both session (online time between a login and the next logout) and inter-session lengths (offline time between a logout and the next login) are exponentially distributed, with averages of 0.5 and 1.0 hours, respectively. In line with other works in the literature [16], we use exponential instead of heavy-tailed distributions because the latter would make simulations intractable. Heterogeneity is modelled as in [19].

We assume the existence of a highly available cloud service which nodes can access to store and retrieve data, allowing users to create personal storage areas under their control, i.e, they can selectively allow/deny read/write access to others.

User profiles are small (a few GB) and likely within the free quota currently allowed by some cloud providers. However, to better elucidate the trade-off between delay and monetary cost, we adopt the *requester-pays* billing scheme of Amazon S3 [2], where users accessing data are charged for it. This establishes the basis for the fair cost model of our solution: a user might opt to either go directly to the cloud and pay to immediately download updates from his friends, or use the free P2P network instead, possibly at a performance penalty.

In S3, costs can be broken down into three components: storage, bandwidth and request. Storage is normally cheap: at the time of writing, the yearly cost for 1 GB is around $1. Since storage costs do not impact our figures to a measurable extent, we choose to abstract them away altogether. Similar considerations hold for bandwidth: updates are small and, as shown later, our protocol works by favoring many lightweight requests over fewer, larger requests. The cost per request, however, cannot be ignored. In S3, a user pays $0.01¢$ for every $10,000$ GET requests, or every $1,000$ PUT requests. The cost per read is the dominant in our case, and is therefore the one we focus on in this paper.

III. Problem Statement

The problem of *update dissemination* has been described in [13] and consists in diffusing copies of small profile updates posted by a *sender* node to a set of *receiver nodes* over an SO, with an acceptable delay. In particular, let G_u be the ego network of some node u. We want to be able to disseminate updates from a sender node $v \in V(G_u)$ to all other receivers in $V(G_u)$. Note that it could be that $v = u$, i.e., the sender is posting an update to its own profile, or $v \neq u$, e.g. when v replies to a previous post in the profile of u. Further, to reap the benefits of friend-to-friend cooperation, avoid spamming uninterested nodes, and avoid leaking updates to nodes who are not supposed to see them, we want to disseminate this update using only nodes from G_u itself.

Disseminating updates with low latency over ego networks can be done efficiently when nodes are always online, as shown in [13]. When availability is taken into account, however, things get more difficult [14]. Absent nodes might create transient disruptions on the network, introducing *communication delays*. We consider two notions of delay. The first one is network-centric, defined as the total time elapsed from the instant t_0 at which an update is posted by a source v to the instant t_w at which it is received by a receiver w. We refer to this metric as the *end-to-end delay* from v to w w.r.t. t_0, or $\mathbf{ed}(v, w, t_0)$. It can be easily computed as $t_w - t_0$.

Being network-centric, $\mathbf{ed}$ is not representative of *user experience*. We capture the latter by measuring how long a user *had to wait online* before receiving an update. The intuition is that a receiver that logs in infrequently does not care if an update was posted long ago (i.e., with a high $\mathbf{ed}$), provided it is received shortly after login. We refer to this metric as the *receiver delay* from v to w w.r.t. t_0, or $\mathbf{rd}(v, w, t_0)$.

As we have demonstrated in our previous work [14], update dissemination over a purely P2P is not practical: average receiver delays are in the order of hours for a significant fraction (1%) of the nodes, and remains above tens of minutes for over 10% of the nodes, leading to poor user experience.

Our goal is to solve the dissemination problem while providing an acceptable user experience. Formally, we want ensure a *target delay bound* δ on the receiver delay $\mathbf{rd}$. Capping $\mathbf{rd}$, however, is not enough, as it allows some undesirable situations to emerge [15].

We therefore choose to allow a *soft delay bound* on $\mathbf{ed}$ instead. The idea is that as soon as the target delay bound δ is crossed, the system must deliver the update *at the next login of the receiver*. We express this by adding some slack time to the bound δ in case w is offline. This slack time represents the residual offline time $\mathbf{R}_{off}$ of w until its next login. Formally:

$$\mathbf{ed}(v, w, t_0) \leq \begin{cases} \delta & \text{if } w \text{ online at } t_0 + \delta \\ \delta + \mathbf{R}_{off} & \text{otherwise} \end{cases} \quad (1)$$

The slack time is a random variable, whose probability distribution results directly from the availability model. Since the slack is composed entirely of offline time, it does not count as receiver delay, so that Equation (1) implies $\mathbf{rd}(v, w, t_0) \leq \delta$ as well. The goal of this paper is providing a hybrid cloud/P2P dissemination protocol which can honor the soft latency bound of Equation (1) while being efficient and low-cost.

IV. Cloud-assisted Dissemination

The need for cloud resources arises because the social overlay cannot provide acceptable delays to all source/destination pairs in the network. This happens because the absence of certain nodes can create transient partitions that disrupt communication paths and introduce delay. We need to "patch" such partitions somehow. A simple way to do so would be associating each node u to an *alias* $\tilde{u}$ of itself in the cloud, which gets activated on-demand should u be offline.

Unfortunately this solution has several drawbacks. Both on-demand (e.g. EC2 [2]) and full-blown instances tend to be

expensive (i.e. more than $100/year). Further, we want to avoid exposing the full memory state of the running software to the cloud provider, as this might reveal sensitive information (e.g. private keys). Finally, such solution requires nodes on the critical path of transient partitions to use and pay for an alias—not a feasible strategy in general.

The aforementioned problems led us to an alternative solution where $\tilde{u}$ is no longer a full clone of u, but rather a simple high-availability *profile store* in which $\mathbf{pp}(u)$ is kept.

To publish an update to to $\mathbf{pp}(u)$, a user v simply writes it directly to $\tilde{u}$. Access control can be easily handled by primitives provided by services such as Amazon's S3 [2]. By adopting the requester-pays model of S3, we ensure that each user has complete control over costs. To minimize exposure of sensitive data to the cloud provider, all data stored in $\tilde{u}$ is encrypted. This is in stark contrast with centralized solutions such as Facebook, whose business model effectively precludes storing encrypted data from being acceptable practice.

However, differently from cloud aliases, profile stores are passive in that they cannot initiate interactions with other nodes. Therefore, to actively overcome the transient partitions that cause delays, we resort to periodic *polling*.

Naïve approach. In its simplest form, our protocol works by having each node $w \in V(G_u)$ independently poll $\tilde{u}$ every δ time instants, retrieving any updates to $\mathbf{pp}(u)$ posted in the meantime. If w happens to be offline after δ time instants have passed, then w accesses the cloud immediately once it logs back in. This simple protocol, which we refer to as PUREPOLL, allows us to circumvent the transient partitions through an out-of-band channel, satisfying Equation (1).

Hybrid approach. While simple, PUREPOLL is wasteful in that it disregards the existence of low-delay paths in the social overlay which could be used to our advantage. To understand how, note that if we were to discard all paths in the social overlay that have delays above or close to the delay bound δ we wish to maintain we would be left with a set of disjoint groups for which the *internal* delays are low. We call these *delay groups*. To get a message disseminated over a set of delay groups while respecting the delay bound δ, all we have to do is to ensure that *at least one* node in each of these groups actually accesses the cloud every δ time instants.

The second method we propose, named HYBRID, does just that. Each node $w \in V(G_u)$ keeps track of the last instant in time at which it heard any news from u. Whenever w goes for more than δ time instants without hearing from u, it polls the profile store of u to see if there are updates. If so, w downloads them from the cloud and pushes them into the social overlay by means of the push gossip protocol of [13]. Otherwise, w pushes a special QUENCH message that contains the time t_0 at which w accessed the cloud and found nothing new. This message serves to inform other nodes that, as of t_0, there are no new updates, and that they can therefore refrain themselves from accessing the cloud for an extra δ time instants. We call this mechanism *access quenching*.

To see how HYBRID approximates the ideal situation of dissemination over delay groups described previously, note that if two nodes belong to the same delay group then one will likely hear from the access of the other, resulting in quenching. If two nodes do not belong to the same delay group, instead, it is unlikely that they hear each other in time to promote quenching, and two separate cloud accesses will ensue. Therefore, the protocol adjusts to the delay characteristics of the surrounding network, and provides a self-organizing mechanism for bridging transient partitions by polling.

Randomizing cloud accesses. A side effect of the protocol we described is that it induces nodes belonging to the same delay group to synchronize their accesses to the profile store.

We address the problem by scattering access times near $t_0 + \delta$ as follows. First, we divide δ into a *fixed* component ψ and a *random* component α, such that $\delta = \psi + \alpha$. Then, let $\mathbf{T}$ be a random variable drawn from a uniform distribution $\mathcal{U}(0, \alpha)$. When node w_1 accesses the cloud and, later, when w_2 receives the QUENCH message, we now set their access times to $t_0 + \psi + \mathbf{T}$. This causes w_1 and w_2 to have slightly different access times after $t_0 + \psi$ which, provided α is sufficiently large, is enough to make quenching effective. Since $0 \leq \mathbf{T} \leq \alpha$, this implies $\psi + \mathbf{T} \leq \delta$, i.e., the modified protocol still honours the soft bounds of Equation (1).

V. EVALUATION

We evaluate our protocol towards two goals: 1) providing supporting evidence that it performs significantly better than a pure P2P approach, and it is competitive with centralized approaches; 2) estimating its monetary and network costs.

A. Baselines

We compare our protocol against three baselines: *i)* PUREPOLL, which is the naïve protocol of Section IV; *ii)* PUREP2P, which disseminates updates exclusively over the social overlay with no cloud assistance; and *iii)* SERVER, which emulates a centralized approach akin to Facebook. This is achieved by adding a special server node which is connected to all the others, manages all profile pages, and can relay updates with zero delay. This represents the best performance reference for our system. Clearly, we wish to perform as close as possible to SERVER, while incurring only modest monetary costs.

B. Experimental Setting

Protocols are evaluated over a sample of 700 ego networks picked uniformly at random from the Orkut crawl of Mislove et al. [1]. This is the same sample we used in [14], and a detailed description can be found there. For each sample, we pick *one* source node $v \in V(G_u)$ uniformly at random and we repeat the following experiment 100 times. First, we run the simulation for a *burn-in period*, in which no measurement is taken. After burn-in, the experiment progresses by waiting for the first login of the source v, at which point we cause v to post an update to $\mathbf{pp}(u)$. This marks the beginning of our *measurement session*, which proceeds until the update reaches all destinations in $V(G_u)/\{v\}$. The simulation then ends.

C. Metrics

Delay. In this paper, we focus on *average* end-to-end and receiver delays, which we refer to as $\mathbf{aed}$ and $\mathbf{ard}$, respectively. We compute estimates for each source/destination pair (v, w).

Monetary cost. We estimate yearly running costs for each node $w \in V(G_u)$, denoted $\mathbf{ycs}(w,u)$ by first estimating, from simulation, the average number of accesses-per-hour performed by w while keeping up-to-date with u's profile and then multiplying it by the number of hours in a year, and then by the price of a get request.

Finally, to get an estimate of the overall yearly spendings $\widehat{\mathbf{cs}}(w)$ of w, we would need to compute $\sum_{m \in V(G_w)} \mathbf{ycs}(w,m)$. In practice this means computing estimates for thousands of ego networks, which is intractable given the high costs of these simulations. We therefore choose to use two distinct approximations for $\widehat{\mathbf{cs}}(w)$ instead: *i)* a *flat approximation*, in which we multiply $\mathbf{ycs}(w,u)$ by the average size of the ego networks in our sample, and *ii)* a *degree approximation*, in which we multiply $\mathbf{ycs}(w,u)$ by the degree of w in the social network. The first approximation works by applying a fixed penalty to all nodes, while the second assumes that costs increase linearly with the number of friends. Given our fixed computational budget, these cost models represent a tradeoff we had to make by favouring accurate estimates for delay at the expense of accurate estimates for cost.

Network cost. To assess the usage of network resources, we estimate from simulation the number of messages a node w has to process per time unit while keeping up-to-date (and helping disseminate) updates from her friend u. We then apply the same flat and degree approximations to obtain overall costs (i.e. how many messages a node has to process to help and keep up with all friends), with the same caveats as before.

D. Results

We use the notation HYBRID/ψ/α to refer to the variant of HYBRID with parameters ψ and α, with unit given in minutes, and the notation PUREPOLL/δ, to refer to the PUREPOLL variant with target delay bound δ.

We simulate two versions of HYBRID, HYBRID/30/14, and HYBRID/15/14. These parameter settings, as we later show, provide good performance in terms of delay and cost. However, as explained in Section IV, the target delay bound of HYBRID is randomized, and varies in $[\psi, \psi+\alpha]$ with average $\psi + \alpha/2$. Since PUREPOLL is not randomized, we compare each setting of HYBRID to three settings of PUREPOLL: *i) "fast"*, PUREPOLL/ψ; *ii) "intermediate"*, PUREPOLL/$(\psi+\alpha/2)$; *iii) "slow"*, PUREPOLL/$(\psi+\alpha)$.

This yields six PUREPOLL variants: three ($\delta \in \{30, 37, 44\}$) to compare against HYBRID/30/14 and three ($\delta \in \{15, 22, 29\}$) to compare against HYBRID/15/14. Since PUREPOLL/30 and PUREPOLL/29 behave essentially the same, we do not show the former and use the latter. Finally, to understand at which point PUREPOLL can overtake HYBRID, we add a seventh setting for PUREPOLL in which we set $\delta = 5$.

Delay. Figure 1 shows cumulative distribution functions (CDFs) for receiver and end-to-end delays of HYBRID and baselines. Since **rd** is always zero for SERVER, we omit it from the plot. Complementary statistics are given in Table I.

The data confirms the results of [14]. PUREP2P suffers from significant performance issues and its **ard** distribution has long tail, with a maximum of 2.9 hours, and a 99^{th} percentile of 1.5h—unacceptable for a Facebook-scale system. Further, the data shows that our cloud-based alternatives—HYBRID and PUREPOLL—effectively *solve the problem of the long delay tail* by putting a bound on **rd**, which can be seen from the much smaller maximum and 99^{th} percentiles.

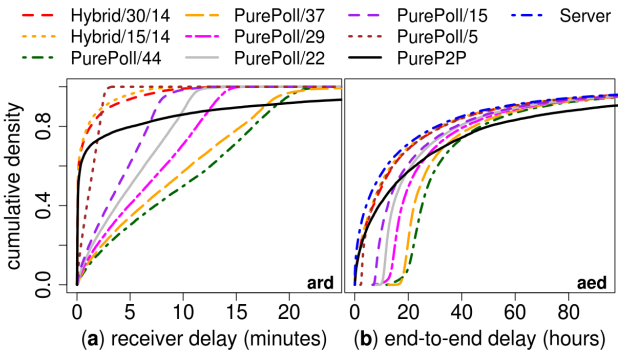

Fig. 1. CDFs for **aed** and **ard** for all source/destination pairs.

Finally, we see that HYBRID outperforms its associated PUREPOLL variants while providing a better experience for most users across all parameter settings, even as we compare HYBRID/30/14 to PUREPOLL/5. By combining both approaches, HYBRID effectively reconciles the best of both worlds: the fast performance of PUREP2P for the regions of the network that exhibit low delay—which can be seen in Figure 1a as the nearly vertical shape of the CDF up until the 60^{th} percentile—with the ability of mitigating the long delay tails of PUREPOLL. Table I shows that HYBRID has maximum and 99^{th} percentile **ard** values which are comparable to those of their *fast* PUREPOLL counterparts, with HYBRID being faster. Indeed, HYBRID/30/14 and HYBRID/15/14 perform around 837% and 685% percent faster, on average, than PUREPOLL/29 and PUREPOLL/15, respectively.

Figure 1b shows that HYBRID significantly improves end-to-end delays as well, particularly at lower percentiles: the polling period, never smaller than δ, is a barrier to PUREPOLL but not for HYBRID. Further, HYBRID *is much closer to* SERVER *than* PUREP2P.

Monetary cost. Yearly cost figures are provided in Table II. We use the current Amazon S3 pricing [2], as per the model of Section II. Again, the costs for HYBRID are generally lower than their associated PUREPOLL variants. Costs are attractive under the flat model, with HYBRID/15/14 presenting good cost/latency tradeoff. The other side of the coin is given by the

	ard		
	avg.	99^{th}	**max.**
PUREP2P	5.76m	1.5h	2.9h
HYBRID/30/14	48s	9.8m	15.2m
HYBRID/15/14	35s	7.2m	13.5m
PUREPOLL/44	9.94m	22.9m	27.1m
PUREPOLL/37	8.9m	21.6m	26.4m
PUREPOLL/29	6.7m	14.5m	16.6m
PUREPOLL/22	5.2m	11.9m	14.3m
PUREPOLL/15	4m	10.7m	12.3m
PUREPOLL/5	1.2m	2.8m	4.5m
SERVER	0	0	0

TABLE I
DISSEMINATION DELAY (m = MINUTE, s = SECOND, d = DAY).

	flat (USD)			degree (USD)		
	avg.	99^{th}	**max.**	**avg.**	99^{th}	**max.**
HYBRID/30/14	1	2.9	3.4	1.25	9	230
HYBRID/15/14	1.42	3.9	4.66	1.72	12.4	304
PUREPOLL/44	1.84	2.38	2.9	2.5	17.4	282
PUREPOLL/37	2.13	2.8	3.47	2.9	20.3	324
PUREPOLL/29	2.63	3.59	3.81	3.57	25	390
PUREPOLL/22	3.32	4.72	4.88	4.51	31.6	483
PUREPOLL/15	5	9.5	20	9.5	48	737
PUREPOLL/5	13	27	64	17	128	1909

TABLE II
YEARLY COSTS, IN US DOLLARS.

	avg.	90^{th}	95^{th}	99^{th}	**max.**
node degree	197.5	409	658	2 897	33 313
HYBRID/30/14 (degree)	25.98	43.17	77.9	298.16	9 498
HYBRID/15/14 (degree)	26.02	43.17	77.6	294.73	9 472
HYBRID/30/14 (flat)	11.64	32.60	46	73.2	178
HYBRID/15/14 (flat)	11.8	33	56	74.8	168

TABLE III
NETWORK COST, IN QUENCH MESSAGES PER SECOND.

degree model, which reaches high maximum values. Such high values, however, all originate from a small set of extremely connected nodes, the most connected having 33 313 friends (more details in [15]). For nodes with less than 1 000 friends, however, costs for HYBRID/30/7 and HYBRID/15/7 are no larger than $15 and $21 a year, respectively (i.e. cheaper than hosting solutions such as EC2 [2] or low-cost hosting [9]).

Even if some users do get a large number of friends, we still do not expect to see these high costs in practice: users with huge ego networks are likely to trim friends they do not want to keep in touch so often, bringing down costs considerably.

Network cost. We focus here on the propagation of QUENCH messages and its impact on the underlying P2P network, for two reasons: *i)* costs incurred by updates relate to user posting frequencies and habits, which are variables beyond our control; *ii)* bandwidth available for actual updates is ultimately given by whatever is available, minus QUENCH overhead, which we measure here. Cost statistics are are presented in Table III.

Assuming a QUENCH message has a size of around 48b (40b for a TCP/IP header plus 8b for identifier and timestamp), costs are reasonable for a large fraction of the nodes: 90% process less than 40 msgs/s $\sim$ 2 kB/s, even under the degree approximation. Maximum values, however, are clearly too high, again due to high degree nodes (above 1 000 friends). We argue, however, that such nodes are unlikely to fare well in any decentralized OSN architecture we know of.

VI. RELATED WORK

A number of P2P OSNs have been proposed recently [4], [5], [12]. Most of these are directly based on DHTs and, as such, are different from our work. Safebook [5], however, uses an SO implicitly to anonymize transmissions. Since SOs are inherently partition-prone, the techniques we use here could be of use in rendering it practical. A similar observation holds for other SO-based proposals [7], [10], [18].

The idea improving P2P performance with cloud resources is not new. Cloud helpers are proposed in [11] to increase availability in a friend-to-friend (F2F) backup system. In backup, however, high availability is relevant to the data owner only. Further, objects are larger and time requirements are less stringent, shifting concerns to throughput instead of latency. Confidant [6] directly targets OSNs by relying on cloud aliases. Data is kept only at the P2P layer, replicated only among friends, while aliases act as coordinators tracking replicas and membership (i.e., who replicates what, who is online). In this setting, churn becomes a challenge to the availability of data, instead of its dissemination.

VII. CONCLUSIONS AND FUTURE WORK

Social overlays are an interesting option for building decentralized OSNs, but their susceptibility to transient partitioning under churn renders key functionality such as fast dissemination of profile updates difficult to implement efficiently. In this paper, we have presented a solution to these inefficiencies, by introducing a protocol that leverages a highly available cloud infrastructure to adaptively support the overlay when and where needed, without sacrificing the fundamental property of allowing communication only among friends. We show that delays can be dramatically improved and monetary costs limited for users with less than one thousand friends. However, network costs, albeit acceptable, can be optimized further. We intend to address this issue by investigating a combination of the current push approach with an anti-entropy mechanism.

REFERENCES

[1] A. Mislove et al. Measurement and analysis of online social networks. In *Proc. Internet Measurement Conf.*, 2007.
[2] Amazon AWS website. http://aws.amazon.com/, [August 2013].
[3] D. Boyd and E. Hargittai. Facebook privacy settings: Who cares? *First Monday*, 15(8), 2010.
[4] S. Buchegger, D. Schiöberg, L.-H. Vu, and A. Datta. PeerSoN: P2P social networking – early experiences and insights. *Proc. Workshop on Social Network Systems (SNS'09)*, 2009.
[5] L. A. Cutillo, R. Molva, and T. Strufe. Safebook: A privacy-preserving online social network leveraging on real-life trust. *IEEE Communications*, 47(12):94–101, Dec. 2009.
[6] D. Liu et al. Confidant: Protecting OSN Data without Locking it Up. In *Proc. Conf. Middleware*, 2011.
[7] A. Datta and R. Sharma. GoDisco: Selective gossip based dissemination of information in social community based overlays. In *Proc. of Intl. Conf. Distributed Computing and Networking (ICDCN'11)*, 2011.
[8] Diaspora website. https://joindiaspora.com/, [March 2013].
[9] Dreamhost website. http://dreamhost.com/, [March 2013].
[10] Freenet website. http://www.freenet.org, [March 2013].
[11] R. Gracia-Tinedo, M. Sanchez-Arigas, and P. Garcia-Lopez. F2box: Cloudifying F2F storage systems with high availability correlation. In *Proc. IEEE Conf. on Cloud Computing*, 2012.
[12] 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.
[13] G. Mega, A. Montresor, and G. P. Picco. Efficient dissemination in decentralized social networks. In *Proc. Conf. P2P Computing*, 2011.
[14] G. Mega, A. Montresor, and G. P. Picco. On churn and communication delays in social overlays. In *Proc. Conf. P2P Computing*, 2012.
[15] G. Mega, A. Montresor, and G. P. Picco. Cloud-assisted dissemination in social overlays. Technical Report DISI-13-031, Univ. of Trento, 2013.
[16] A. Singh, G. Urdaneta, M. van Steen, and R. Vitenberg. Robust overlays for privacy-preserving data dissemination over a social graph. In *Proc. Intl. Conf. Dist. Computing Sys. (ICDCS)*, 2012.
[17] T. Paul et al. Exploring decentralization dimensions of social networking services: Adversaries and availability. In *Proc. Workshop on Hot Topics in Interdisciplinary Social Networks Research*, 2012.
[18] Tonika website. http://5ttt.org, [March 2013].
[19] Z. Yao et al. Modeling heterogeneous user churn and local resilience of unstructured P2P networks. In *Proc. of ICNP'06*, 2006.