On Churn and Communication Delays 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*—Peer-to-peer systems based on an overlay network that *mirrors* the social relationships among the nodes' owners are increasingly attracting interest. Yet, the churn induced by the availability of users raises the question—still unanswered—of whether these *social overlays* represent a viable solution. Indeed, although constraining communication to take place only among "friends" brings many benefits, it also introduces significant limitations when healing the overlay in the presence of churn.**

This paper puts forth two contributions. First, we show through simulation on real datasets that churn induces relevant delays in information dissemination, which may ultimately hamper the practical application of social overlays. Yet, identifying opportunities for improvement and evaluating design alternatives through simulation is impractical, due to the size of the target networks, the large parameter space, and the many sources of randomness involved. Therefore, in our second contribution we combine analytical and simulation techniques to enable the estimation of dissemination delays at a practical cost.

I. Introduction

Social networks are complex networks that arise from the interaction patterns of social beings. Apart from possessing a number of well-known structural properties [18], these networks present unique opportunities for system developers to build computer systems that are more secure, robust, and privacy-friendly [3]. In this study, we are interested in a particular instantiation of this opportunity, in which users are "represented" in the network by their computers or personal devices, and communication links are restricted, by design, to pairs of devices whose owners are friends. Given that the current Internet is not organized in this way, such a network must necessarily be deployed as an *overlay network*, which we refer to as a *social overlay* (SO).

Social overlays. The idea of assembling a computer network whose links mirror those of an existing social network is not new. Indeed, social overlays have increasingly found their way into research proposals and applications over the last few years, ranging from censorship-resistant storage systems [2], to new ways of structuring the Internet [3]. The use of a social overlay implies a decentralized, peer-to-peer architecture: not surprisingly, the most widely-known, SO-based system in use, Freenet [2], is commonly ascribed to the class of P2P systems.

Social overlays have recently been proposed as a substrate for information dissemination. In [8] this is realized as a form of publish-subscribe over the social overlay. Instead, the work in [13] proposes an architecture for decentralized online social networks where the dissemination of information (e.g., profile updates) from a user to its corresponding *ego network* (i.e., the portion of the SO including the user, her friends, and the social links among them) is realized solely atop the social overlay.

By restricting communication to friends, social overlays are expected to improve, *by design*, privacy (non-friends do not see the information disseminated), locality (due to network homophily), and cooperation (due to friendship).

Churn and communication delays. An issue hitherto unexplored, however, is the one of *churn*. Like any other P2P system, those based on social overlays must cope with the fact that users are allowed to join and leave the network of their own volition, according to availability patterns known to be highly heterogeneous [20]. This is a well-studied problem in P2P systems. For instance, common "synthetic" overlays (e.g. [16]) deal with joins and leaves by automatically rewiring the connections among neighbors. In performing this self-healing process, however, they are free to choose how to establish connectivity between the remaining nodes: constraints are imposed only for optimizing system parameters, or respecting certain overlay invariants. Social overlays do not enjoy the same luxury, as their defining property is that connectivity is allowed only among 1-hop friends. Breaking this constraint would break their essence. However, this very same constraint severely reduces the "opportunities for healing" in the presence of churn, and may render the system unreliable and inefficient.

For instance, suppose some information (e.g., a profile update) is injected in the system by a peer u, to be disseminated to all of u's friends, including a friend v. Due to their usage habits, however, it so happens that u and v are never online at the same time. To relay the message to v, the system would have to find a third peer w to broker the message, such that w is online at the same time as u, and then later at the same time as v. Further, w must be a friend of both u and v. Clearly, a peer w satisfying these conditions might not exist, causing the message to traverse a chain of friends acting as brokers before it can actually reach v—assuming it ever does. Moreover, the longer this chain, the greater the *delay* experienced by the receiver v, as one broker must wait for the next one to come online before passing the message forward. If this delay is too large, the applicability of the solution becomes limited.

In this study, we focus on the notion of *delay* as the metric determining the practical usability of the system. Intuitively, if a message from a user takes "too long" to be delivered to any of her friends, the system may become unusable for practical purposes. In Section II we define precisely our system model, while in Section III we formally define the problem and the metrics—end-to-end delay and receiver delay—we use.

Goals and contributions. In this context, the research questions at the heart of this paper are two:

1) *Does the constraint of using only "friend links" limit the applicability of social overlays?*
2) *If so, how can we mitigate the effects of churn on social overlays?*

We investigate the answer to these questions by putting forth two contributions.

First, we analyze and compare a pure SO-based solution against a mainstream server-based solution (e.g., representative of those used by online social networks), using simulation from real datasets combined with a well-known availability model [20]. As we show in Section IV, 1% of the receivers experience a delay greater than 3 hours. This result may appear good, and it is for applications (e.g., data storage [2]) that are not sensitive to delay. Also, 1% may seem at first a small percentage. However, when considering large-scale, interactive applications such as Facebook or Twitter, this result would translate into an unacceptable user experience for millions of users. Since the culprit is churn, we also compare against a solution where a variable fraction of the nodes composing the SO are assumed to be fixed. This allows us to glance at opportunities for improvement that could be easily harvested by, say, assuming that some of the peers are willing to rely on cloud computing services to provide a continuous online presence. Note that this would still preserve many of the desirable properties achieved by SOs, by placing the decision about where to put the data into the hands of the users, rather than of the online social network providers (e.g., Facebook).

We argue that the simulation study above is a valuable indication that indeed churn *is* a concern for social overlays, therefore partially answering question 1. Nevertheless, a more articulate answer, as well as the answer to question 2, both require a level of analysis that can hardly be supported by a simulation study alone, due to *i)* the sheer size of the networks to be considered when working with social network datasets, *ii)* the large parameter space inherent to modeling availability, e.g., the many ways in which it can be mapped on network nodes; and *iii)* the many sources of randomness involved, with heavy-tailed distributions increasing the number of repetitions required to obtain meaningful results.

Therefore, our second contribution, detailed in Section V, revolves around an analytical model of dissemination over social overlays. Specifically, we present:

1) a hybrid analytical/simulation framework which enables us to determine, at a practical cost, an upper bound to dissemination delays;
2) algorithms for identifying key graph substructures that, as we show, are responsible for a substantial fraction of the observed delays.

The paper is completed by a concise overview of related works in Section VI, and by final remarks in Section VII.

II. System Model

Definitions and notation. We represent a social network as an undirected graph $G = (V, E)$ where V represents users and E is the friend relationship between them. For each user $u \in V$, the function $f : V \to 2^V$ maps users to their set of friends. The *ego network* $G_u = (\{f(u) \cup u\}, E_u)$ of a user u is defined as the subgraph of G induced by u, its friends $f(u)$, and the set of interconnections E_u among them. We assume without loss of generality that there is a one-to-one mapping between users and nodes in the system.

A. Availability Model

Our system is a P2P network with $|V|$ participants, where each user u can be either logged in (online) or out (offline) at a given time instant t. In this context, we were faced with two options when it came to specifying online/offline behavior.

The first option we considered was to use one of the publicly-available traces from measurement studies of P2P systems (e.g., [15]) and, more recently, from instant messaging applications (e.g., [17]). Although directly applying real datasets is appealing, it is also a challenge in itself in our context, since: *i) size*: the social graphs required for our experiments are much larger than the networks captured by availability traces, and it is unclear how to employ the latter to simulate the former without biasing the results; *ii) duration*: even the longest traces available are too short to accommodate a sufficient number of decorrelated experiments; *iii) noise*: most traces contain high rates of permanent departures (e.g., due to peer aliasing [15]), which affect the statistical properties of peer sessions, and in general introduce unacceptable bias.

We therefore chose a second option, namely, the use of a well-known synthetic churn model, hereafter referred to as the Yao model [20]. This model is derived from the statistical behavior of measurement studies in the context of P2P systems, and therefore provides us with a good approximation of real peer behavior, while avoiding the aforementioned problems associated with the use of the actual real data.

The Yao model associates an *alternating renewal process* to each node $u \in V$. In these processes, both session lengths (online time between a login and the next logout) and intersession lengths (offline time between a logout and the next login) of a node u are given by random variables $\mathbf{X}^u_{on}$ and $\mathbf{X}^u_{off}$, drawn from node-specific distributions F^u_{on} and F^u_{off}.

Although the analysis tools we later develop are independent of these distributions, we consider in this paper one of the instantiations of the model proposed by [20], in which both F^u_{on} and F^u_{off} are shifted Pareto distributions $\mathsf{pareto}[\alpha, \beta]$ with parameters α and β and average $\beta/(\alpha - 1)$. Their cumulative distribution function (CDF) is given by:

$$\mathsf{pareto}[\alpha, \beta](x) = 1 - (1 + x/\beta)^{-\alpha}, x > 0, \alpha > 1$$

This distribution is heavy-tailed, being known for accurately modeling the skewed availability patterns characteristic of P2P systems [15]. Let the average session and intersession lengths for node u be $\ell^u_{on} = \mathbb{E}(\mathbf{X}^u_{on})$ and $\ell^u_{off} = \mathbb{E}(\mathbf{X}^u_{off})$, respectively. Then, the *asymptotic availability* of u is [20]:

$$a_u = \lim_{t \to \infty} \mathbb{P}(u \text{ is on-line at time } t) = \frac{\ell^u_{on}}{\ell^u_{on} + \ell^u_{off}} \quad (1)$$

B. Heterogeneity

Heterogeneous user availabilities are modelled once again as in Yao et al. [20], by drawing the ℓ_{on} and ℓ_{off} shown in Eq. (1) from yet another pair of shifted Pareto distributions. For session lengths, we use $\mathsf{pareto}[3,1]$ yielding $\mathbb{E}(\ell_{on}) = 0.5$ hours. For inter-session lengths we use $\mathsf{pareto}[3,2]$, yielding $\mathbb{E}(\ell_{off}) = 1$ hour. These averages, again, coincide with those of measurement studies found in the literature [14].

With those in place, we set a session length distribution for each node u to be $F^u_{on} = \mathsf{pareto}[3, 2 \cdot \ell_{on}]$ and the inter-session length distribution $F_{off} = \mathsf{pareto}[3, 2 \cdot \ell_{off}]$. Note that, under those settings, $\mathbb{E}(\mathbf{X}_{on}) = \ell_{on}$ and $\mathbb{E}(\mathbf{X}_{off}) = \ell_{off}$.

Each node $u \in V$ is then associated to a pair of distributions (F^u_{on}, F^u_{off}). We call the set A_G of all of these pairs, one per node, the *availability assignment of graph* G.

III. Problem Statement

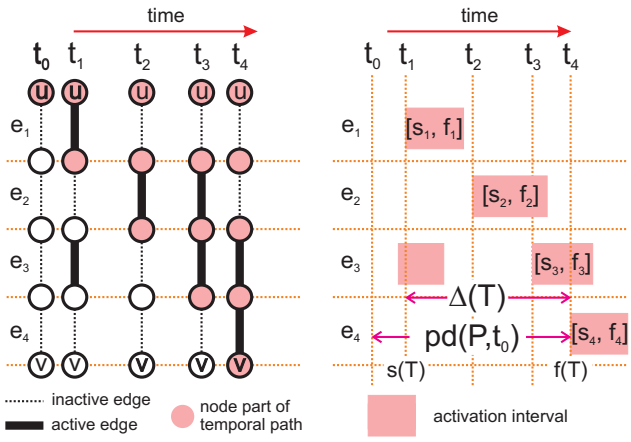
Given an arbitrary connected graph $G = (V, E)$ and its availability assignment A_G, we want to understand how long it takes for a message to propagate between an arbitrary pair of vertices u, v. A lower bound for such *communication delay* can be obtained by studying the *temporal connectivity* of G under A_G. The problem is deeply connected to the study of *temporal paths*, defined next. Note that this problem formulation is very general, as G can represent an entire social network just as well as some subgraph of it, such as an ego network.

We say that an edge $e = (u, v) \in E$ is *active* when both u and v are online at the same time. Each edge e is associated to a infinite set $I(e)$ of disjoint *activation intervals*, which represent the periods of time in which e is active. An interval $[s, f] \in I(e)$ starts at time s and finishes at time f.

Let $P = \{e_1, \cdots, e_n\}$ be a path between u and v in G. We say that a *temporal path* $T = (P, S)$ exists between nodes u and v if and only if there exists a sequence of intervals $S = [s_1, f_2], [s_2, f_2], \ldots, [s_n, f_n]$ such that $[s_i, f_i] \in I(e_i)$ $(1 \leq i \leq n)$ and $s_i \leq f_{i+1}$ $(1 \leq i < n)$. In other words, a temporal path only exists if there is a sequence of activation intervals for which the edges of P are activated one after the other, allowing a message to be propagated from u to v. The idea is illustrated in Fig. 1a: as the edges become active in sequence, a message can flow from node u towards node v. We say that the temporal path T *materializes* the underlying path P or, equivalently, that T is a *materialization* of P.

For a given temporal path T, we define $s(T) = s_1$ and $f(T) = s_n$ as the time instants at which the first and last edges of T get activated, respectively. The *duration* $\Delta(T)$ of a temporal path can then be defined as $f(T) - s(T)$. The idea is illustrated in Fig. 1b, which shows the activation intervals of Fig. 1a as they develop in time. The duration of the temporal path in the diagram is given by $\Delta(T) = t_4 - t_1$.

Note that many temporal paths form between u and v over time, but we are only interested in those that form after u has actually something it wants to transmit to v. The *earliest* time instant when this may happen is when u logs in. Therefore, we choose to (pessimistically) measure the communication delay w.r.t. the login events of u, rather than the duration of the temporal paths connecting u and v. This is equivalent to assuming that a node u sends a message as soon as it logs in.



(a) Materialization. (b) Corresponding time diagram.

Fig. 1. Two views of a temporal path.

The idea is illustrated again in Fig. 1b, where $\mathbf{pd}(P, t_0)$ represents the communication delay along the temporal path materializing P, measured w.r.t. the login event of u at t_0 instead of the beginning of the first activation interval at t_1. The communication delay, *in this case*, is given by $t_4 - t_0$.

Formally, let t_0 be a time instant at which u has logged in. For a given path $P \in G$ between u and v, let $\mathcal{M}(P, t_0)$ represent the set of all temporal paths T that materialize P, such that $s(T) > t_0$. We define the *path delay* of P w.r.t. t_0 as:

$$\mathbf{pd}(P, t_0) = \min\{f(T) - t_0 \colon T \in \mathcal{M}(P, t_0)\}$$

This notion captures the communication delay among u and v over a single path. In general, however, if several temporal paths materialize between u and v, the path that actually carries the message sent from u is the one that forms first. Based on this intuition, we can now define the *end-to-end delay* $\mathbf{ed}(u, v, t_0)$ among two nodes u and v w.r.t. t_0 as the smallest path delay, also w.r.t. t_0, over the set $\mathcal{P}(u, v)$ of all paths connecting u and v in G:

$$\mathbf{ed}(u, v, t_0) = \min\{\mathbf{pd}(P, t_0) \colon P \in \mathcal{P}(u, v)\}$$

For a given set of time instants $L = \{t_0, \cdots, t_n\}$ representing logins of u, we can compute the *average end-to-end delay* w.r.t. v as:

$$\mathbf{aed}(u, v, L) = \frac{1}{|L|} \sum_{t_i \in L} \mathbf{ed}(u, v, t_i) \tag{2}$$

The end-to-end delay is, by nature, a network-level metric. From an application and user perspective, however, what really matters is the delay *experienced by the receiver* of a message. For instance, a receiver that logs in very infrequently would not really care whether a message was sent a long time ago, as long as it is received shortly after login.

We therefore define a second, application-level metric based on the fraction of time that the receiver spends online, i.e., with a chance of getting the message. Let $\mathsf{up}(v, t) : V \times \mathbb{R} \to \mathbb{R}$ be

the function yielding the *total uptime* of node $v \in V$, i.e., the accrued time v spent online until some time instant t. Now, let t_0 be an instant when a sender u logs in. We define the *receiver delay* $\mathbf{rd}(u, v, t_0)$ between u and v w.r.t. t_0 as the total time that v spends online between the sending and the receipt of the message sent by u, i.e., in the interval $[t_0, t_0 + \mathbf{ed}(u, v, t_0)]$. Using the up function, we can express this as:

$$\mathbf{rd}(u, v, t_0) = \mathsf{up}(v, t_0 + \mathbf{ed}(u, v, t_0)) - \mathsf{up}(v, t_0) \quad (3)$$

Similarly to end-to-end delay, we can define an *average receiver delay* w.r.t. a set of login events $L = \{t_0, \cdots, t_n\}$ of node u as:

$$\mathbf{ard}(u, v, L) = \frac{1}{|L|} \sum_{t_i \in L} \mathbf{rd}(u, v, t_i)$$

The two metrics, $\mathbf{ed}$ and $\mathbf{rd}$, are related. By recalling from Eq. (1) that the asymptotic availability a_v of node v expresses the average fraction of time that v spends online, we can approximate $\mathbf{rd}$ as:

$$\mathbf{rd}(u, v, t_0) \sim a_v \cdot \mathbf{ed}(u, v, t_0) \quad (4)$$

The practical impact of this relation is that the analytical model in Section V can focus on end-to-end delay, as receiver delay can be derived from it. Before illustrating our model, however, we turn our attention to determining whether churn poses a threat to the practical application of social overlays.

IV. Can Social Overlays Support Communication Under Churn?

A simple approach to answering this question is by means of plain, discrete-event simulations. If we simulate the churn model and collect enough $\mathbf{ed}$ and $\mathbf{rd}$ values, we can obtain $\mathbf{aed}$ and $\mathbf{ard}$ values that are "close enough" to the "population averages" for these values. These simple simulations entail simulating all events in all nodes in the graph. We call them *full simulations*, in contrast with the smaller-scale simulations we exploit in Section V.

We now describe in detail how we designed these simulations, along with our specific experimental setting. Although the latter is limited by the relatively small scale of the experiment and the specific choice of parameters, the results point at the fact that a pure P2P social overlay fails to perform acceptably under churn. This calls for a more comprehensive study, for which full simulations do not represent an adequate tool, motivating the contribution put forth in Section V.

A. Simulation Design

To simplify the discussion, we focus this subsection on the simulations for the average end-to-end delay $\mathbf{aed}$. Later, we discuss briefly how to use the very same method to directly obtain values for the average receiver delay $\mathbf{ard}$.

Estimating $\mathbf{aed}$ by full simulations amounts to running the churn model in a discrete event simulator and directly observing a large number of $\mathbf{ed}$ values. The way this simulation works is described in Algorithm 1. Initially, we set all nodes to "logged out" (line 3). We then remove the bias introduced by this initial state by running the churn simulation for a *burn-in* time δ_b (line 10) until the number of nodes logged in stabilizes. We have empirically established $\delta_b = 48$ hours to be sufficient for most practical cases, where we observe stable-state parameters similar to what described in [20].

Algorithm 1: FullSimulation

Input: Graph $G = (V, E)$, number of repetitions n, burn-in time δ_b, source node u.

```
1  forall v ∈ V do aed[v] ← 0                          % avg. comm. delay
2  for i ← 1 to n do
3      U ← ∅                                           % nodes currently logged in
4      R ← ∅                                           % nodes reached from u so far
5      forall v ∈ V do ed[v] ← ⊥                       % comm. delay
6      while R ≠ V do
7          e ← nextSimulationEvent()
8          if e.type = LOGIN then U ← U ∪ {e.node}
9                            else U ← U − {e.node}
10         if e.time ≥ δ_b and e.type = LOGIN then
11             if e.node = u and u ∉ R then
12                 R ← {u}
13                 ed[u] ← e.time
14             if u ∈ R then
15                 Reachable(G, R, U, ed, e.time)
16     for v ∈ V do
17         aed[v] ← aed[v] + (ed[v] − ed[u])
18 forall v ∈ V do aed[v] ← aed[v]/n
19 return aed
```

The simulation progresses by identifying which nodes are "reachable" from the source u. A node v becomes reachable when the first temporal path starting from the source reaches it. The first node to become reachable is the source u itself. This happens when u logs in for the first time (line 11–12).

Then, for each subsequent login event (from any node), we check whether there are any nodes that, previously unreachable from u, are now reachable, using Algorithm 2. We start a breadth-first-search (BFS) from each member v of the set of

Algorithm 2: Reachable

Input: $G = (V, E)$, R, U, ed from Alg. 1 and time t.

```
1  Q ← R ∩ U
2  while Q ≠ ∅ do
3      v ← extract(Q);                  % Remove in FIFO order
4      forall (v, w) ∈ E do
5          if w ∈ U and ed[w] = ⊥ then
6              ed[w] ← t
7              R ← R ∪ {w}
8              Q ← Q ∪ {w}
```

Metric	Value
Number of Vertices	137892
Number of Edges Edges	1460043
Avg. Clustering Coefficient	0.112
Average Egonet Size	197.5

TABLE I
STATISTICS FOR EGO NETWORK SAMPLE.

nodes $Q = R \cap U$ reachable from u and currently online. The BFS visits a node w if it is online ($w \in U$) and has not been visited before ($ed[w] = \perp$). When w is visited, $ed[w]$ is set to the current time and w is added to both Q and R.

The **ed** values generated for each experiment are our samples taken from the underlying, unknown distribution of **ed** values between u and v. These get accumulated in an array (line 16) which, at the end of the whole procedure, is divided by the number of repetitions to yield the **aed** values.

The average receiver delay **ard** can be obtained by keeping track of the accrued uptimes for each node, so that we can have an implementation of the $\mathsf{up}(v,t)$ function required by Eq. (3). When the source u logs in for the first time (line 12), we take a snapshot of the uptimes of all nodes, and store them in an auxiliary array. These are the values of $\mathsf{up}(v,t_0)$ of Eq. (3). Then, whenever a node v becomes reachable at instant t_1 we compute $\mathbf{rd}(v,t_0) = \mathsf{up}(v,t_1) - \mathsf{up}(v,t_0)$.

B. Experimental Setting

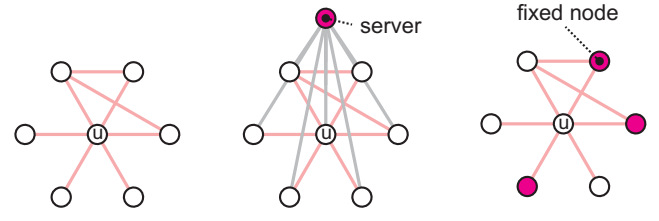
We perform full simulations over ego networks extracted from the Orkut crawl in [5]. The original graph contains 3 million vertices (3 million ego networks), 223 million (undirected) edges, and has an average clustering coefficient of 0.171. We single out 700 of these ego networks uniformly at random. This sample includes around 5% of the vertices in the graph, and its average clustering coefficient is slightly smaller, as shown in the statistics in Table I.

We chose ego networks because *i)* they allow us to focus on relatively smaller networks, for which simulations are nonetheless already expensive, and *ii)* dissemination over ego networks is often a building block in SO-based approaches, including our own [13].

For each ego network G_u in our sample, we:

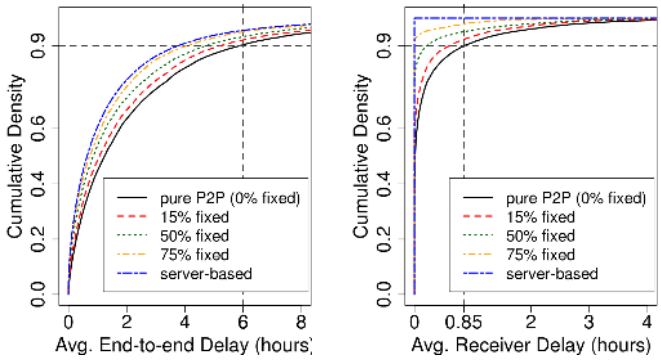
1) select a node $v \in G_u$ uniformly at random;
2) perform 100 000 full simulation runs taking v as a source;
3) estimate **aed** and **ard** from v to all other nodes in G_u.

This yields delay estimates for a total of 137 260 source/destination pairs, coming from 7×10^8 full simulation runs. The high number of repetitions – another complication of the approach based purely on simulations – is required because the underlying distribution of the **ed** and **rd** metrics is heavily skewed (possibly heavy-tailed), meaning that it is not possible to obtain an accurate approximation of the average unless we run a *very large* amount of repetitions, so that "rare events" that impact the average have the opportunity to unfold [10].



(a) Ego network G_u. (b) G_u with server node. (c) G_u with 3 fixed nodes.

Fig. 2. The three types of experiments.



(a) (b)

Fig. 3. CDFs for average end-to-end and receiver delay.

As a baseline for comparison, we adopt a hypothetical "server-based" approach which mimics current centralized solutions like Facebook. To that end, we add a special node to the graph which is connected to all other nodes, and which is also available (up) 100% of the time. This allows us to understand, for a given availability assignment, what is the best achievable end-to-end delay. Needless to say, the receiver delay under such a setting will always be equal to zero. This is illustrated in Fig. 2b, where we add the special node to the ego network depicted in Fig. 2a.

We also perform another kind of experiment, where we select a percentage of the nodes in the overlay uniformly at random and make them *fixed*; i.e., we make them always online. Note that, as illustrated in Fig. 2c, this is different from the "server" approach – we do not add any new nodes here, we change *existing* nodes so that they are always available. This allows us to understand whether and how performance numbers change as we "patch up the holes" in the network, as well as how much "patching" we need before performance becomes acceptable. Also note that, unlike the server-based approach, this experimental setting mimics *decentralized* systems such as Diaspora [1], in which a percentage of the nodes might choose to run their nodes in paid, cloud-hosting services.

C. Results and Discussion

Fig. 3 shows the cumulative distributions for the values we obtained for **aed** and **ard**, against their baselines, as well as for varying proportions of fixed nodes (15%, 50%, and 75%). Plots are truncated near the largest 90^{th} percentile, with complementary information provided in Table II.

The experiment, despite the aforementioned limitations, reveals a number of important properties and issues. The first one is the generally high values for **aed**, which can be seen all across the board. Averages are no less than 1 hour and 48 minutes, and maximum values are as high as 724

	Average		99^{th} Perc.		Maximum	
	aed	**ard**	**aed**	**ard**	**aed**	**ard**
0% **fixed**	2.36h	14 mins	17.8h	3.6h	724h	10h
15% **fixed**	2.35h	14 mins	16.1h	3.0h	719h	9.5h
50% **fixed**	1.88h	7 mins	14h	2.7h	262h	8.9h
75% **fixed**	1.61h	2 mins	12h	1.4h	167h	7h
server	1.48h	0	11.7h	0	111h	0

TABLE II
STATISTICS FOR COMMUNICATION DELAY.

hours (a month). The fact that even the server-based approach exhibits significantly high values means, however, that most of this delay is likely due to receivers that come online very infrequently. This is confirmed by the comparatively much lower values of $\mathbf{ard}$, which remain in the order of minutes, on average, and hours, in the worst case.

By examining $\mathbf{ard}$ values closely, however, a number of observations can be made. First, the maximum values are unacceptably high (10 hours for a purely P2P solution), and seem to decrease quite slowly, even as we add a large percentage of fixed nodes. Second, as Fig. 3b shows, although performance is acceptable for a large portion of the users (on the order of tens of seconds for around 50% of the users), curves flatten significantly as we approach the 90^{th} percentile, with receiver delays never lower than 1h for the slowest 1%, even with 75% of fixed nodes. The pure P2P approach performs particularly poorly, with receiver delays nearing 1 hour already at the 90^{th} percentile, and climbing to 3.6 hours at the 99^{th} percentile.

Given that this constrained experiment already reveals significant performance issues, we conclude this section by pointing out that a deeper, more comprehensive investigation is required if we are to understand under which availability conditions social overlays are viable. Yet, to base this investigation on full simulations is simply not feasible, because:

1) *Social network datasets are large.* Claiming something meaningful about social networks under churn entails studying a large number of them, which translates into prohibitively high simulation costs.
2) *Parameter choices are large.* The experiments in this section are run with a single, randomly-generated availability assignment, as described in Sec. II. Properly exploring the parameter space would entail experimenting with more of these assignments, including ones with different distribution parameters, varying assumptions of correlation/no correlation among graph-structural (e.g., node degree) and node properties (e.g., availability), among others. The costs of each assignment gets multiplied by the size of the datasets, yielding even higher simulation costs.
3) *Distributions are heavy-tailed.* Adding to the previous issues, skewed and heavy-tailed distributions both in the availability model and the observed metrics mean a large number of repetitions is required before reliable results can be obtained, further increasing the costs.

The combination of these issues ultimately renders full simulations an impractical approach for studying the problem at hand, motivating a quest for more efficient alternatives.

V. TOWARDS AN ANALYTICAL MODEL

In this section, we present our first results towards such an alternative: a partial, hybrid model which uses simple analytical results at its core, and fills in the remaining gaps with simulations. Our current model allows us to determine an upper bound for the average end-to-end delay, as well as to identify key graph substructures that, as we will show, determine a significant fraction of the observed delay values.

Given the relation between $\mathbf{ed}$ and $\mathbf{rd}$ expressed by Eq. (4) we focus our model only on the end-to-end delay $\mathbf{ed}$.

A. Formation of Temporal Paths

The key observation behind our model lies in something we briefly mentioned in Sec. II: that the only requirement for nodes u and v to be able to communicate over a graph G is that *some* temporal path forms between them – we do not really care which one. We can gain insight on how to model the end-to-end delay between two vertices u and v, then, by reasoning probabilistically about the time it takes for some temporal path T to materialize a path P between them.

Let $P = \{e_1, ..., e_m\}$ be a path between u and v in $G = (V, E)$. For each edge $e_i = (w_i, w_{i+1}) \in P$, let $\mathbf{X}_{e_i}$ be the random variable representing the time elapsed between a (randomly picked) login event of w_i and the beginning of the next activation interval of e_i. We refer to $\mathbf{X}_{e_i}$ as the *edge delay* of e_i. Given that our churn model is probabilistic, we can also represent the duration of a temporal path – i.e, the time it takes for a temporal path T to materialize P – as a random variable Δ_P. We can then express the path delay $\mathbf{D}_P$ of a path P as:

$$\mathbf{D}_P = \mathbf{X}_{e_1} + \Delta_P \tag{5}$$

Note that Eq. (5) is just a restatement of the notion of path delay $\mathbf{pd}$ laid out in Sec. III, but expressed this time as a sum of random variables. Now, we claim that:

$$\Delta_P \sim \sum_{i=2}^{m} \mathbf{X}_{e_i} \overset{(5)}{\Rightarrow} \mathbf{D}_P \sim \sum_{i=1}^{m} \mathbf{X}_{e_i} \tag{6}$$

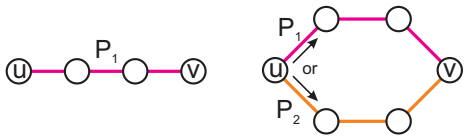
In other words, we claim that the duration of a temporal path is approximately equal to the sum of the delays of its edges, except for the first, which always contributes with zero. To understand why, think of a temporal path with a single edge, and note that it will always have duration zero, since its materialization happens instantaneously at the instant of activation of its single edge.

Substituting this claim back into Eq. (5) yields the final result of the equation, namely, that the delay of a path is approximately equal to the sum of the delay of its edges.

We are not yet able to prove whether Eq. (6) holds as an equality, but we could experimentally verify that:

$$\mathbb{E}(\mathbf{D}_P) \sim \sum_{i=1}^{m} \mathbb{E}(\mathbf{X}_{e_i}) \tag{7}$$

that is, even if there are effects (e.g., residuals) unaccounted for in Eq. (6), these are empirically negligible from the point of view of the expectation. An explanation of how



(a) (b)

Fig. 4. Two simple graphs.

we conduct this empirical validation, as well as numerical evidence supporting our assertion, is given in Sec. V-D1.

B. Bounding from Above

Given the previous results, we can provide an upper bound to the average end-to-end delay. For a pair of nodes u, v, let $\mathbf{C}_{u,v}$ be the random variable representing the end-to-end delay between them. The upper bound follows from the observation that the following holds for any path P between u and v:

$$\mathbb{E}(\mathbf{C}_{u,v}) \leq \mathbb{E}(\mathbf{D}_P)$$

This inequality states that the expected end-to-end delay among two nodes is always smaller than the expected delay of any single path connecting them. To see why this is true, it helps to think of the delay for a path P in terms of probabilities. Suppose we had a single path P_1 connecting u and v, as in Fig. 4a, and that u were to log in the system at time $t_0 = 0$. The probability that *at least one* path gets materialized between u and v by time t is thus given by the probability that the delay for this path is smaller than t, or $\mathbb{P}[\mathbf{D}_{P_1} \leq t]$.

Now, without loss of generality, assume that we had two disjoint paths P_1, P_2 connecting u and v in G instead of just one (Fig. 4b), while everything else remains the same. In this case, the probability that *at least one* path gets materialized by time t is given by $\mathbb{P}[\mathbf{D}_{P_1} \leq t \vee \mathbf{D}_{P_2} \leq t]$ (i.e., the probability that the delay of P_1 or P_2 is smaller than t), which is obviously higher than the probability of one single path being materialized by time t.

This intuition extends to $\mathbb{E}(\mathbf{C}_{u,v})$: the average end-to-end delay between nodes u and v is always less than or equal to the expected delay time $\mathbb{E}(\mathbf{D}_P)$ of *any* individual path P with equality holding if and only if P is the only path in the graph that connects u and v. This also implies that computing the "true" value of $\mathbb{E}(\mathbf{C}_{u,v})$ would require us to account for the effects of *all* paths connecting u and v.

There is one particular path, however, that is "closest" to $\mathbb{E}(\mathbf{C}_{u,v})$ than any other: the path P_{min} for which $\mathbb{E}(\mathbf{D}_{P_{min}})$ is the smallest (i.e., the "fastest path" between u and v). Using Eq. (7), we can find P_{min} through the following procedure:

1) *Assign edge weights.* Given the graph $G = (V, E)$ and its availability assignment A_G, we assign to each edge $e \in E$ a weight $h_e = \mathbb{E}(\mathbf{X}_e)$, i.e., equal to its expected delay. As we discuss later, we can compute an estimate for $\mathbb{E}(\mathbf{X}_e)$ by means of simulations which are much simpler than the full simulations of Sec. IV-A. Once we have the weights h_e for all edges in the graph, computing an estimate for the expected path delay $\mathbb{E}(\mathbf{D}_P)$ for any arbitrary path P in G becomes, by virtue of Eq. (6), a matter of summing the weights of its edges. In other words, for a path $P = \{e_1, \cdots, e_n\}$, we have:

$$\mathbb{E}(\mathbf{D}_P) \overset{(7)}{\sim} \sum_{e \in P} \mathbb{E}(\mathbf{X}_e) \sim \sum_{e \in P} h_e$$

2) *Compute shortest paths.* Using a shortest path algorithm (e.g. Dijkstra's) and the weight assignment of the previous step, we compute the minimum cost path between u and v. The resulting path, as per the previous discussion, is the *fastest path* P_{min} between u and v.

Algorithm 3: EstimateEdgeDelay

Input: An edge $e = (w_1, w_2)$; the number of samples n.

```
1  s ← 0;                                   % Sum of all samples
2  r ← 0;                                   % Total number of samples
3  L ← ∅;                                   % Sample buffer
4  on ← {false, false};                     % Online status of w1, w2
5  while r ≤ n do
6      e ← nextSimulationEvent()
7      on[e.node] ← (e.type = LOGIN)
8      if e.type = LOGIN and e.node = w1 then
9          L ← L ∪ {e.time}
10     if on[w1] and on[w2] then
11         s ← s + Σ_{l∈L} (e.time − l)
12         r ← r + |L|
13         L ← ∅
14 return s/r
```

The pseudocode for the simulations estimating $\mathbb{E}(\mathbf{X}_e)$ is given in Algorithm 3. We generate samples of $\mathbf{X}_e$ by buffering all login instants $\{l_1, \cdots, l_n\}$ of node w_1 in set L (line 8), until the time t at which node w_2 logs in (line 10). At that point, we are able to generate $|L|$ samples $s_1, \cdots, s_n$ by taking $s_i = t - l_i$. These samples are accumulated in variable s (line 11) and, once a sufficient amount of samples are generated, we compute the average as the estimator for $\mathbb{E}(\mathbf{X}_e)$.

These simulations are much simpler than the ones in Section IV-A, for three reasons. First, when estimating the weight for an edge, we need to simulate only two nodes at a time instead of the entire network. Second, the quantity we estimate can be sampled by simply looking at the state of the processes, without an expensive breadth-first search over the entire graph. Third, due to the simplicity of these simulations, running the simulation longer is sufficient to trust the resulting average is not biased, without a costly per-repetition burn-in period.

C. Improving on the Bound

We expect the upper bound we just derived to be tight in some cases, and not so tight in others (e.g., when there are many similar-weight paths connecting source and destination). Intuitively, one can improve it by considering the top-k "fastest paths" between source and destination instead of taking just one. The main problem, however, is that reasoning about the

expected delay of at least one among a set of (possibly overlapping) paths is significantly more complex than computing this expected delay for a single path, since we can no longer simply compose edge delay expectations as before.

We can, however, verify the extent to which the top-k fastest paths between u and v are actually enough to approximate $\mathbb{E}(\mathbf{C}_{u,v})$ by means of the following procedure:

1) *Compute the top-k fastest paths.* Using the same edge weights $h_e = \mathbb{E}(\mathbf{X}_e)$ as we did in Sec. V-B, we run a top-k least-cost paths algorithm between u and v.
2) *Simulate over the resulting sub-graph.* The k least-cost paths between u and v induce a subgraph $G_{k,u,v}$ over G. By modeling churn with the same availability assignment of G for the corresponding vertices in $G_{k,u,v}$, we perform full simulations on $G_{k,u,v}$. We then compare the results for the source/destination pair u, v with those obtained by running the full simulations on G.

For this paper, we consider two top-k least-cost paths algorithms. The first one is due to Yen [19], and it finds a set of top-k least-cost paths which are allowed to share both edges and vertices. We refer to it as *Yen's top-k*. The second algorithm, instead, finds a set of paths which are edge-disjoint, and we refer to it as *edge-disjoint top-k*. It consists of a simple iterated application of Dijkstra's shortest-path algorithm, similar to the heuristic in [9], and works as follows. Starting with a graph G', we repeat for k iterations:

1) compute the shortest path P between u and v by using Dijkstra's algorithm;
2) remove the edges used in P from G'.

Again, estimating $\mathbb{E}(\mathbf{C}_{u,v})$ over $G_{k,u,v}$ is in general significantly cheaper than doing so over G since, as we discuss in Section V-D2, $G_{k,u,v}$ tends to be much smaller than G.

D. Evaluation

Our evaluation is divided in two parts. Section V-D1 provides numeric evidence to support the claim made in Eq. (7) that the expected delay for a path P is approximately the same as the sum of the expected delays of its edges. Section V-D2, instead, focuses on the end-to-end delay in ego networks, providing evidence that *i)* the upper bound from Section V-B holds, and *ii)* the subgraph $G_{k,u,v}$ induced by top-k least-cost paths connecting a pair of vertices u and v accounts for a significant part of the end-to-end delay between them. We also show the differences in tightness for the bounds produced by the two top-k approaches we adopt.

Again, we use ego networks, for the same reasons outlined in Section IV and to allow comparison of the results obtained here with those of the full simulations discussed there.

All the experiments we run in this section provide us with a value – either an upper bound or an estimate – on the end-to-end delay for a *pair* of source/destination nodes u and v. A single pair of nodes is, therefore, associated to different delay values, which are generated by different methods. To avoid confusion about the type of delay we are referring to, and how it has been obtained, we establish that:

1) end-to-end delay estimates produced by full simulations are referred to as $\mathbf{aed}$ (average end-to-end delay);
2) upper bounds produced by summing the estimates for expected edge delays along the least-cost path connecting u and v (Sec. V-B) are referred to as $\mathbf{aed_1}$;
3) upper bounds produced by full simulations over the subgraph induced by the top-k shortest paths connecting u and v (Sec. V-C) are referred to as $\mathbf{aed_k}$.

1) Numeric Evidence for Expectation: To support the claim of Eq. (7), we emulate the behavior of a path of size n by using a *list graph* of size n – a graph in which nodes are linked together in a doubly-linked list. An example for $n = 4$ is given in Fig. 4a. The experiment consists in measuring the end-to-end delay from node u (the leftmost vertex) to all nodes to the right. Note that the end-to-end delay is the same as the path delay in this case, since there is only one path connecting the source to each destination. For each node along the path, we compare the values of $\mathbf{aed}$ obtained by running full simulations with u as a source versus those obtained by summing the estimates for the expected edge delays along P. For our experiments, we consider $n = 15$, which is already more than twice the widely-accepted diameter of 6 for social networks [7]. We test 130 random availability assignments for this long path, which are generated according to the model described in Sec. II. Furthermore, for each assignment:

1) full simulations are ran 700 000 times;
2) edge delay estimates are obtained by averaging 500 000 samples per edge – in other words, we run Algorithm 1 with $n = 500\,000$.

Fig. 5a shows the delay values produced by the two methods for all pairs u, w, where u is the leftmost vertex in the list graph. Pairs are sorted in ascending order of their sum of expected edge delay values, shown as a curve. Points fall close to the curve, showing that Eq. (7) indeed holds in practice. A quantitative measure of error can be seen in Fig. 5b, which shows the ratio between the value produced by summing the edge delay estimates and $\mathbf{aed}$, with pairs sorted by their $\mathbf{aed}$ values this time. We put a dashed line at $y = 1$ where the

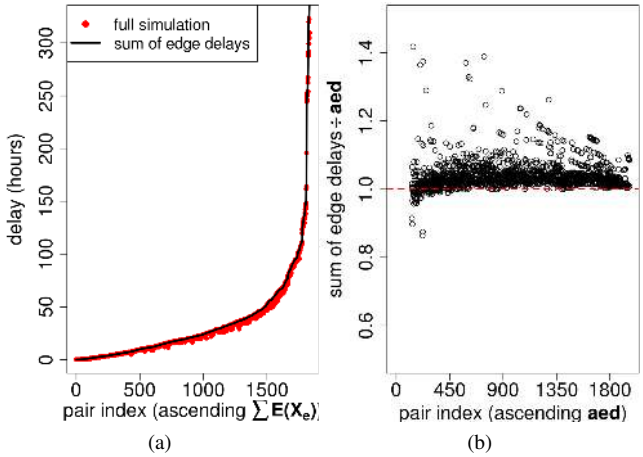
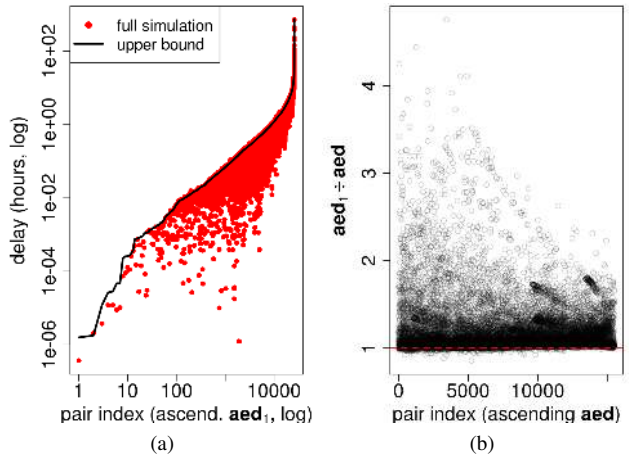


Fig. 5. Delay estimates of full simulations ($\mathbf{aed}$) vs. sums of edge delays.



(a) (b)

Fig. 6. Upper bound $\mathbf{aed_1}$ vs. full simulations $\mathbf{aed}$.

values coincide. The error stays on average below 4%, with 95% of the points staying under 10%. Most of the outliers with large relative errors (above 20%) lie in a zone where $\mathbf{aed}$ is small, meaning that the high percentage error translates into a small absolute error. Further, values in that area are more susceptible to random noise.

2) Delay on Ego Networks: We analyse, in this section, the same set of 700 ego networks we described in Sec. IV-B. To validate our model, we compare the $\mathbf{aed}$ values from the full simulations which we obtained previously against the values produced by our model. The expected edge delay estimates required by the model (Algorithm 1) are again obtained by averaging 500 000 samples per edge.

Upper Bound Holds. We validate our claim that the expected delay $\mathbb{E}(\mathbf{D}_{P_{min}})$ for the least-cost path connecting two nodes u and v serves as an upper bound to the expected end-to-end delay $\mathbb{E}(\mathbf{C}_{u,v})$. Fig. 6a shows, for each pair u, v in our sample, a log-log plot of the values of $\mathbf{aed}$ (dots), together with the corresponding $\mathbf{aed_1}$ values for the least-cost paths (thick black line). Pairs are ordered in the x axis in ascending order w.r.t. $\mathbf{aed_1}$.

As expected, dots fall consistently below the black line, providing solid evidence for our claim. The fact that some dots fall slightly above the black line is not really an issue, rather a result of the fact that we are using sample averages as estimators both for $\mathbb{E}(\mathbf{D}_{P_{min}})$ and $\mathbf{ard}$, and those are subject to noise, even with the many repetitions we run.

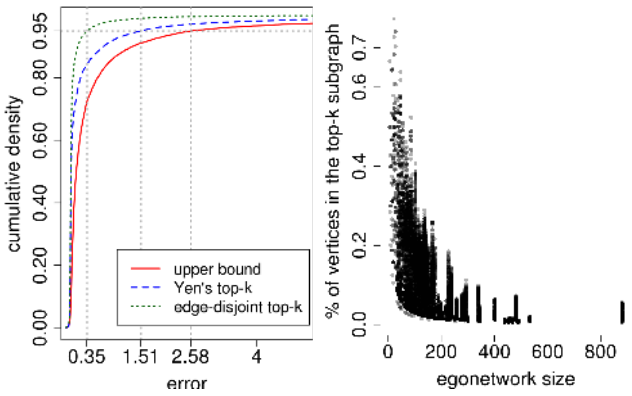
As for how "tight" the upper bound is, we again provide a plot which shows, for each pair, the ratio between $\mathbf{aed_1}$ and $\mathbf{aed}$ (Fig. 6b). To avoid the effects in the low $\mathbf{aed}$ region we observed in Fig. 5b, we constrain the pairs in Fig. 6b to those in which the $\mathbf{aed} \geq 0.5$ hour (72% of the pairs).

The upper bound provides a good estimate of $\mathbf{aed}$, falling within 16% of the full simulation values, 21% if we consider the worst 25% points, and 75% if we consider the worst 5% points. The bound becomes more accurate in relative terms for larger values of $\mathbf{aed}$, meaning it is better at helping identify cases for which we can expect poor performance.

Top-k Paths Provide a Better Bound. By using the procedure described in Sec. V-C, we investigate how much we can improve on our bound by considering not only the single least-cost path connecting a pair u, v, but the k least-cost paths. We carry out this evaluation on a subset of the 137 260 pairs we considered in Section IV-B, by drawing 14 000 source/destination pairs at random and without replacement from the original sample. For the experiments carried out in this section, we use a value of $k = 10$.

	Average	95^{th} percentile
Least-cost path (r_1)	21%	258%
Yen's Top-k (r_k)	7%	151%
Edge-Disjoint Top-k (r_k)	2%	35%

TABLE III
ERROR (r_1 AND r_k) STATISTICS FOR BOUNDS.



(a) (b)

Fig. 7. Least cost vs. top-k least cost paths, and size of top-k subgraphs.

We examine the metrics $r_k(u,v) = \frac{\mathbf{aed_k}}{\mathbf{aed}} - 1$, and $r_1(u,v) = \frac{\mathbf{aed_1}}{\mathbf{aed}} - 1$. These give an error measure, in percentage terms, of how much $\mathbf{aed_k}$ and $\mathbf{aed_1}$ deviate, respectively, from the $\mathbf{aed}$ estimate produced by the full simulations.

We plot in Fig. 7 the cumulative distributions for both r_k and r_1, with the 95% percentiles marked by lines. Further statistics are summarized in Table III. The top-k estimate stays significantly closer to the actual $\mathbf{aed}$ estimate than the upper bound (2% vs. 21% on average for the upper bound), with the top-k, edge disjoint paths performing better than Yen's for the value of k we consider. This can be attributed to the fact that the paths selected by Yen's algorithm tend to overlap heavily when, intuitively, the discussion in Sec. V-C points to the fact that selecting disjoint paths increases the probability that at least one of them materializes by some time instant t.

Finally, we show that although the delay estimate measured over the subgraphs $G_{k,u,v}$ (the subgraphs induced by the top-k edge-disjoint paths connecting u and v) is close to the delay observed over the full graph G, these subgraphs are in fact significantly smaller. This is shown in Fig. 7b, where we plot the percentage of the vertices carried over from each ego network G to $G_{k,u,v}$ as a function of how many vertices G has. We see that, the larger the graph, the smaller the percentage of vertices carried over, providing evidence that $G_{k,u,v}$ is indeed a smaller substructure that is capable of explaining the end-to-end delay over the larger graph, and hence where future analysis efforts should be directed.

VI. Related Work

Social overlays. There has been a growing interest in the use of social networks as communication networks, with social overlays having made their way into a number of system proposals over the past few years [2], [3], [8], [13]. Commonly cited reasons for using social overlays include desirable security properties, anonymity, censorship-resistance, among others. Yet, none of these properties are really useful if the network cannot enable basic message exchange among participants: yet, to the best of our knowledge, the impact of churn on social overlays has not been analyzed systematically. The work we develop here, therefore, is key to understanding under which circumstances such systems are viable or not.

Scheduled and temporal networks. Our work can be related on the surface to previous literature on scheduled [6] and temporal networks [12]. Although some of the formalisms found in these papers are similar to ours (e.g., Berman's [6] characterization of temporal paths) these works concern themselves mostly with combinatorial problems over temporal graphs for which the entire edge schedule is known in advance (e.g. generalizing Menger's theorem to temporal paths). Our work, instead, concerns itself with the asymptotic behavior of simpler properties of graphs for which we have a generating model, but for which the observation window is not finite.

Delay-tolerant networks (DTNs). Unlike the systems we target, where the network dynamics are determined by *node* churn, DTNs are characterized by *links* appearing and disappearing (e.g., due to mobility). The temporal networks that arise in both contexts are, however, similar.

Tang et al. [11] establishes a bridge between the realms of social network analysis and DTNs by considering mobile social networks. Their focus, however, is on the characterization of temporal graphs by appropriate "temporal metrics", which notably include a notion of temporal distance similar to our definition of end-to-end delay. In contrast, we consider only two metrics, but with the goal of developing techniques for analyzing their asymptotic behavior.

Chaintreau et al. [4] tackle problems similar to ours while studying the diameter of temporal networks. They analyze the conditions under which paths with logarithmic delay and hop count emerge as the network size approaches infinity, thus establishing bounds on the expected end-to-end delays. Their theoretical results are, however, based on a simple temporal generalization of Erdös-Rényi graphs, with the purpose of providing insight into the behavior of real opportunistic networks, along the lines of what Watts and Strogatz [18] did for social networks. In contrast, our goals are driven by the immediate need to provide system designers with latency bounds over real, arbitrary social graphs, given an availability model. These two differences—network model and overall goals—prevent a straightforward reuse of the results in [4] in our context.

VII. Conclusion and Future Work

This paper makes two main contributions. First, we have shown through simulations over real datasets that churn might induce non-negligible delays on information dissemination over social overlays. Second, we have introduced a hybrid analytical model for the communication delay between two vertices in general graphs and, based on this model, we have shown how to obtain reasonably accurate upper bounds by using lighter-weight simulations.

Finally, this work opens up a large research space. While it is clear that, in practical terms, a real P2P system based on a social overlays might provide adequate performance to the majority of its users, a small percentage of that user base might experience delays on the order of hours or more . Given the targeted scale of these systems, that small percentage can translate into millions of users. Understanding exactly what are the conditions under which this problem happens and how it can be mitigated is an open research question, and one that we plan to answer using our analytical model. As for mitigation strategies, a promising future direction is the use of hybrid architectures, where P2P dissemination over social overlays is assisted by helper servers, for example located in a cloud, to help improving only those scenarios where the availability of friend nodes is not sufficient to enable adequate performance.

References

[1] Diaspora website. https://joindiaspora.com/, [Apr 2012].

[2] Freenet website. http://www.freenet.org, [Mar 2012].

[3] Tonika website. http://5ttt.org, [Mar 2012].

[4] A. Chaintreau et al. The diameter of opportunistic mobile networks. In *Proc. of the 2007 SIGCOMM CoNEXT*, 2007.

[5] A. Mislove et al. Measurement and analysis of online social networks. In *Proc. Internet Measurement Conf.*, 2007.

[6] K. A. Berman. Vulnerability of scheduled networks and a generalization of menger's theorem. *Networks*, 48:125 – 134, 1996.

[7] D. Chuo. E-mail study corroborates six degrees of separation. *Scientific American*, 2003.

[8] 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.

[9] D. A. Dunn, W. D. Grover, and M. H. MacGregor. Comparison of k-shortest paths and maximum flow routing for network facility restoration. *IEEE Jrnl. Selected Areas in Comm.*, pages 88–99, 1994.

[10] G. S. Fishman and I. Adan. How heavy-tailed distributions affect simulation-generated time averages. Technical Report UNC/OR TR03-2, University of North Carolina, 2003.

[11] J. Tang et al. Characterizing temporal distance and reachability in mobile and online social networks. *SIGCOMM Comp. Comm. Review*, 40, 2010.

[12] D. Kempe, J. Kleinberg, and A. Kumar. Connectivity and inference problems for temporal networks. *Jrnl of Comp. and Sys. Sciences*, 64:820 – 842, 2002.

[13] G. Mega, A. Montresor, and G. P. Picco. Efficient dissemination in decentralized social networks. In *Proc. Conf. P2P Computing*, 2011.

[14] S. Saroiu, K. P. Gummadi, and S. D. Gribble. A measurement study of peer-to-peer file sharing systems. In *Proc. Multimedia Computing and Networking*, 2002.

[15] M. Steiner, T. En-Najjary, and E. W. Biersack. A global view of KAD. In *Proc. Internet Measurement Conf.*, 2007.

[16] I. Stoica, R. Morris, D. R. Karger, M. F. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *Proc. of ACM SIGCOMM*, pages 149–160, 2001.

[17] L. Toka, M. Dell'Amico, and P. Michiardi. Data transfer scheduling for p2p storage. In *Proc. Conf. P2P Computing*, 2011.

[18] D. Watts and S. Strogatz. Collective dynamics of small-world networks. *Nature*, 393(6684):440–442, 1998.

[19] J. Y. Yen. Finding the k shortest loopless paths in a network. *Management Science*, pages 712–716, 1971.

[20] Z. Yao et al. Modeling heterogeneous user churn and local resilience of unstructured P2P networks. In *Proc. Intl. Conf. Network Protocols (ICNP'06)*, 2006.