Prometheus: User-Controlled P2P Social Data Management for Socially-Aware Applications

Nicolas Kourtellis[1], Joshua Finnis[1], Paul Anderson[1], Jeremy Blackburn[1], Cristian Borcea[2] and Adriana Iamnitchi[1]

[1] Department of Computer Science and Engineering, University of South Florida
{nkourtel,jfinnis,paanders,jhblackb,anda}@cse.usf.edu
[2] Department of Computer Science, New Jersey Institute of Technology
borcea@cs.njit.edu

Abstract. Recent Internet applications, such as online social networks and user-generated content sharing, produce an unprecedented amount of social information, which is further augmented by location or collocation data collected from mobile phones. Unfortunately, this wealth of social information is fragmented across many different proprietary applications. Combined, it could provide a more accurate representation of the social world, and it could enable a whole new set of socially-aware applications. We introduce Prometheus, a peer-to-peer service that collects and manages social information from multiple sources and implements a set of social inference functions while enforcing user-defined access control policies. Prometheus is socially-aware: it allows users to select peers that manage their social information based on social trust and exploits naturally-formed social groups for improved performance. We tested our Prometheus prototype on PlanetLab and built a mobile social application to test the performance of its social inference functions under real-time constraints. We showed that the social-based mapping of users onto peers improves the service response time and high service availability is achieved with low overhead.

Keywords: social data management, P2P networks, socially-aware applications

1 Introduction

Recently, we have witnessed the emergence of a wealth of socially-aware applications, which utilize users' social information to provide features such as filtering restaurant recommendations based on reviews by friends (e.g., Yelp), recommending email recipients or filtering spam based on previous email activity [19], and exploiting social incentives for computer resource sharing [37]. Social information is also leveraged in conjunction with location/collocation data to provide novel mobile applications such as Loopt, Brightkite, Foursquare and Google's Latitude.

These applications need to collect, store, and use sensitive social information (including location and collocation). The state of the art is to collect and manage such information within the context of an application (as in the examples above) or to expose this information from platforms that specifically collect it, such as online social networks (OSN). For example, Facebook allows third-party application developers and websites to access the social information of its roughly 500 million users. Similarly, OpenSocial provides a common API for third-party applications to access social information stored on any of the supported social networks.

However, relying on social information exposed by OSNs has two major problems. First, the hidden incentives for users to have as many "friends" as possible lead to declarations of contacts with little connection in terms of trust, common interests, shared objectives, or other such manifestations of (real) social relationships [12]. Thus, an application that tries to provide targeted functionalities must wade through a lot of noise. Second, the business plans of the companies behind the social networks often conflict with the need to respect user privacy. Additionally, some social networks institute particularly draconian policies concerning the ownership of user-contributed information and content. For example, users cannot easily delete their OSN profiles, as in the case of Facebook which owns their data; they cannot export their social data to a service of their choice; and their social information has restricted use.

This paper presents Prometheus, a peer-to-peer service for user-controlled social data management that enables innovative socially-aware applications. Prometheus collects social information from multiple sources (e.g., OSNs, email, mobile phones) by quantifying the actual social interactions between users. Thus, it maintains richer and more nuanced social information, which can lead to more accurate inferences of trust, interests, and context. Prometheus implements and exposes to applications several social inference functions.

The choice of a peer-to-peer (P2P) architecture was motivated by two factors: user privacy and service availability. We considered two alternative architectures. In a centralized architecture (as in current OSNs) there are no incentives or appropriate business models to store social data for free and allow users to control the privacy of their data. Prometheus leverages the P2P infrastructure of Mobius [4] to provide user-controlled social data management and social inferences for services and applications. An alternative architecture stores social information on users' mobile devices, as proposed in [26, 27, 34, 36]. Even though much of the social information is nowadays generated by mobile devices, they are inherently unsuitable for running a complex social service such as Prometheus due to resource constraints: the mobile devices may not be always online or synchronized to provide fast and efficient inference support, and energy and computation power may be scarce.

Prometheus gives users the ultimate control over where their private information is stored in the system, and which applications and users are allowed to access it. We ensure data security and access control on the peers using built-in public-key cryptography primitives and user-defined access control policies.

We prototyped and evaluated Prometheus on a large-scale deployment on PlanetLab and tested its performance with realistic workloads. The results show that the response time for social inferences decreases when user information is mapped onto peers using a social-aware approach. Furthermore, high service availability is achieved with low overhead. To test the Prometheus service under real-time deadlines, we implemented CallCensor, a mobile social application that uses Prometheus to decide whether to filter out incoming calls based on the current social context. The response time for this application running on a Google Android phone meets the real-time deadlines, proving that the Prometheus service overhead is reasonable.

An overview of the Prometheus service is presented in Section 2, and related work is covered in Section 3. Section 4 presents a detailed design of Prometheus, and Section 5 describes the experimental evaluation. We conclude in Section 6.

2 Prometheus Overview

Prometheus is a peer-to-peer service that enables socially-aware applications by collecting information from multiple sources and exposing it through an interface that implements non-trivial social inferences. This is accomplished via decentralized storage and management of a weighted, labeled, directed multi-edge social graph on user-contributed nodes. The access to social data is controlled by user defined policies.

Specifically, users register with Prometheus and allow it to collect social information about them from multiple sources that we refer to as *social sensors*. Social sensors are applications running on behalf of a user which report to Prometheus information such as interactions with other users via email, phone, instant messaging, comments on blogs, ratings on user-generated content, or even face-to-face interactions determined from collocation data. Social sensors can therefore be deployed on the user's devices (e.g., mobile phone) or on the Web (e.g., as a Facebook application reporting wall writing interactions).

The information collected by social sensors is processed by Prometheus to create a decentralized, weighted, directed, and labeled multi-edged graph, where vertices correspond to users and edges correspond to interactions between users as reported by social sensors. The interactions are described with a label (e.g., "work", "hiking") and a weight that specifies the intensity of the interaction. Prometheus uses this decentralized graph to answer various social queries from applications. Both the social information from sensors and the social subgraphs are stored and maintained in a P2P network formed by user PCs. The information from sensors is stored in an encrypted form and can be decrypted only by "trusted" peers, which are selected by users. The subgraph of each user is stored on her trusted peers. The trusted peers also enforce user-defined access control policies for each social query.

Figure 1 presents the Prometheus architecture. Prometheus runs on top of Pastry [31], a distributed hash table (DHT)-based overlay, and uses Past [32] for replicated storage of sensor data. Each user has a group of trusted peers responsible for maintaining replicas of her social subgraph for high service availability. Only trusted peers can decrypt the user's social data. The maintenance of the

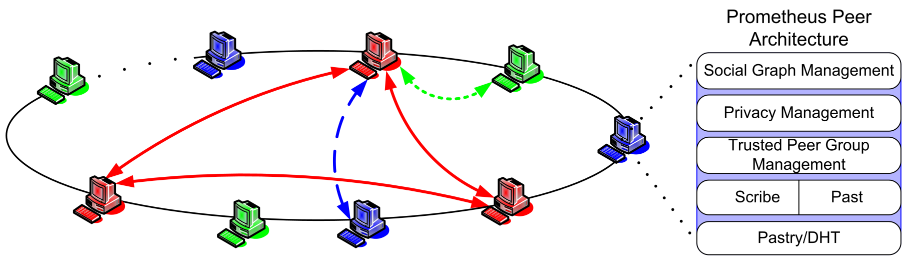


Fig. 1. Prometheus architecture: The service runs on top of Pastry, a DHT-based overlay, Scribe, a DHT multicast infrastructure, and Past, a DHT storage system. The social data from sensors are encrypted and can be stored at any peer. The social subgraphs are stored on groups of trusted peers. Each peer runs three Prometheus components for social graph management, privacy management, and trusted group management. Same color nodes are members of a user's trusted peer group. Continuous (*red*) arrows show group communication for social graph maintenance, while dashed (*blue* and *green*) arrows show social inference queries submitted to other peers.

trusted peer group is done by leveraging Scribe [6], an application-level DHT multicast infrastructure.

Prometheus exposes an interface that enables applications to have access to a rich set of social inference requests computed over the distributed social graph. For example, an application can request on its user's behalf to receive the top relations (a function of a specified label and weight) of its user. Similarly, it can request the social strength between its user and another user not directly connected in the social graph. Prometheus provides the mechanism by which inference requests can access not only a single user's social graph (i.e., directly connected users), but also the social information of users located several hops away in the global social graph.

All inferences are subject to users' access control policies. These policies allow users to have fine-grained control over the access of all or parts of their graph by other users. For example, these policies can specify access control as a function of social labels. Prometheus uses a public-key infrastructure (PKI) to ensure both message confidentiality and user authentication. Each user and group has its own pair of public/private keys. For access control purposes, users are identified by their public keys. Social sensors use the appropriate keys to encrypt the data submitted to Prometheus.

Prometheus not only enables novel socially-aware applications but also specifically takes advantage of social-based incentives. Social awareness is embedded in the design of Prometheus in two ways. First, Prometheus allows users to select trusted peers to maintain their social subgraph based on out-of-band relationships. Socially-incentivized users keep their computers online, thus reducing churn [22] and consequently increasing service availability for their friends' social inference requests. Second, given the characteristics of social graphs, socially-related users will likely select the same trusted peers to store their social subgraphs (i.e., friends have common friends who contribute peers in the system and thus select the same trusted peers). This allows complex social inferences over several social graph hops to be fulfilled locally, without traversing multiple network hops to access all the needed social information.

3 Related Work

Socially-aware applications and services have so far addressed ways to leverage out-of-band social relationships for diverse objectives such as improving security [41], inferring trust [23], providing incentives for resource sharing [37], or building overlays [28] for private communication. Leveraging online social information has been used to rank Internet search results relative to the interests of a user's neighborhood in the social network [13], to favor socially-connected users in a BitTorrent swarm [29], and to reduce unwanted communication [25].

In all of these cases, social knowledge has been mined in the context of a single application and from a single source of information. Our work differs from these systems in four important ways. First, we collect information from multiple sources. The only system we are aware of that considers social information aggregation for enriching information value is SONAR [16]. Unlike our approach, SONAR is limited to improving information flow in an organization by aggregating social network information within the enterprise context, from sources such as email, instant messaging, etc. Second, we extract social knowledge from the social graph by accessing larger portions of the network, not only a user's direct neighborhood. Somewhat similarly, RE [11] uses two hop relationships to automatically populate email whitelists. Third, in Prometheus, social information can cross application boundary contexts, similar to Ramachandran and Feamster's proposal to "export" the social ties formed in social networks for authentication in off-social applications [30]. And finally, our social graph representation offers richer information for fine-grained evaluation of social ties. Kahanda and Neville also represent interactions with a weighted, directed multi-edged graph [17]. Prometheus differs in its generality of the graph—its edges can represent interactions collected from different sources.

APIs for accessing social information has been offered by Facebook through its Open Graph API [1] for access to simple, unprocessed social data. This API does not provide direct "friends of friends" inferences as Prometheus does, but instead an application must explicitly crawl the graph.

Privacy in Online Social Networks (OSNs) through decentralization has been addressed in [7]. Prometheus similarly forwards messages based on social relations while restricting a large scale view of the graph through a P2P mechanism. Prometheus differs dramatically, however, in the exposure of the graph as a first class data object through inference functionality. Persona [3] uses an attribute-based encryption (ABE) to provide privacy in OSNs. Prometheus can leverage this system to provide more flexible access control policies.

Peer-to-peer management of social data is also proposed in PeerSoN [5] and Vis-à-Vis [35]. PeerSoN uses direct data exchange between users' devices.

Prometheus differs from this approach by using trusted peers, which are independent from the users' devices, to store and exchange social data. Vis-à-Vis introduces the concept of a Virtual Individual Server (VIS) where each user's data are stored on a personal virtual machine. While similar to the trusted peer concept in Prometheus, VISs are hosted by a cloud computing provider to counter peer churn, while Prometheus uses social incentives to reduce churn on user-supplied peers.

Our work significantly differs from the previously noted approaches in that it not only collects and stores user social information from multiple sources in a P2P network but also exposes social inference functions useful for novel socially-aware applications. This approach shares ideas with the MobiSoc middleware[15]; however, the MobiSoc middleware is logically centralized leading to "big brother" concerns similar to existing OSNs.

4 Prometheus Design

4.1 User Registration

Users register with Prometheus from a trusted device by contacting any peer in the network and are assigned a unique user id (UID). At registration time, they specify the peer(s) they will contribute to the network (if any). The peer handling the registration creates a mapping between UID and the list of these peers' IP addresses. This mapping is stored in the network as the key-value pair UID={IP_1, ..., IP_n}. When a peer returns from an offline state, it updates the mapping with its current IP address. Users have a pair of public/private keys, and the mappings are signed with their private key.

Users select, deploy, and configure the social sensors they want to use. They may wish to declare particular social relationships, such as family relations, that may prove difficult or impossible to infer by social sensors. Finally, they select an initial set of trusted peers based on out-of-bound trust relations with other Prometheus users. The larger the size of this set, the higher the service availability; but at the same time, the consistency and the overall performance may decrease. Users have also a pair of public/private keys for their trusted peer group. The group keys are transferred to each group peer by encrypting them with the peer's owner public keys. As social information about a new user will be incorporated in the social graph, users may be prompted with different choices for trusted peers (e.g., peers belonging to users with stronger ties).

4.2 Trusted Peer Group Management

We address three issues concerning the trusted peer group management of a user: (a) group membership, (b) search for trusted peers, and (c) group churn.

A user can add peers in the trusted peer group by executing a secure three-way handshake procedure which establishes a two-way trust relationship between her and the peer owner. All the messages in this exchange are signed and verified.

If the owner of a candidate peer accepts a peer addition request, the initiator will send the group keys to the new peer as described above. Then, the new peer subscribes to the Scribe *Trusted_Peer_Group_UID* of the user.

If a user decides to remove a peer from the trusted group (e.g., she no longer trusts the peer's owner), the user submits an unsubscribe request to one of the other trusted peers. This request is multicast to all group peers. Then, the peer generates a new pair of public/private keys for the group; this pair is distributed to all other trusted peers except the newly removed one. Additionally, the social sensors must be informed that the public key of the group has been changed. A peer owner may also decide to remove her peer from a trusted group of another user. This request is multicast to all the other group members.

Service requests can be sent to any peer, but only the trusted peers of a user can provide data about that user. Therefore, a random peer can find trusted peers for a user by submitting a multicast request to the user's trusted peer group. The group peers respond with their signed membership which is verified for authenticity with the user's group public key. The random peer is subsequently able to directly communicate with the individual trusted peers.

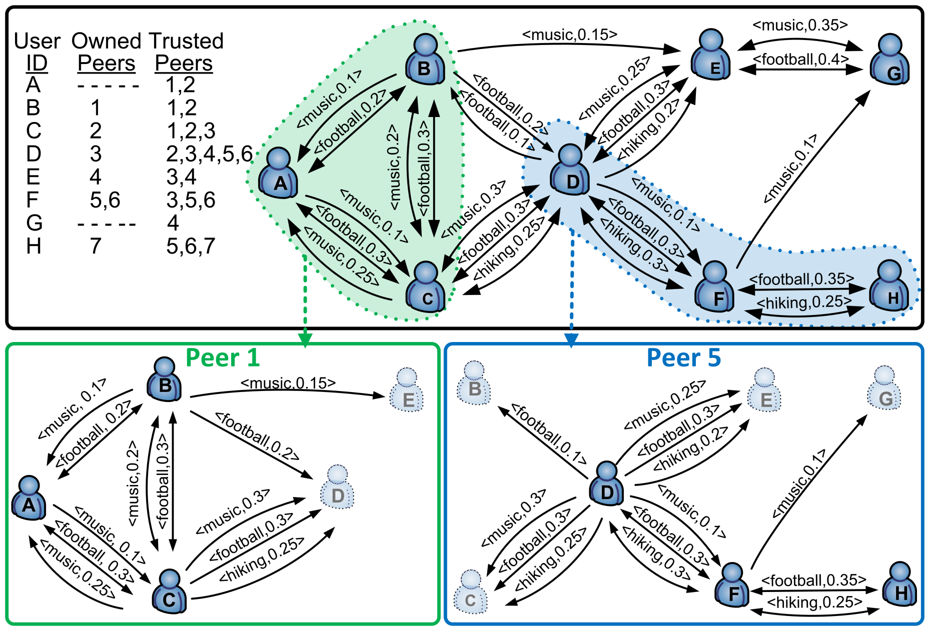
Past replicates the sensor data, thus maintaining data availability. However, the social graph for a user is unavailable if all her trusted peers leave the network; no service requests involving this user can be answered until a trusted peer rejoins the network. We ensure that generated data (e.g., input from social sensors) are not lost while a user's group is down via replication in the DHT by PAST. If a peer becomes untrusted while offline, a trusted peer from the particular group instructs the returning peer to unsubscribe.

4.3 Geo-Social Graph Representation

Prometheus represents the social graph as introduced in [2]: a directed, labeled, and weighted multi-edged graph, maintained and used in a decentralized fashion, as presented in Figure 2.

Multiple edges can connect two users, and each edge is labeled with a type of social interaction and assigned a weight that represents the intensity of that interaction. The labels for interactions and their associated weights are assigned by sensors. From an application point of view, distinguishing between different types of interactions allows for better functionality. Weights are assigned as a function of the number and frequency of interactions. This allows for a more accurate estimation of relationship strengths [40]. The latest known location of a user is also maintained as an attribute of the vertex representing the user in the graph.

We chose to represent the graph as directed because of the well-accepted result in sociology that "ties are usually asymmetrically reciprocal" [38]. Representing edges as directed enhances security by limiting the potential effects of illegitimate graph manipulation. For example, if Alice repeatedly performs interactions with Bob that are not reciprocated, only the edge connecting Alice to Bob will have its weight increased, not the edge from Bob to Alice.



User ID	Owned Peers	Trusted Peers
A	- - - - -	1,2
B	1	1,2
C	2	1,2,3
D	3	2,3,4,5,6
E	4	3,4
F	5,6	3,5,6
G	- - - - -	4
H	7	5,6,7

Fig. 2. An example of a social graph for eight users (*A*-*H*) who own seven peers. The mapping between users, peer owners, and trusted peers is shown in the upper left corner. Each edge is marked with its label and weight. The bottom figures illustrate the subgraphs maintained by peers *1* and *5*. Users shown in dark color (e.g., users *A*, *B* and *C* on peer *1*) trust the peer to manage their social data. Users shown in light shaded color (e.g., users *E and D* on peer *1*) do not trust the peer but are socially connected with users who do.

4.4 Social Sensors

Two types of social ties can typically be inferred from user interactions. The first type, *object-centric* ties, is identified through the use of similar resources or participation in common activities. Examples include tagging the same items in collaborative tagging communities such as Delicious or CiteULike and repeatedly being part of the same BitTorrent swarms. The second type, *people-centric* ties, is determined from declared social relationships (e.g., in online social networks), declared membership to groups (e.g., networks in Facebook or company name in LinkedIn), or collocation information.

Social sensors are applications that are installed on the user's mobile device or PC that aggregate and analyze the history of user's interactions with other users and report to Prometheus labels and weights corresponding to directed edges. These social data are encrypted with the public key of the user's trusted group, signed with the user's private key, and stored in the DHT by Past. Note that if web sensors (such as Facebook applications) are used, the data have to be first sent to one of the user's devices to be signed and encrypted. The data for each user are stored as records in an append-only file, named *Social_Data_UID*. Only the trusted peers can decrypt these records.

We designed Prometheus to be oblivious to the types of social activity reported by social sensors, thus allowing extensibility. For example, many such sensors already exist, although they have been implemented in different contexts; these sensors record and quantify OSNs' user activity [21], co-appearance on web pages [24], or co-presence recorded as collocation via Bluetooth [8]. Sensors can perform relatively sophisticated analysis before sending the data to Prometheus. For instance, collocation social sensors can differentiate between routine encounters with familiar strangers and interactions between friends [8].

4.5 Social State Maintenance

The social data for each user are stored in the file *Social_Data_UID* as encrypted records. To ensure atomic appends, a lock file associated with the data file is created by the sensor trying to append a record. Other sensors trying to append concurrently will receive an error when trying to create this file. Once the append is done, the lock file is deleted. Since the file is append-only, readers can access it at any time: in the worst case, they will miss the latest update.

Social sensors can send updates to create new edges, remove old edges, or modify an edge weight. Each record contains a sequence number and encrypted data with the label and its associated weight. Trusted peers periodically check the file for new records and retrieve all such records: this is easily done based on sequence number comparison starting from the end of the file. The peer decrypts the new records and verifies the digital signature to make sure the updates are authentic. Then, it updates the subgraph of the user with the newly retrieved records. For short periods of time, the trusted peers may have inconsistent data, but this is not a major problem as social graphs do not change often.

Social sensors may specify that certain edges must be "aged" over time if no new updates for those edges are received (i.e., lack of social interactions associated with those labels). The social sensors may also specify the decrement value and the time period for aging (these values are also stored in the *Social_Data_UID* file for each user).

4.6 Service Interface

Prometheus implements in a distributed fashion a set of basic social inference functions, which are exposed to applications through a service interface. We assume that every device running an application that interacts with Prometheus caches a number of peer IP addresses to bootstrap the interaction.

Social Inference Functions: We implemented the following social inference functions; more complex inferences can be built on top of this set.

relation_test(ego, alter, α, x) is a boolean function that checks whether *ego* is directly connected to *alter* in the social subgraph of *ego* by an edge with label α and with a minimum weight of x. A mobile phone application can use this function, for example, to determine whether an incoming call is from a coworker with a strong social tie, and therefore, should be let through even on weekends.

top_relations(ego, α, n) returns the top n users in the social subgraph of *ego* (ordered by decreasing weights) who are directly connected to *ego* by an edge with label α. An application can use this function, for example, to invite users highly connected with *ego* to share content related to activity α.

neighborhood(ego, α, x, radius) returns the set of users in *ego*'s social neighborhood of size *radius* who are connected through social ties of a label α and minimum weight of x. The *radius* parameter allows for a multiple hop search in the social graph (e.g., setting *radius* to 2 will find friends of friends). Our CallCensor mobile phone application which silences *ego*'s cell phone during meetings at work (Section 5.3) uses this function to determine if a caller is in *ego*'s work neighborhood in the social graph even if not directly connected.

proximity(ego, α, x, radius, min_distance, timestamp) is an extension of the neighborhood function which filters the results of the social neighborhood inference based on physical distance to *ego*. After the location information is collected for *ego* and the set of users is returned by *neighborhood*, *proximity* returns the set of users who are within *min_distance* from *ego*. A mobile phone application might use this function to infer the list of collocated coworkers within a certain distance of *ego*. Users who do not share the location or have location information older than the *timestamp* will not be returned.

social_strength(ego, alter) returns a real number in the range of [0, 1] that quantifies the social strength between *ego* and *alter* from *ego*'s perspective. The two users can be multiple hops apart in the social graph. The return value is normalized, as shown below, to *ego*'s social ties to ensure that the social strength is less sensitive to the social activity of the users. NW is the normalized weight between two directly connected users, K is the path length joining two indirect users, N_i is a user in this path, w is the weight of an edge, and *strength* is the return value for a multi-hop path:

$$NW(ego, neighbor_i) = \frac{\sum\limits_{all-labels} w(ego, neighbor_i)}{\max\limits_{all-neighbors}\Big(\sum\limits_{all-labels} w(ego, neighbor)\Big)} \,. \tag{1}$$

$$strength\big(path(N_1, N_2, \ldots, N_K)\big) = \max_{all-paths} \frac{\min\limits_{i=1,\ldots,K-1}\big(NW(N_i, N_{i+1})\big)}{K} \,. \tag{2}$$

Such a function could be used, for example, to estimate social incentives for resource sharing. We limit the length of the indirect path that connects two users to two, using a well-accepted result in sociology known as the "horizon of observability" [9].

Inference Function Execution: Social inference requests are signed with the private key of the user who submitted them and sent to any Prometheus peer. The request will then be forwarded to one of the trusted peers of the user, which enforces the access control policies and verifies the submitter's identity through her public key. The peer fulfills the request by traversing the local social subgraph for the information requested by the application, encrypting and signing the result using the requesting user's public key and the trusted group's private key, respectively, and returning the result to the application.

For functions that need to traverse the graph for more than one hop (e.g., social_strength), the peer will forward requests for information about other users to their trusted peers. The requests include the UID of the original submitter (in order to verify her access rights). The peer signs these requests with the group private key and optionally encrypts them with the destination group public key. Each receiving peer authenticates the request and checks the access control policies for the requesting user. If the request is granted, the result is returned to the requesting peer. If the request still needs more information, the peer repeats the same process. Finally, the original requesting peer collects recursively all the replies and submits the final result to the application.

4.7 Access Control Policies

Users can specify access control policies (ACP) upon registration and can update them any time thereafter. These policies are stored on each trusted peer of the user and are applied each time an inference request is handled by these peers. For availability, the policies are also stored encrypted with the group public key in the DHT, thus allowing rejoining trusted peers to recover policies updated while the peer was offline. The same mechanism used for updating the social graph is used to update policies. As future work, we plan to investigate providing strong consistency for policies.

Currently, we consider four different categories of social information for which users can set access control policies: relations, labels, weights, and location. By design, ACPs are whitelists. To specify who is allowed to access these categories of social information, ACPs allow the following elements to appear: individual users, groups of users identified by labels, application name to allow access to certain applications no matter which user runs them, and number of social hops the requesting users must be within. To verify the access rights, Prometheus may call its inference functions. For example, to detect whether the originator of the request is within a certain number of hops, the originator is checked against the result of a two-hop *neighborhood* function. ACPs also allow for blacklisted users to provide convenience in the case that a user wants to provide access to a large set of users and exclude a small number of members of that set.

Relations, labels, and weights allow fine-grained control over access to users' social graphs. Setting the relations field would prevent any user not allowed access from completing an inference that asks whether the user is connected with some other user, which accounts for most social inferences. Social inference functions which make use of labels as a filter provide an increased amount of data about a user; hence, it is likely that users will set this field more restrictively than the relations field. Prometheus allows users to set restrictions for any label

```
relations: hops-2
hiking-label: lbl-hiking
work-label: lbl-work
general-label:
weights:
location: hops-1
------------------
blacklist: user-Eve
```

Fig. 3. Example of an access control policy in Prometheus

in their social graph in their ACP. Upon an inference request, Prometheus will first check to see if the specific label has any restrictions, and if not, check the "general-label" category for restrictions on all labels. Weights provide even more detail about user social graphs. Users can set additional restrictions upon social inferences that request weighted data. Finally, location governs access to any inference function which asks for a user's location.

Figure 3 shows an example ACP for Alice that disallows any social inference that makes use of the social graph (through the relations field) originating from a user outside of two social hops, with an additional restriction that only those connected to her with a "hiking" label can request social inferences for her "hiking" information, excluding the user Eve. Social inferences which use location are restricted to those she is directly connected with. If a "hiking" neighborhood inference is submitted to Alice, Prometheus checks her ACP in the order relations→hiking-label→weights.

5 Experimental Evaluation

We implemented Prometheus on top of the FreePastry Java implementation of Pastry DHT which also provides API support for Scribe and Past. The social graph management is implemented in Python. The Prometheus prototype was deployed and evaluated on PlanetLab.

Our evaluation had three goals: (1) measure Prometheus performance over a large scale worldwide distributed network using realistic workloads with a large number of users, created based on previous studies of social media user behavior; (2) assess the effect of socially-aware trusted peer selection on the system's overall performance; and (3) validate Prometheus with a social application developed on mobile phones and measure the performance of this application. An implicit goal was to identify potential performance optimizations, as we focused so far on building a robust end-to-end system. The two metrics used in this evaluation were *end-to-end response time* to quantify the user-perceived performance and *number of network messages* to quantify the service overhead.

In the first set of experiments, we used 100 peers deployed on PlanetLab, and each peer submitted workload on behalf of 1000 users. The number of users trusting each peer was varied from 10 to 30, i.e., each user trusted 1 and 3 peers respectively. The user distribution across peers was varied using two methods: i) *random* — users randomly trust peers, and ii) *social* – socially connected users trust the same peer. We limit the number of users assigned on a peer to 30, based on sociological studies [10] that claim that, on average, an urban person has meaningful social relationships with about 30 other individuals.

The second set of experiments was designed to assess Prometheus performance when used with a real application and on a real social graph. We implemented a mobile application, CallCensor, that silences the phone ring based on the social context of the callee and the relationship with the caller.

In the experiments, we used a timeout of 15 seconds for every social hop in the graph traversed by Prometheus to fulfill a request. After this timeout, a Prometheus peer handling either the initial request or subsequent secondary requests responded with the results it had at that point. Applications could modify the timeout parameter to trade waiting time for more information returned by social inferences.

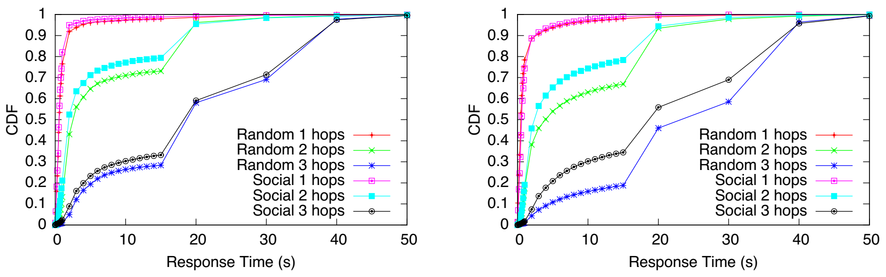
5.1 Synthetic Workloads and Social Graph

We evaluated the operation of Prometheus with a large user base. We emulated the workload of two socially-aware applications and one social sensor based on previous system characterizations [39, 20, 14].

Workload for Social Sensor: We emulated a Facebook social sensor based on a Facebook trace analysis [39]. The workload was characterized by the probability distribution function for users to post comments on walls and photos. Users were ranked into groups based on their social degree and each group was mapped onto a probability class using the cumulative distribution function from Figure 8 in [39]. To emulate a social interaction from *ego* to *alter*, a group was selected based on its associated probability, and a user *ego* from the group (who was not selected yet) was picked as the source of input. *Alter* was randomly selected from the *ego*'s direct social connections. The weight of each input was kept constant to a small value for all users. Since users were picked based on their social degree, the users with higher social degree probabilistically produced more input, leading to higher weights on their corresponding edges in the social graph.

Workload for Neighborhood Inference: We used an analysis of Twitter traces [20] to associate a tweet with a neighborhood request (centered at the leader of the tweet) in Prometheus. Thus, we extracted the probability distribution function of submitted requests. Users were ranked into groups based on their social degree ratio (the number of incoming edges divided by the number of outgoing edges). Based on Figure 4 in [20], each group was mapped onto a particular probability to be selected for a neighborhood request. Once the group was selected, a user from the group (who was not selected yet) was picked to be the source of the request. The number of hops for the request was randomly picked from 1, 2 or 3 hops.

Workload for Social Strength Inference: We used an analysis of BitTorrent traces to emulate the workload of a battery-aware BitTorrent applica-



(a) 10 users per peer (b) 30 users per peer

Fig. 4. CDF for the average end-to-end response time of *neighborhood* inference for two types of user-to-peer mapping (*Random* and *Social*) and two numbers of users per peer (*10* or *30*).

tion [18] on mobile devices: a user may rely on social incentives to be allowed to temporarily "free ride" the system when low on battery. Members of the same swarm check their social strength with the needy leecher to see if they want to contribute by uploading on her behalf.

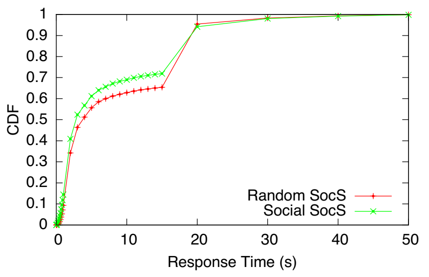
We assumed that users participated at random in swarms. Two users were randomly selected as the source and destination of the social strength inference request. The user selected as the source was associated with a total number of requests she would submit throughout the experiment. This number was extracted from an analysis of BitTorrent traces (Figure 9b in [14]).

Social Graph: We used a graph of 1000 users created with a synthetic social graph generator described in [33]. The generator consistently produces graphs with properties such as degree distribution and clustering coefficient similar to real social graphs. We used this graph as a bidirectional graph and applied a low weight threshold on the inference requests to produce a high-stress load.

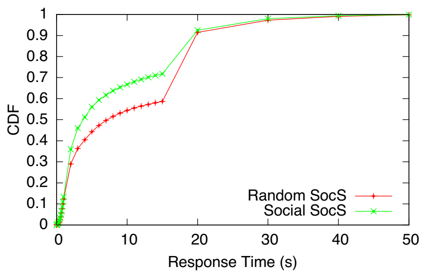
5.2 Results from Synthetic Workload Experiments

For every run of the experiments, more than 200,000 social strength and neighborhood requests and more than 32,000 social inputs were submitted from the emulated applications and social sensors. Figure 4 shows the end-to-end average response time for the *neighborhood* inference. We learn four lessons from these experiments.

First, the social-based mapping of users onto peers leads to significant improvements, especially for the 30 users/peer case. For this case, we have as much as 15% of the invocations finishing faster when compared to the random case (some invocations can finish in half the time). Additionally, the benefits compound as the number of hops increases. Of course, the difference is visible only for 2 and 3 hops, as the 1 hop function is computed locally. While we do not plot the number of messages in the system for the sake of brevity, we notice that



(a) 10 users per peer



(b) 30 users per peer

Fig. 5. CDF for the average end-to-end response time of social inference function *social strength (Socs)* for two types of user-to-peer mapping (*Random* and *Social*) and two levels for number of users per peer (*10* and *30* users per peer).

the social-based mapping reduces the communication overhead by more than an order of magnitude.

Second, the results show that a three-fold improvement in service availability can be achieved with minimum performance degradation. This is because the graph on the right side has three times more trusted peers per user and its overall response time is only marginally inferior compared with the graph on the left. Therefore, Prometheus design for availability (i.e., replicating the social graph on all trusted peers) is proven to work well in a realistic scenario.

Third, the absolute values of the response time are relatively high, especially for 2 and 3 hops. This is mostly due to the communication delay introduced by the P2P network. In our testbed, the average RTT is 200-300 msec. This value is multiplied by the number of hops traveled by a request/response. For example, for 10 users/peer, an average of 67 peers have to be contacted to collect *neighborhood* data from users located 3 social hops away (the number of users is 350). To improve these results, we plan to implement caching of recently computed results as well as pre-computing results in the background. These methods are expected to work well because the social graph changes rarely. Therefore, the cost associated with maintaining consistency should be low.

Fourth, creating the trusted peer list can be an expensive operation. Let us recall that a request can arrive at a random node, which has to first acquire the list of trusted peers for the user, and then forward the request to one of these trusted peers. This operation involves several lookups in the DHT, which result in multiple peer traversals. To solve this problem, Prometheus caches the list of trusted peers after the first access. The two graphs show the performance using this caching mechanism. The overhead associated with the cold start of creating the trusted peer list is as much as 10 sec for 3 hops.

Figure 5 shows the end-to-end average response time for the *social strength* inference. The performance is almost identical to the one for *neighborhood* for 2 hops because this function has to verify all possible paths between two users, but is limited to users located 2 social hops away.

5.3 CallCensor: A Real-Time Mobile Socially-aware Application

The CallCensor application leverages social information received from Prometheus to decide whether to allow incoming calls to go through. In addition to the social information from Prometheus, this application uses the phone location to infer whether the user is in a meeting (e.g., in the office). For each incoming call, the application queries Prometheus with a *social strength* or *neighborhood* inference request to assess the type of social connection between the caller and the phone owner. Based on the owner settings (e.g., allow calls from the spouse anytime), the application decides if the phone should ring, vibrate or silence upon receiving the call. The application was written in Java for mobile devices running the Google Android OS and was tested on a Nexus One mobile phone from HTC (1GHz processor, 512MB RAM).

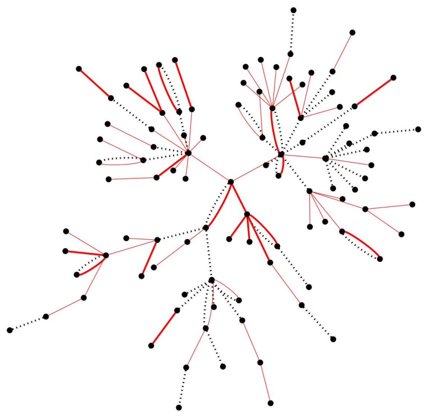
We tested three scenarios in which a caller can be connected to the callee: directly connected within 1 social hop, indirectly connect by 2 social hops, and connected with a high social strength (independent of the number of hops). We tested each of these scenarios 50 times on 3 PlanetLab peers. We measured the end-to-end response time of an inference request submitted to Prometheus. This experiment introduced additional overhead due to the communication between the mobile application and Prometheus.

The social graph used in these experiments was based on data collected at NJIT. The graph has two types of edges, representing Facebook friends and Bluetooth collocation. Mobile phones were distributed to students and collocation data (determined via Bluetooth addresses discovered periodically by each mobile device) were sent to a server. The same set of subjects installed a Facebook application to participate in a survey, and they gave us permission to collect their friend lists. The user set was small (100 users) compared to the size of the student body (9000), therefore resulting in a somewhat sparse graph. About half of the subjects reported less than 24 hours of data over the span of a month. The collocation data have two thresholds of 45 and 90 minutes for users to have spent together; thus, the 90 minute collocations comprise of a subgraph of the 45 minute collocations.

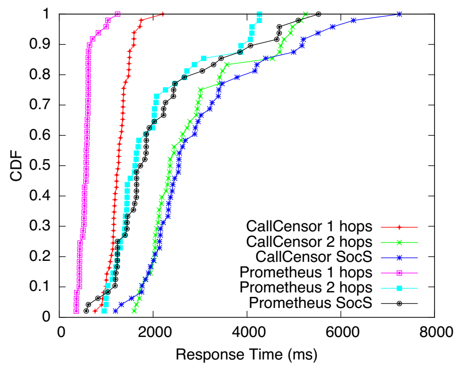
While the edges on the graph were not initially weighted, we applied synthetic weights of 0.1 for "facebook" edges, 0.1 for "collocation" of 45 minutes and 0.2 for "collocation" of 90 minutes. For the experiments, we consider the "collocation" edges to represent a work relationship, while the "facebook" edges represent a personal relationship. The user (*ego*) was assumed to be in a work environment when another user (*alter*) called. Figure 6(a) illustrates this graph and demonstrates one of the features of Prometheus: using multi-edge graphs provides for better social inferences. Neither the "facebook" nor the "collocation" graph is connected, but the graph containing both types of edges is.

The users were assigned to trust 3 PlanetLab peers using a social-based distribution. For each of the scenarios tested, the *ego* and *alter* were randomly chosen, and the inference request was sent to a random peer.

Figure 6(b) presents the end-to-end average response time for the requests sent by CallCensor for the three cases mentioned above. The results also show



(a) Input social graph



(b) Average response time

Fig. 6. The input social graph (a) has two types of edges: black dashed lines are Facebook edges, and red continuous lines are collocation edges. Line thickness demonstrates the weight of the edge. The results (b) show the CDF for the average end-to-end response time of CallCensor on 3 PlanetLab nodes for *neighborhood* for *1* and *2 hops* and *social strength (SocS)*. We show the time needed by Prometheus to produce a response and also the overall time needed by CallCensor to request and handle this response.

the time spent by the requests only in Prometheus. We first observe that the results are acceptable for the real-time constraint of the application: the response must arrive before the call is forwarded to the voice-mail of the callee. Second, we notice that the application itself introduced a significant overhead: for example, 50% in the 2-hop *neighborhood* and *social strength* cases due to both communication overhead and relatively slow execution on the mobile phone. Third, we observe the similarity of the *social strength* results with the *neighborhood* for 2 social hops, as found in the previous experiments.

6 Conclusions and Future Work

This paper presented Prometheus, a P2P service that enables socially-aware applications by providing decentralized, user-controlled social data management. Its decentralized, multi-edged, directed and weighted graph offers a fine-grained representation of the users' social state. Since Prometheus provides good privacy and availability, we expect users to provide a significant amount of social information, well beyond what is available today. We built and evaluated Prometheus using a large scale distributed testbed and a realistic workload. Additionally, we implemented a proof-of-concept mobile social application that utilizes Prometheus functionalities.

The performance results can certainly be optimized as we focused only on functionality so far. As mentioned in the previous section, we plan to cache and pre-compute results benefiting from the slow changes that occur in social graphs. A possible solution for ensuring consistency in such a case is to use the DHT storage to store "dirty bits" for each user. These bits would show if users' information has been updated by social sensors, thereby informing peers that their cached results are stale and that they should rerun the inferences.

We plan to expand the set of social inferences as well as to allow different sensors to provide input for the same label. Additionally, we will explore activity ontologies, provided to social sensors by Prometheus, to support label consistency across multiple sensors.

Prometheus peers have been assumed trusted and cooperative. Due to its distributed nature, Prometheus is harder to be completely compromised than centralized solutions. Similarly, it is more resilient to DoS attacks. Nevertheless, we plan to examine the implications of malicious users and peers in the near future. Of special concern is the case where a (previously) trusted peer becomes malicious. While this newly malicious peer can certainly be removed from a user's trusted peer group, it still retains previously acquired social knowledge, and thus, the ability to subvert the service experienced by the user-owner of the social data. Also, a trusted peer can become faulty and provide inaccurate results. We plan to investigate Byzantine fault-tolerance protocols to guarantee the validity of the results. Finally, while our default access control policies prevent any single user from gaining knowledge of the entire graph, we have yet to ascertain what level of collusion between users would expose the entire graph.

Acknowledgments. This research was supported by the NSF under Grants No. CNS 0952420, CNS 0831785 and CNS 0831753. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the sponsors. We thank Daniel Boston, Juan Pan, and Steve Mardenfeld from NJIT for collecting the Bluetooth collocation traces.

References

1. Graph api - facebook developers. `http://developers.facebook.com/docs/api`
2. Anderson, P., Kourtellis, N., Finnis, J., Iamnitchi, A.: On managing social data for enabling socially-aware applications and services. In: 3th Workshop on Social Network Systems (2010)
3. Baden, R., Bender, A., Spring, N., Bhattacharjee, B., Starin, D.: Persona: An online social network with user-defined privacy. ACM Computer Communication Review 39(4), 135–146 (2009)
4. Borcea, C., Iamnitchi, A.: P2P systems meet mobile computing: A community-oriented software infrastructure for mobile social applications. In: 2nd Int. Conf. on Self-Adaptive and Self-Organizing Systems Workshops. pp. 242–247 (2008)
5. Buchegger, S., Schiöberg, D., Vu, L., Datta, A.: PeerSoN: P2P social networking: early experiences and insights. In: 2nd Workshop on Social Network Systems. pp. 46–52 (2009)
6. Castro, M., Druschel, P., Kermarrec, A., Rowstron, A.: Scribe: A large-scale and decentralized application-level multicast infrastructure. IEEE Journal on Selected Areas in communications 20(8), 1489–1499 (2002)

7. Cutillo, L., Molva, R., Strufe, T.: Privacy preserving social networking through decentralization. In: 6th Int. Conf. on Wireless On-Demand Network Systems and Services. pp. 133–140 (2009)
8. Eagle, N., Pentland, A.S.: Reality mining: sensing complex social systems. Personal and Ubiquitous Computing 10(4), 255–268 (2006)
9. Friedkin, N.E.: Horizons of observability and limits of informal control in organizations. Social Forces 62(1), 57–77 (1983)
10. Friedkin, N.E.: The development of structure in random networks: an analysis of the effects of increasing network density on five measures of structure. Social Networks 3(1), 41 – 52 (1981)
11. Garriss, S., Kaminsky, M., Freedman, M.J., Karp, B., Mazières, D., Yu, H.: Re: reliable email. In: 3rd Conf. on Networked Systems Design and Implementation (2006)
12. Golder, S.A., Wilkinson, D., Huberman, B.A.: Rhythms of social interaction: Messaging within a massive online network. In: 3rd Int. Conf. on Communities and Technologies (2007)
13. Gummadi, K.P., Mislove, A., Druschel, P.: Exploiting social networks for internet search. In: 5th Workshop on Hot Topics in Networks. pp. 79–84 (2006)
14. Guo, L., Chen, S., Xiao, Z., Tan, E., Ding, X., Zhang, X.: Measurements, analysis, and modeling of bittorrent-like systems. In: 5th Conf. on Internet Measurement (2005)
15. Gupta, A., Kalra, A., Boston, D., Borcea, C.: MobiSoC: a middleware for mobile social computing applications. Mobile Networks and Applications 14(1), 35–52 (2009)
16. Guy, I., Jacovi, M., Shahar, E., Meshulam, N., Soroka, V., Farrell, S.: Harvesting with SONAR: the value of aggregating social network information. In: 26th Conf. on Human factors in computing systems. pp. 1017–1026 (2008)
17. Kahanda, I., Neville, J.: Using transactional information to predict link strength in online social networks. In: 3rd AAAI Int. Conf. on weblogs and Social Media (2009)
18. King, Z., Blackburn, J., Iamnitchi, A.: BatTorrent: A battery-aware bittorrent for mobile devices. In: 11th Int. Conf. on Ubiquitous Computing, Poster Session (2009)
19. Kong, J.S., Rezaei, B.A., Sarshar, N., Roychowdhury, V.P., Boykin, P.O.: Collaborative spam filtering using e-mail networks. Computer 39(8), 67–73 (2006)
20. Krishnamurthy, B., Gill, P., Arlitt, M.: A few chirps about twitter. In: 1st Workshop on Online Social Networks. pp. 19–24 (2008)
21. Lewis, K., Kaufman, J., Gonzalez, M., Wimmer, A., Christakis, N.: Tastes, ties, and time: A new social network dataset using Facebook.com. Social Networks 30(4), 330 – 342 (2008)
22. Li, J., Dabek, F.: F2F: reliable storage in open networks. In: 5th Int. Workshop on Peer-to-Peer Systems (2006)
23. Maniatis, P., Roussopoulos, M., Giuli, T.J., Rosenthal, D.S.H., Baker, M.: The LOCKSS peer-to-peer digital preservation system. ACM Trans. Comput. Syst. 23(1), 2–50 (2005)
24. Matsuo, Y., Mori, J., Hamasaki, M., Ishida, K., Nishimura, T., Takeda, H., Hasida, K., Ishizuka, M.: Polyphonet: an advanced social network extraction system from the web. In: 15th Int. Conf. on World Wide Web. pp. 397–406 (2006)
25. Mislove, A., Post, A., Druschel, P., Gummadi, K.P.: Ostra: leveraging trust to thwart unwanted communication. In: 5th Symposium on Networked Systems Design and Implementation. pp. 15–30 (2008)

26. Mokhtar, S.B., McNamara, L., Capra, L.: A middleware service for pervasive social networking. In: 1st Int. Workshop on Middleware for Pervasive Mobile and Embedded Computing. pp. 1–6 (2009)
27. Pietiläinen, A.K., Oliver, E., LeBrun, J., Varghese, G., Diot, C.: MobiClique: Middleware for mobile social networking. In: 2nd Workshop on Online Social Networks. pp. 49–54 (2009)
28. Popescu, B., Crispo, B., Tanenbaum, A.: Safe and private data sharing with Turtle: Friends team-up and beat the system. In: Security Protocols. LNCS, vol. 3957, pp. 213–220. Springer Berlin / Heidelberg (2006)
29. Pouwelse, J., Garbacki, P., Wang, J., Bakker, A., Yang, J., Iosup, A., Epema, D.H.J., Reinders, M., van Steen, M., Sips, H.: Tribler: A social-based peer-to-peer system. Concurrency and Computation: Practice and Experience 20, 127–138 (2008)
30. Ramachandran, A.V., Feamster, N.: Authenticated out-of-band communication over social links. In: 1st Workshop on Online Social Networks. pp. 61–66 (2008)
31. Rowstron, A., Druschel, P.: Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In: 1st Int. Conf. on Distributed Systems Platforms. pp. 329–350 (2001)
32. Rowstron, A., Druschel, P.: Storage management and caching in PAST, a large-scale, persistent peer-to-peer storage utility. In: 18th Symposium on Operating systems principles. pp. 188–201 (2001)
33. Sala, A., Cao, L., Wilson, C., Zablit, R., Zheng, H., Zhao, B.Y.: Measurement-calibrated graph models for social network experiments. In: 19th Int. Conf. on the World Wide Web. pp. 861–870 (2010)
34. Sarigol, E., Riva, O., Alonso, G.: A tuple space for social networking on mobile phones. In: 26th Int. Conf. on Data Engineering (2010)
35. Shakimov, A., Varshavsky, A., Cox, L., Cáceres, R.: Privacy, cost, and availability tradeoffs in decentralized OSNs. In: 2nd Workshop on Online Social Networks. pp. 13–18 (2009)
36. Toninelli, A., Pathak, A., Seyedi, A., Sepicys Cardoso, R., Issarny, V.: Middleware support for mobile social ecosystems. In: 2nd Int. Workshop on Middleware Engineering (2010)
37. Tran, D.N., Chiang, F., Li, J.: Friendstore: cooperative online backup using trusted nodes. In: 1st Workshop on Social Network Systems. pp. 37–42 (2008)
38. Wellman, B.: Structural analysis: From method and metaphor to theory and substance. Social structures: A network approach. pp. 19–61 (1988)
39. Wilson, C., Boe, B., Sala, A., Puttaswamy, K.P.N., Zhao, B.Y.: User interactions in social networks and their implications. In: 4th European Conf. on Computer systems. pp. 205–218 (2009)
40. Xiang, R., Neville, J., Rogati, M.: Modeling relationship strength in online social networks. In: 19th Int. Conf. on World Wide Web. pp. 981–990 (2010)
41. Yu, H., Kaminsky, M., Gibbons, P.B., Flaxman, A.: Sybilguard: defending against sybil attacks via social networks. In: Conf. on Applications, technologies, architectures, and protocols for computer communications. pp. 267–278 (2006)