January 2012

On the Design of Socially-Aware Distributed Systems

Nicolas Kourtellis
University of South Florida, nkourtel@mail.usf.edu

On the Design of Socially-Aware Distributed Systems

by

Nicolas Kourtellis

A dissertation submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
Department of Computer Science and Engineering
College of Engineering
University of South Florida

Major Professor: Adriana Iamnitchi, Ph.D.
Cristian Borcea, Ph.D.
Miguel Labrador, Ph.D.
Jay Ligatti, Ph.D.
Kingsley A. Reeves, Jr., Ph.D.
John Skvoretz, Ph.D.

Date of Approval:
May 3, 2012

Keywords: socially-aware system design, decentralized social data management, peer-to-peer network, projection graph, social network analysis

Dedication

To my beloved parents Panayiota and Ioannis Kourtellis for teaching me the importance of an education, and always motivating me to pursue my dreams.

Acknowledgments

I would like to thank Dr. Adriana Iamnitchi for being my major professor and academic advisor for the past six years. Her help and guidance inspired me to overcome any difficulties in my research, and her persistence motivated me throughout my doctoral studies. I would also like to thank Dr. Cristian Borcea, Dr. Miguel Labrador, Dr. Jay Ligatti, Dr. Kingsley A. Reeves Jr., and Dr. John Skvoretz for serving as my Ph.D. advisory committee, and Dr. Chris Ferekides for serving as the chair in my Doctoral defense.

I am obliged to my colleagues from the Distributed Systems Lab for their support. In particular, I would like to acknowledge the significant contribution of Josh Finnis on the peer communication/gateway module of Prometheus and of Paul Anderson on the peer geo-social management module of Prometheus, and Jeremy Blackburn's contribution in the study of the Prometheus' resilience.

I would also like to show my gratitude to the CSE technical staff, as well as the technical support from the PlanetLab testbed, for their help during the testing of the Prometheus system and the social network experiments on projection graphs.

Finally, I would not be able to finish this dissertation without the support, motivation and guidance of my parents Panayiota and Ioannis, my brothers Achilleas and Adonis, my relatives and all my friends, who were always there to help me during my Doctoral studies, here at the University of South Florida.

Table of Contents

List of Tables

List of Figures

Abstract

Social media services and applications enable billions of users to share an unprecedented amount of social information, which is further augmented by location and collocation information from mobile phones, and can be aggregated to provide an accurate digital representation of the social world. This dissertation argues that extracted social knowledge from this wealth of information can be embedded in the design of novel distributed, socially-aware applications and services, consequently improving system response time, availability and resilience to attacks, and reducing system overhead. To support this thesis, two research avenues are explored.

First, this dissertation presents Prometheus, a socially-aware peer-to-peer service that collects social information from multiple sources, maintains it in a decentralized fashion on user-contributed nodes, and exposes it to applications through an interface that implements non-trivial social inferences. The system's socially-aware design leads to multiple system improvements: 1) it increases service availability by allowing users to manage their social information via socially-trusted peers, 2) it improves social inference performance and reduces message overhead by exploiting naturally-formed social groups, and 3) it reduces the opportunity of attackers to influence application requests. These performance improvements are assessed via simulations and a prototype deployment on a local cluster and on a worldwide testbed (PlanetLab) under emulated application workloads.

Second, this dissertation defines the projection graph, the result of decentralizing a social graph onto a peer-to-peer system such as Prometheus, and studies the system's network

properties and how they can be used to design more efficient socially-aware distributed applications and services. In particular: 1) it analytically formulates the relation between centrality metrics such as degree centrality, node betweenness centrality, and edge betweenness centrality in the social graph and in the emerging projection graph, 2) it experimentally demonstrates on real networks that for small groups of users mapped on peers, there is high association of social and projection graph properties, 3) it shows how these properties of the (dynamic) projection graph can be accurately inferred from the properties of the (slower changing) social graph, and 4) it demonstrates with two search application scenarios the usability of the projection graph in designing social search applications and unstructured P2P overlays.

These research results lead to the formulation of lessons applicable to the design of socially-aware applications and distributed systems for improved application performance such as social search, data dissemination, data placement and caching, as well as for reduced system communication overhead and increased system resilience to attacks.

Chapter 1: Introduction

The dramatic increase in the number of computing devices used by billions of people from around the world allows individuals to share with each other their location, collocation with others, daily schedules, personal preferences such as in dining, movies, and music, hobbies and other social activities such as sports, games, etc., through numerous social applications and services (Figure 1.1).

Figure 1.1: Users share a wealth of social information over a variety of social applications and services.

Social information collected by such applications and services include declared relationships (e.g., friendships on Facebook or membership in LinkedIn groups), as well as inferred social relationships (e.g., users "like" the same video on YouTube or play online games together) and location of their users. Typically, users are represented as nodes in a social graph and are connected with each other by a social edge.

Socially-aware applications and services, by definition, exploit user relationships to provide enhanced functionality and better performance. For example, such services have

leveraged out-of-band social relationships for filtering restaurant recommendations based on reviews by friends (e.g., Yelp [Yel12]), recommending email recipients or filtering spam based on previous email activity [KRS+06], exploiting social incentives for computer resource sharing [TCL08, LD06], improving security in social networks [YKGF06, YGKX08], inferring trust in peer-to-peer storage systems [MRG+05], and building peer-to-peer overlays [PCT04] for private communication.

In addition, social relationships inferred from online social information have been used to rank Internet search results relative to the interests of a user's neighborhood in the social network [GMD06], to favor socially connected users in a BitTorrent swarm [PGW+08], and to reduce unwanted communication between users [MPDG08]. Social information has also been leveraged in conjunction with location and collocation data to provide novel mobile applications such as Loopt [Loo12], Foursquare [Fou12] and Latitude [Goo12a].

The common approach for building such social applications is the vertical integration, where one source of information is used to construct a social graph of users-nodes connected over application specific edges and used within the application bounds. Instead, combining declared and interaction-based social information from multiple sources can provide more accurate and personalized support for novel social applications covering a wide spectrum of domains. This can be achieved through an infrastructure that absorbs social information from an unrestricted set of domains, and can export it to an ever-evolving collection of socially-aware applications and services [IBK12].

At the heart of this infrastructure must be a persistent social knowledge management service, scalable with the number of users represented and the number of social sources providing input. Such a service should support a variety of requests from social applications through a basic API. Furthermore, depending on where users store their social information, the search workload imposed by applications can lead to socially-informed routing of requests within the computing system supporting the service.

This dissertation makes two main contributions. First, it presents the design and evaluation of such a social knowledge service, Prometheus. Second, it proposes the projection graph model to study the network properties that the service's computing nodes acquire during the mining of social information from applications.

The next two sections (Section 1.1 and 1.2) present the types of social information collected by Prometheus and the system architecture used to support and manage these social data. Section 1.3 expands on the network properties that system peers acquire during application traversals of the user social graph. Section 1.4 presents the thesis and extracts relevant research questions addressed through this dissertation. Section 1.5 summarizes the research contributions and Section 1.6 outlines the chapter structure of this dissertation.

1.1 Collection and Use of Social Information

Social information can be collected and managed within the context of an application, as in the examples presented earlier, or can be exposed from platforms such as *online social networks (OSNs)* (e.g., Facebook, Google+, etc.), which are specifically designed to collect and manage social information on user's behalf, and make it available to 3rd party online applications and services. In such networks, however, hidden incentives for users to have many "friends" can lead to declarations of contacts with little connection in terms of trust, interactions, common interests, shared objectives, or other such manifestations of real social relationships [GWH07]. For example, 90% of Facebook users perform 70% of their interactions with only 20% of their friends [WBS+09].

Alternatively, interaction-based social information provides an unprecedented level of detail compared to the binary declared relationships typical of OSNs. First, it provides the opportunity to quantify the strength of the social relationship based on domain-specific

metrics, such as quantity (e.g., phone call duration or number of characters in instant messaging exchanges) and frequency. Second, it conveys more accurate information than declared relationships, which are generously created and rarely removed. However, there are declared relationships that could not be removed, even if rarely supported by interactions, such as family relations or long lasting ties with close friends. Thus, a social application must wade through a lot of noise embedded in the collection of such social information in OSNs to provide targeted functionalities.

In this work, we argue that the combination of social information from diverse application domains can enable novel socially-aware applications. For example, a context-aware phone-call filtering application (e.g., CallCensor [KFA+10a]) may filter calls when caller *Bob* tries to reach callee *Alice*, based on 1) the declared professional relationship between them in LinkedIn, 2) the personal relationship between them in Facebook, 3) the phone call interactions between them, and 4) the current collocation of *Alice* with other individuals. A social knowledge service could store all these types of social information about *Bob* and *Alice* and allow the mobile application to query for a particular type of social edge connecting *Bob* and *Alice* within the social graph. Using this combined information, personal calls can be automatically silenced during professional meetings, but co-workers' or other professional-related calls are let through.

Therefore, such a social knowledge service should manage and expose to applications a combination of declared and interaction-based information from diverse social sources, as well as location and collocation information. This wealth of information can 1) lead to a more accurate inference of trust and incentives for resource sharing [KFA+10a, IBK12], 2) help identify social contexts, e.g., for geo-socially-aware data sharing when in a personal vs. professional context [KFIB09], and 3) enable novel classes of social applications [AKFI10, KFA+10a, IBK12].

1.2 Management of Social Information

The graph constructed from the social knowledge service could be stored and managed by a wide range of system architectures, as illustrated in Figure 1.2.

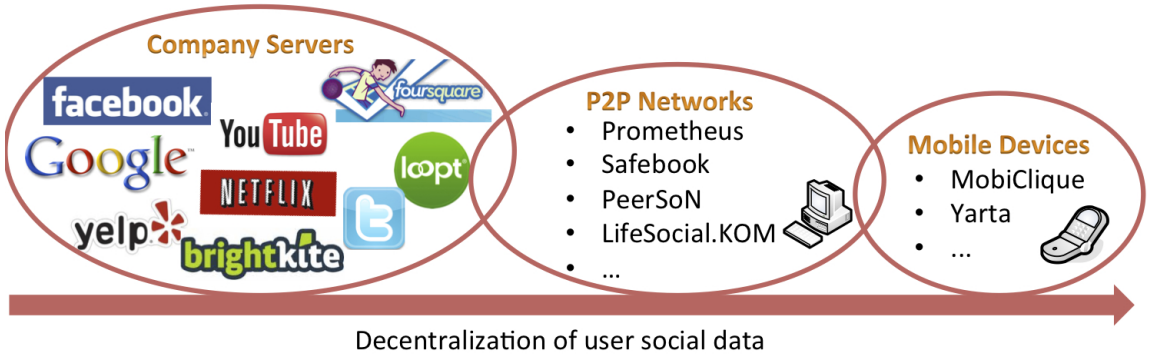

Figure 1.2: Decentralization of users' social information in different system architectures. A mobile device can typically hold only the user-owner's social circle. Centralized company servers have access to all users' social data. A P2P network node holds information about a small subset of users.

On one side of the spectrum, the social graph could be stored in a centralized way on company servers, and exposed to services and applications via APIs. However, the business model of a centrally administrated architecture (e.g., Facebook, Google, etc.) is typically based on selling users' data to 3rd party companies for advertising [NM09, Con12, Gun11]. Therefore, there are no incentives or appropriate business models to store social data for free and, at the same time, respect user privacy and allow users full control over their data. Additionally, users must trust their provider for complying with privacy and availability policies and not practicing "Big Brother" monitoring, which is especially important when it comes to aggregated collection of social information [Nis04, BDMN06].

Furthermore, some OSNs institute particularly draconian policies concerning the ownership of user-contributed information and content. For example, users cannot easily delete their OSN profiles (e.g., from Facebook servers); they cannot export their social data to a service of their choice in a transparent and easy way (e.g., from Facebook to

Google+); and the use of social information is restricted by the functionalities offered within the OSN that manages it. In summary, current OSN users depend on information collected and exposed by centralized OSNs and have to trade privacy and ownership of their data, as well as transparency in usage of their data by 3rd-party companies, for service availability and functionality.

On the other side of the spectrum, as shown in Figure 1.2, the users' social data could be managed by the users' mobile devices in a fully decentralized fashion [MMC09, POL+09, SRA10, TPI11]. Much of the social information is nowadays generated by mobile devices. However, they are inherently unsuitable for running a complex social service that combines social information from multiple sources, performs multi-hop inferences on multiple users' social data (not just the device's owner) and exposes these inferences to applications and services of many users. This is due to resource constraints: the mobile devices may not be always online or synchronized to support up-to-date inferences; and computational resources, and more importantly energy, are likely to be scarce.

In between these two extremes, there is a wide range of distributed solutions where groups of users have their social information stored on the same machine. Of the various distributed architectures, the peer-to-peer (P2P) architectural approach [BSVD09, SVCC09, CMS09, AR12, GGS+11, KFA+10a] has significant benefits over centralized systems and mobile devices. In the P2P approach, the social graph is divided into subgraphs and stored and maintained by the P2P system in a decentralized fashion across the user-contributed peers. Therefore, a social knowledge service that uses a P2P architecture can: 1) eliminate single points of failure and provide better user control over privacy of social data compared to centralized systems (e.g., Facebook) and, 2) provide better service availability and functionality for social applications mining the social graph than mobile devices.

1.3 System Traversal of Social Graphs

Regardless of the type of system architecture, the computing nodes (company servers, peers or phones) are "connected" with each other because of the social ties connecting the users-owners of the social data stored on each node.

When an application submits requests in the system to traverse the social graph, the requests will be forwarded between computing nodes in a manner informed by the way the graph is divided and stored in these nodes. In particular, the routing of requests between nodes can follow the social edges connecting users over multiple social hops. To study the properties of this *socially-informed* routing we use *projection graphs*. In these graphs, computing nodes are connected if users storing their data on different nodes are directly connected in the social graph.

Furthermore, during request routing, system nodes acquire particular network centrality properties because of the users' position in the social graph, and could be forced to serve a high (or low) load of requests. These load imbalances could be significant for centralized systems, but especially for decentralized systems such as P2P networks, where peers are typically less resourceful than company servers, are interconnected over high delay network connections and exhibit higher churn.

Consequently, studying the projection graph and the centrality properties of the P2P system that supports a social knowledge service such as Prometheus can reveal opportunities to inform the service's design for better data caching, data replication, or system load balancing. Furthermore, these properties can be used to inform the design of applications using the social graph, thus improving overall application and system performance.

1.4 Thesis and Research Questions

In general, a social knowledge service should: 1) be able to store and manage social information from multiple sources in the form of declared and interaction-based social edges, 2) distribute this aggregated information on a P2P system instead of a mobile-based or centralized system, for better user-controlled privacy, service availability and functionality, and 3) expose this aggregated information via an inference API, thus enabling novel socially-aware applications and services.

Furthermore, the decentralization of the service's social graph on multiple peers can influence the routing of queries in the system. Hence, studying the network centrality properties acquired by particular peers can reveal opportunities to inform the design and improve the performance of the social knowledge service and other socially-aware distributed applications and services.

These observations on the collection and management of social information via a social knowledge service, and the use of social information from applications and services lead to this dissertation's thesis:
Embedding social information in the design of a distributed social data management system leads to improved service availability and query response time, reduced system overhead and increased resilience to attacks.

This thesis raises a number of research questions which motivate the work presented in this dissertation:

- How can the system store and manage users' social data, collected from multiple sources, in a decentralized fashion?

- How can the system process social graph information on behalf of social application and service requests, while offering users privacy and access control to their data?
- How can the system take advantage of social knowledge from the user graph to enhance service scalability, availability and inference functionality?
- How does the topology of the decentralized social graph affect the routing of application queries in a social knowledge service?
- How does the degree of social data decentralization affect the network properties of nodes in such a service?
- How can a system or application designer use these node network properties to inform the design, and thus improve the performance, of distributed applications and systems?

1.5 Research Contributions

The research approach we followed to support this thesis can be grouped into two main parts. First, we designed, implemented and evaluated Prometheus, a large-scale distributed system for social data management. Our work on Prometheus resulted in the abstraction of several network properties that are applicable to many similar systems. This abstraction led to the *projection graph*, a generalized model of these distributed systems. Within this second part, we defined a model for the projection graph, an evaluation of its network properties and how they can be used to design more socially-aware applications and P2P overlay systems. The research contributions of this dissertation are described in more detail in the next paragraphs.

First, we design Prometheus [KFA+10a], a socially-aware P2P service, that manages user social information and exposes it to applications and services through an interface that implements non-trivial social inferences. This service collects social information from actual interactions between users within multiple environments (e.g., OSNs, email, mobile phones), and stores it on user-selected peers. Thus, it maintains richer and more nuanced social information than current OSNs, which can lead to more accurate inferences of trust, interests, and context. Prometheus represents social information as a directed, weighted and labeled social multi-graph distributed on a P2P network comprised of user-contributed peers. Access to social data by applications is controlled by user-defined policies.

Second, we demonstrate that Prometheus' socially-aware design increases service availability, improves social inference execution performance and enhances resilience to attacks. This is shown experimentally via simulations on real social graphs and via worldwide large-scale experiments on hundreds of machines on the PlanetLab testbed [Pla12].

Third, we define the *projection graph* (PG) [KI11] emerging from the decentralization of a social graph on a P2P system such as Prometheus, and study its network properties. We discover that within a range of social data decentralization on the P2P network, the projection graph inherits the network structure of the social graph it projects. We identify experimentally this range through the study of three classical network centrality measures: i) degree centrality, ii) node betweenness centrality, and iii) edge betweenness centrality, We investigate how these metrics in the projection graph correlate with the metrics of the social graph, while varying the degree of social data decentralization in the system (as seen in Figure 1.2).

Fourth, we empirically demonstrate how the projection graph centrality properties can be used to enhance application performance. We focus on social data search and experimentally show significant improvements on search success rate, as well as reduction of the

expected system overhead during the traversal of the social graph distributed on multiple peers.

Fifth, we empirically demonstrate how the projection graph centrality properties can be used to reduce overlay overhead in a P2P system that stores a social graph. We focus on unstructured P2P overlays and experimentally show how we can leverage the projection graph properties to construct an overlay that improves overall success rate in social search as well as reduces system overhead imposed by the application search queries.

1.6 Dissertation Outline

The outline of this dissertation is described in the following paragraphs.

Chapter 2 presents the details of the Prometheus' design. Chapter 3 examines experimentally Prometheus' performance under high-stress workloads from emulated applications on a local 10-node cluster and on a hundred machines on PlanetLab. Furthermore, it demonstrates the system's usability with CallCensor, a mobile social application which imposes real-life constraints. Chapter 4 discusses the resilience of a socially-aware system such as Prometheus to attacks from malicious users and peers.

Chapter 5 introduces the projection graph and presents a formal model for the emerging projection graphs in P2P systems such as Prometheus. Chapter 6 presents the analytical relations of the three social network metrics between projection and social graphs. Chapter 7 examines experimentally on real networks the association between projection and social graphs and estimation methods for the centrality metrics. Chapter 8 demonstrates the use of projection graph properties in the traversal of distributed social graphs at the application level, as well as for the overlay organization of unstructured P2P networks.

Chapter 9 presents a literature review in the main research areas covered by this dissertation. Chapter 10 discusses a set of lessons applicable to previous social-based P2P systems and provides guidelines for the design of future socially-aware distributed systems. Finally, Chapter 11 concludes this dissertation with a summary of the main findings and a discussion on future research directions.

Chapter 2: Decentralized Social Data Management with Prometheus [1]

Prometheus is a socially-aware P2P service that manages the social information of registered users in a decentralized fashion on user-contributed peers. Social information populates the Prometheus-managed social graph via social sensors. Access to social data is controlled by user-defined policies. This social information can be exposed to a wide range of applications and services through an API that implements non-trivial social inferences.

In order to better understand the functionality of the Prometheus service and how it supports novel socially-aware applications, we first identify in Section 2.1 its role in the *social hourglass infrastructure* proposed in [IBK12] In Section 2.2 we present the design objectives of this service and in Section 2.3 we present in detail the design characteristics that enable these functionalities.

2.1 Prometheus Overview

The social hourglass infrastructure (illustrated in Figure 2.1) consists of five main components: *social signals, social sensors, personal aggregators, social knowledge service* and *social applications*. Social signals are unprocessed user social information such as interaction logs (e.g., phone call history, emails, etc) or location and collocation information.

[1] Portions of this work have been previously published in [KFA+10a, IBK12] and are utilized with permission of the publisher.

User social sensors parse the users' social signals, analyze them and send processed social information to the personal aggregator of the user.

A user's personal aggregator combines social information from the user's sensors and produces a personalized output based on user preferences. This is the social input of a *social knowledge service (SKS)*. Based on this input, the SKS builds an augmented social graph which is decentralized on user-contributed machines. The social graph can be mined by applications and services through an API that implements social inferences. Prometheus fulfills the role of the SKS in the social hourglass infrastructure.

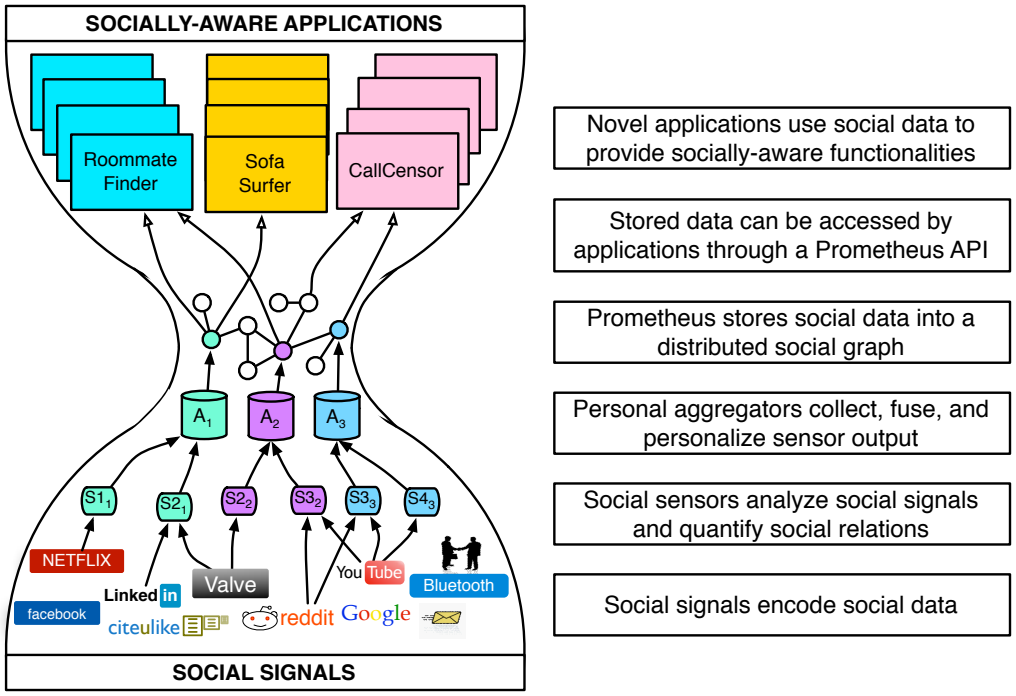

Figure 2.1: Prometheus in the social hourglass infrastructure. [IBK12]. ©2012 IEEE.

2.1.1 Social Input: Social Sensors and Personal Aggregators

Current socially-aware applications and services depend on what we refer to as *social signals*: information that exposes social *relationships* between people. A vast diversity of so-

cial signals already exist as byproducts of Internet- or phone-mediated interactions, such as email logs, comments on blogs, instant messaging, ratings on user-generated content, phone call history, or via face-to-face interactions determined from (GPS or Bluetooth-reported) collocation data. For example, a social signal could reflect the interactions of two users A and B over a soccer video posted on YouTube by user A. These interactions could reflect comments, "likes", other video sharing, friendship creation, etc.

Social sensors analyze social signals of users. Sensors are applications running on behalf of users on various platforms such as their mobile phone, PC, web browser, or trusted 3rd party services, and transforming the domain-specific interactions for a social activity into a weighted and labeled social edge.

Two types of social ties can typically be inferred from user interactions. The first type, *object-centric* ties, are 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, repeatedly being part of the same BitTorrent swarms, as well as ties inferred from recorded collocation traces [MLF+08] or personal conversations [LBBP+11]. The second type, *person-centric* ties, are determined from declared social relationships (e.g., in online social networks), or declared membership to groups (e.g., networks and groups in Facebook or LinkedIn).

Many such sensors already exist, although they may not output social ties as they have been implemented in different contexts and for different purposes. For example, these sensors record and quantify user activity in online social networks [LKG+08], co-appearance on web pages [MMH+06], or co-presence recorded as collocation via Bluetooth [EP06].

All sensors deployed on behalf of a user report social edges specific to each sensor's domain to the personal *aggregator* of each user, with the following format:

```
ego: <alter,label,weight>
e.g.,
```

```
A: <B, soccer ,0.1>        (from sensor reading YouTube signal)
A: <B, football ,0.2>      (from sensor reading NFL website signal)
```

The personal aggregator, a trusted application typically running on a user-owned device (laptop, desktop, mobile phone, etc), fuses multiple same activity social edges and personalizes information to user preference. The aggregator could perform sophisticated analysis on these edges, such as differentiating between routine encounters with familiar strangers and interactions between friends [EP06]. Weights can be assigned on each edge as a function of the number and frequency of interactions which allows for a more accurate representation of the relationship strength [XNR10]. Therefore, the aggregator incorporates all sensors' input for the same domain, and outputs social data corresponding to labeled and weighted directed edges for its owner and sent as input to Prometheus in the following format:

```
<ego , alter , label , aggregated  weight>
e.g.,
<A,B, sports ,0.15>
```

If the labels given by the social sensors are very domain-specific, the aggregator may apply more general labels to enhance data usability by the user's social applications. Also the weight can be personalized further by the user, who can prioritize sensor input based on importance for his social applications.

2.1.2 Prometheus as a Social Knowledge Service

The SKS provides a mechanism for storing and managing user social data and exposing them to applications and services. Prometheus, which fills-in the role of SKS in the social hourglass infrastructure, distributes the social information on multiple peers contributed by users and performs replication for better service availability and data durability. Furthermore, access to social information is subject to user-defined access control policies.

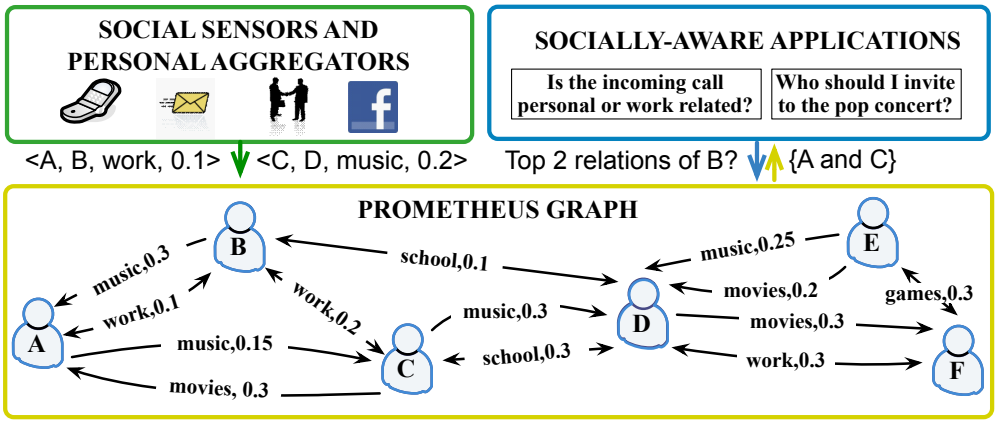

Figure 2.2: Prometheus input, social graph maintained, and output to applications. Social input from aggregators is stored as a directed, labeled and weighted multi-graph. Applications mine the graph via Prometheus API social inferences.

Figure 2.2 presents an overview of the Prometheus architecture. The information reported by the users' aggregators is processed by the service to create a weighted, directed, labeled, and multi-edged social graph, where vertices correspond to users and edges correspond to interactions between users as reported by their aggregators. The interactions are described with a label (e.g.,"work", "hiking") and a weight that specifies the intensity of the interaction.

2.1.3 Output to Applications and Services

Prometheus exposes an interface 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 her top relations over a particular label. 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 user-defined access control policies enforced by the trusted peers of the user-owner of the data. 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.

2.1.4 Prometheus in Use: SofaSurfer Application

A typical user application scenario is presented below for explaining how Prometheus is used in the context of the social hourglass infrastructure. Let us assume that user *Bob* installs a new application, *SofaSurfer*, that allows him to tap his social relations and identify who in his social circle to ask for hosting while on a low-budget road trip. At installation, the application checks with the *Bob*'s aggregator which of the required and optional social sensors he is registered with. Assume that *Bob* has accounts on Facebook, Skype, Google (and uses the chat utilities on all these platforms), LinkedIn, and is active on a Team Fortress 2 (TF2) game server, and all the corresponding social signals are observed by previously deployed social sensors running on *Bob*'s behalf.

These sensors consequently report *Bob*'s activity to Prometheus, subject to a personalized filter stored and applied by his aggregator. For example, the instant messaging activity is aggregated into a value recorded on Prometheus that gives more weight to Google Chat than to Skype chat activity, the latter being mainly used for work interactions. This personalization filter can be updated rarely—due to significant changes in activity patterns or to new social sensors deployed—or can be left to a default setting that weighs equally all signals.

SofaSurfer will query Prometheus for a list of *Bob*'s social contacts that are geographically close. Consequently, Prometheus will 1) retrieve *Bob*'s 1-hop social neighborhood, 2) use *Bob*'s location to filter-out those not within his geographical proximity, and 3)

order the remainder of contacts based on a social strength with *Bob*. Among contacts can also be "friends of friends" (i.e., 2-hop social neighborhood), subject to user-specified application-related preferences. For example, the application might allow *Bob* to specify that friends of friends connected via Facebook friendships are trusted enough if also linked over direct TF2 interactions with *Bob* (i.e., they played together on TF2).

If not all necessary social signals are available for *Bob*, they will be identified when the application is first installed. An out-of-band service lists the various implementations of sensors and their social signals. *Bob* will be prompted to agree with the deployment of missing sensors. His aggregator, as his personal assistant, provides the credentials for these sensors (e.g., the Facebook password to access wall posts). Sensors are deployed on where each social signal is (e.g., as a Facebook application) or on user-controlled platforms (e.g., a TF2 sensor running on user's desktop). The cognitive load on the user is determined by the level of sophistication desired for social inferences, from default, one-size-fits-all settings to fully personalized.

2.2 Prometheus Design Objectives

Prometheus was designed to fulfill the following system objectives:

- Extensibility: the service should be independent of the different types of social information stored.
- Accessibility: the social information should be exposed to applications and services through an API that implements basic social inferences.
- Privacy and Access Control: users should be able to control where the social information is stored and what is accessed by applications and services.

- Availability: the service should be able to cope with peer churn and allow users to control the degree of availability of their data.
- Durability: the social information should not be lost due to peer failures, even if not always available for use.
- Scalability: the service should be scalable to thousands or even millions of users and peers.

2.3 Prometheus Design

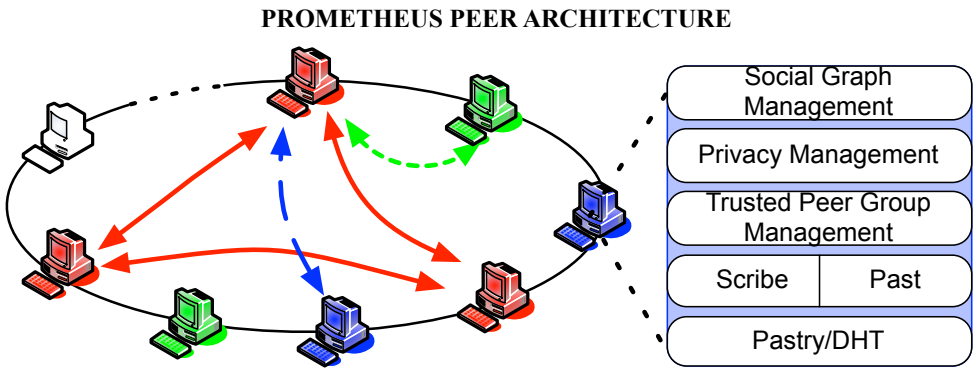

Figure 2.3: Overview of the Prometheus architecture. Prometheus peers organized using Pastry, a DHT-based overlay, Scribe, a DHT multicast infrastructure, and Past, a DHT storage system. Same color machines comprise a user's trusted peer group allowed to decrypt and mine the user's social subgraph. Continuous (*red*) arrows show communication for social graph and group maintenance. Prometheus inferences are executed between peers in a decentralized fashion (dashed arrows).

The Prometheus P2P network is organized using Pastry [RD01a], a distributed hash table (DHT)-based overlay (Figure 2.3). Prometheus uses Past [RD01b] for replicated storage of the social data, which can be stored encrypted at any peer. A registered user creates a group of *trusted* peers by selecting specific peers to manage her social data inserted by the user's aggregator. Only such a trusted peer selected by the owner of the social data can decrypt the social data stored on Past. This group of trusted peers allows improved

service availability of data decryption and processing for social inferences. The maintenance of the trusted peer group is done by leveraging Scribe [CDKR02], an application-level DHT multicast infrastructure.

Prometheus uses a public-key infrastructure (PKI) to ensure both message confidentiality and user authentication. All users have public/private keys for both themselves and their trusted peers accessing their social data. For access control purposes, users are identified by their personal public keys. These keys are used by the aggregators to encrypt and sign all data submitted to the service. Also, applications and services use them for secure communication with Prometheus when requesting and receiving social information. In the future we plan to enhance the security of communication between the various entities involved in Prometheus using the Transport Layer Security protocol [TD08]. To accomplish the various Prometheus functionalities, each peer currently runs three components: 1) for social graph management, 2) for privacy management, and 3) for trusted group management. Applications submit requests to mine the social graph via a distributed interface. Answers to these requests are subject to user-specified access policies.

Prometheus' design leverages user social relations to increase service availability and improve social inference performance. In particular, social awareness is embedded in the design of the service 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 [TCL08, LD06] and consequently increasing service availability for their friends' social inference requests.

Second, socially-related users are likely to 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 enables collocation of neighboring users' data on the same peer(s), leading to scalability and reduced replication cost [PES+10]. Therefore, complex social inferences over several social graph hops can be fulfilled locally,

thus reducing application response time. At the same time, user data collocation reduces the dependability on many peers and thus reduces the opportunity of malicious peers to influence service requests, as discussed in Chapter 4. In the next subsections we present the design details of Prometheus.

2.3.1 Building Blocks: Pastry, Past and Scribe

Prometheus leverages Pastry-based systems as building blocks for several of its P2P functionalities. Pastry [RD01a] is a scalable substrate for peer-to-peer applications, which facilitates request routing between peers and deterministic placement and retrieval of objects in the system. Peers participating in a Pastry-based overlay form a decentralized, self-organizing and fault-tolerant network. Pastry is used in Prometheus to organize the network of peers into a highly scalable DHT overlay.

When joining such an overlay, peers acquire a unique, uniform random ID from a circular 128-bit ID space (typically the cryptographic SHA-1 hash of its IP address or public key). Application-specific messages sent between peers are represented by their own unique ID, depending on the application specific functionality. When presented with such a message, a Pastry peer efficiently routes it to the peer currently live and with peer ID numerically closest to the message ID.

The expected number of steps the message is forwarded in the overlay is less than $\lceil log_{2^b} N \rceil$ under normal operation, for a network size of N live peers (b is a configuration parameter with typical value 4). Also, eventual delivery of the message is guaranteed, unless $\lfloor l/2 \rfloor$ peers with *adjacent* IDs fail simultaneously (l is a configuration parameter with a typical value of 32). To achieve these guarantees, each peer maintains a routing table with $\lceil log_{2^b} N \rceil \times (2^b - 1) + l$ entries (i.e. peer IDs pointing to particular peer IPs). For example,

with $b = 4$, $l = 32$ and $N = 10^8$ peers, a routing table of a peer would contain on average 137 entries and the expected number of routing hops would be 7.

In order to minimize the network distance travelled by a message while being forwarded to the closest peer ID, Pastry utilizes network proximity of peers (e.g., ping delays) to choose the network-wise closest peer for each forwarding step. During the routing of the message from peer to peer, the application instance of each peer which is responsible for that type of message is notified and can perform application-specific computations or take application-specific actions, relevant to the message received.

Past [RD01b] is a P2P scalable storage system that stores and retrieves files in a network of peers using the Pastry DHT overlay. Prometheus uses Past to store files containing the encrypted social data of users. In Past, a peer computes the quasi-unique file ID with a 160-bit hash of the file's name, owner's name or public key and a random salt, and uses the 128 most significant bits of this ID as a Pastry message ID for the file to be stored or retrieved. Replication within Past guarantees that the data will be available unless all replicas are lost (or the corresponding peers fail simultaneously and unexpectedly). Storage balancing performed in Past allows high utilization of the storage resources of peers. In addition, caching allows faster access to popular files and load balancing on peers.

Scribe [CDKR02], an application-level DHT multicast infrastructure is leveraged for the maintenance of the trusted peer group of each user. Scribe allows peers to join highly dynamic publish/subscribe groups and is implemented on top of Pastry. As mentioned earlier, peers trusted from a user to manage her social data join her trusted peer group and respond to inference requests from applications. In Scribe, a peer computes the message ID (i.e., the group-topic ID), using the hash of the topic's name. It then publishes the message using this ID to the rest of the peers subscribed to the particular group,

by sending it through the group *multicast* peer tree implemented on top of the Pastry routing protocol.

2.3.2 User Registration

A user registers with Prometheus from a trusted device (e.g., their PC) by creating a uniform random UID from the circular 128-bit ID space of the DHT (typically the hash of her public key). At registration time, she specifies the peer(s) she owns (controls) and willing to contribute to the network (if any). Through her trusted device, she creates a mapping between UID and the list of these peers' IP addresses and signs it with her private key for verification. By contacting any peer in the network to momentarily join the DHT ring, the user stores this mapping in the network as the key-value pair $UID = \{IP_1, ..., IP_n\}$. When one of these peers returns from an offline state, it updates the mapping with its current IP address.

A user selects, deploys, and configures the social sensors she wants to use via her social aggregator (as described in [IBK12]). She may declare particular social relationships, such as family relations, that are difficult or impossible to infer by social sensors. She compiles a list of other Prometheus users with whom she shares strong out-of-band trust relations and searches the DHT storage for their machine mappings (using their UIDs). From the returned list of peers owned by these users, she selects an initial set of peers to comprise her trusted peer group.

Selecting a large set of trusted peers increases the service availability of the particular user. At the same time it may decrease the consistency of social data maintained across all trusted peers of that user, and furthermore may decrease the overall system performance due to unnecessary redundancy. As social information about a new user will be incorporated in the social graph, a user may be prompted with different choices for trusted

peers (e.g., peers belonging to users with stronger ties than owners of her current trusted peers).

2.3.3 Trusted Peer Group Management

Each user has a dedicated group of trusted peers that she can expand or contract based on application need for availability of her data, and trust in the peers' owners. Therefore, three issues concerning the trusted peer group management of a user are important: (a) group membership, (b) search for trusted peers, and (c) group churn.

A user can add peers in her trusted peer group by initiating a secure three-step handshake procedure to establish a two-way trust relationship between her and the peer owner. During this handshake process, the following steps take place: 1) the owner of the social data sends an invitation to the owner of the peer, 2) if the peer-owner trusts the data-owner not to be malicious, it replies with an acceptance message, 3) upon acceptance of the invitation, the data-owner sends to the peer owner the group keys to enable the peer to join her trusted peer group.

The group keys are transferred to the peer using an *Encrypt/Sign/Encrypt* process: 1) encrypting the group keys with the public key of the peer's owner, 2) adding a plaintext on the ciphertext referring to the peer owner and signing the whole message with the group owner's private key, and 3) re-encrypting all the above with the public key of the peer's owner. Upon receiving of the group keys, the new peer subscribes to the Scribe trusted group of the user. The Scribe group's handle is the concatenation of the predefined string "Trusted_Peer_Group" and the user's *UID* and can be used to publish signed messages to the group using the Scribe multicast protocol.

A user may decide to remove a peer X from her trusted group, if she no longer trusts the peer's owner or she wants to add more stable peers in her group. To this end, the user

creates new group keys and distributes them via *unicasts* to the rest of the peers that are still trusted using the previously described *Encrypt/Sign/Encrypt* process. She also re-encrypts her social data and replaces the copy in the DHT storage for future use. This removal of the peer is *multicasted* to all group peers and peer X is unsubscribed from the group. The distribution of new keys and re-encryption of the social data disallows the newly removed peer from decrypting updated versions of the user's social data in the future.

A peer owner may also decide to remove her peer from a trusted group of another user (e.g, due to overload on the peer or malicious activity from the data owner). This request is *multicasted* to all the other group peers, who alert the data owner and execute the same procedure as above. If a peer becomes untrusted while offline, a trusted peer from the group alerts the returning peer of the change and unsubscribes it from the Scribe group.

Service requests for a user (UID) can be sent to any peer, but only the user's trusted peers can provide data about her. Therefore, a random peer can find a user's trusted peers by submitting a *multicast* request with handle *Trusted_Peer_Group_UID*. With the multicast, all online group trusted peers are required to respond with their IP and signed membership, which is verified for authenticity with the user's group public key. The random peer creates a trusted peer list (TPL) of IPs based on peer responses.

The multicast (instead of using *anycast* for just one peer) allows the random peer to have peer alternatives in case of churn or erroneous communication with the first responding peer. Since the search for trusted peers follows the DHT routing, the responds are typically sorted in the TPL by network proximity (latency). The peer, upon creating the TPL for a user's group, can communicate directly with the individual trusted peers, preferably the network-closest one (i.e., with shortest latency). Prometheus caches the TPL after the first access and updates it when the trusted peers are unresponsive, changed

their IP, or became untrusted. Users could also apply their own policies for refreshing the local TPLs based on their usage patterns (e.g., daily).

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. However, we ensure that generated data (i.e., input from social aggregators) are not lost while a user's group is down via storage and replication in the DHT by Past.

2.3.4 Distributed Social Graph

Prometheus represents the social graph as introduced in [AKFI10, And10]: a directed, labeled, and weighted multi-edged graph, maintained and used in a decentralized fashion, as presented in Figure 2.4.

Multiple edges can connect two users, and each edge is labeled with a type of social interaction and assigned a weight (a real number in the range of $[0, 1]$) that represents the intensity of that interaction. The labels for interactions and their associated weights are assigned by the personal aggregator of each user. From an application point of view, distinguishing between different types of interactions allows for better functionality. The latest known location of a user and an associated timestamp are also maintained as an attribute of the user's vertex in the graph. We chose to represent the graph as directed and weighted because of the well-accepted result in sociology that ties are usually asymmetrically reciprocal [Wel88]. Representing edges as directed also limits the potential effects of illegitimate graph manipulation such as spamming. We elaborate in more detail on this issue in Chapter 4.

The social data for each user UID are stored in Past after the $Encrypt/Sign/Encrypt$ process is applied, i.e., encrypt with the public key of the user's trusted group, add plaintext referring to the group and sign with the user's private key and re-encrypt with group's

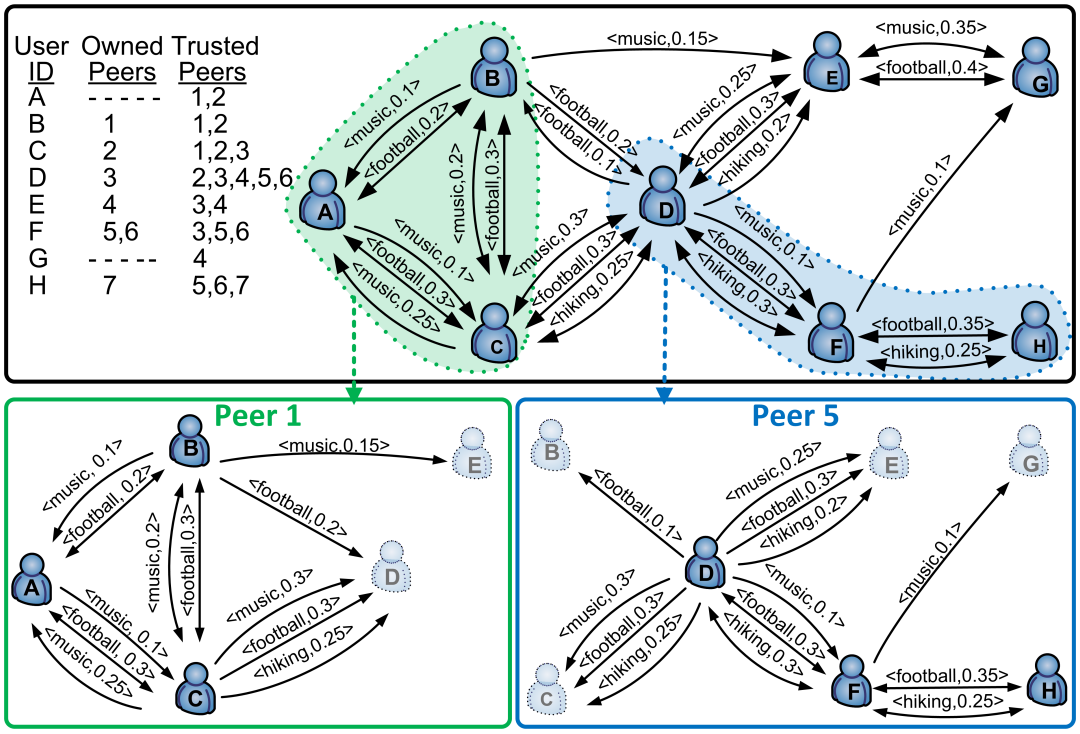

Figure 2.4: An example of a social graph for eight users (A-H) distributed on seven peers. The top figure shows the mapping between users, peer owners, and trusted peers (upper left corner) and how users are connected with each other over social edges, each marked with its label and weight. The bottom figures illustrate the subgraphs maintained by peers *1* and *5*. Users in dark color (e.g., A, B, C on peer *1*) trust the peer to manage their social data. Users in light-shaded color (e.g., E, D on peer *1*) do not trust the peer but are socially connected with users who do.

public key. The data are stored in the append-only file $Social\_Data\_UID$ as encrypted records. Only the user's aggregator can append records in her file and only trusted peers can decrypt and use these records. Since the file is append-only, readers (trusted peers) can access it at any time: in the worst case, they will miss the latest update. We designed Prometheus to be oblivious to the number and different types of social activity reported by the social sensors/aggregator, thus allowing extensibility.

Personal aggregators 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 local 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 [Gol07].

Edges may "decay" over time if few (or no) updates for those edges are received due to reduced number and frequency of social interactions associated with those labels [RD10]. This aging process should be activity specific, but it should also reflect the user's social habits and interests: users who are less socially active and users with a great number of friends should have their relationships age slower. Currently, the system applies a simple aging function to reduce an edge's weight by 10% for every week the two users do not interact over the particular label (thus, the connection never completely disappears and the aging happens slowly). A user's aggregator may specify a different decrement value of the weight and the time period for aging (these values are also stored in the $Social\_Data\_UID$ file for each user).

2.3.5 Social Inference API

Prometheus exposes to applications an API of basic social inference functions that are executed in a decentralized fashion; more complex inferences can be built on top of this set.

The boolean function *Relation_Test(ego, alter, α, x)* checks whether *ego* is directly connected to *alter* 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.

The function *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.

Preliminary experiments (shown in [KFA+10a]) revealed volatility of the peer-to-peer communication and long response delays during multi-hop inference execution on a world-wide testbed. Thus, the following API functions were designed to offer better quality of service to applications by allowing them to define, not only inference-specific parameters (such as label and weight), but also a timeout parameter T, which declares the application waiting time per hop.

The function *Neighborhood(ego, α, x, radius, T)* returns the set of users in *ego*'s social neighborhood who are connected through social ties of a label α and minimum weight of x within a number of social hops equal to *radius*. The *radius* parameter allows for a multiple hop search in the social graph (e.g., setting *radius* to 2 will find *ego*'s friends of friends). Our CallCensor mobile phone application which silences *ego*'s cell phone during meetings at work (Section 3.5) uses this function to determine if a caller is in *ego*'s work neighborhood in the social graph even if not directly connected.

The function *Proximity(ego, α, x, radius, distance, timestamp, T)* is an extension of the neighborhood function which filters the results of the neighborhood inference based on physical distance to *ego*. After the location information is collected for *ego* and the function *neighborhood* returns a set of users, *proximity* returns the set of users who are within *distance* from *ego* and their location information is at most as old as *timestamp* (assuming synchronized clocks with online time servers). Users who do not share their location or have location information older than the *timestamp* are not returned. A mobile phone

application might use this function to infer the list of collocated coworkers within a certain distance of *ego*.

The function *Social_Strength(ego, alter, T)* 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. However, we limit the indirect path length connecting the two users to two hops, using a well-accepted result in sociology known as the "horizon of observability" [Fri83], where two individuals are unlikely to have any meaningful social relationship or being aware of each other's work if connected over more than two social hops. 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. Next, we elaborate in more detail on how this function is calculated on the social multi-graph.

Assume that $\Lambda_{i,j}$ is the set of labels of edges connecting users i and j and $w(i, j, \lambda)$ is the weight of an edge between users i and j over a label λ. J_i is the set of directly connected neighbors to i. Then $NW(i, j)$ is the overall normalized weight between two directly connected users i and j:

$$NW(i,j) = \frac{\sum\limits_{\forall \lambda \in \Lambda_{i,j}} w(i,j,\lambda)}{\max\limits_{\forall j \in J_i} \Big(\sum\limits_{\forall \lambda \in \Lambda_{i,j}} w(i,j,\lambda) \Big)} \tag{2.1}$$

Also, assume that $P_{i,m}$ is the set of different paths $p \in P_{i,m}$ joining two indirectly connected users i and m. Using the results from [Fri83], we limit the indirect paths to 2 social hops and also take into account the number of indirect paths connecting users i and m. Then $S(i, m)$ is the return value for social strength between users i and m, over a multi-level 2 hop path:

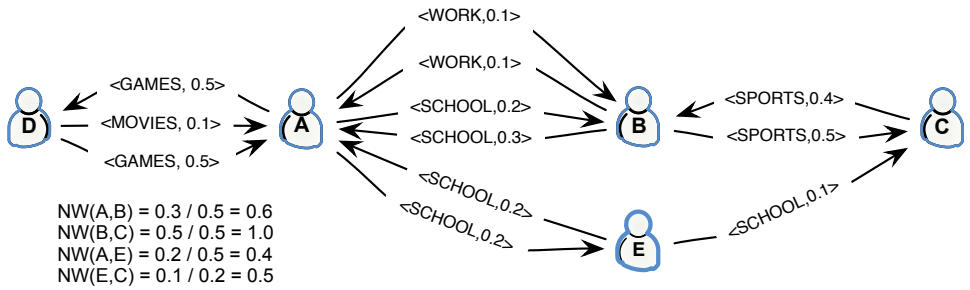

Figure 2.5: A calculation example of the social strength inference function.

$$S(i,m) = 1 - \prod_{\substack{\forall p \in P_{i,m} \\ \forall j \in J_i \cap J_m}} \left(1 - \frac{\min\{NW(i,j), NW(j,m)\}}{2}\right) \tag{2.2}$$

In Figure 2.5 we demonstrate on a weighted, directed multi-graph the calculation of the social strength between users A and C, over 2-hop paths. Using the normalized weights NW between all involved users (shown below the graph), we can calculate the social strength between A and C:

$$S(A,C) = 1 - \left\{\left(1 - \frac{min\{NW(A,B),NW(B,C)\}}{2}\right)\left(1 - \frac{min\{NW(A,E),NW(E,C)\}}{2}\right)\right\}$$
$$S(A,C) = 0.44$$

Such a function could be used, for example, to estimate social incentives for resource sharing. Supported by the results and discussion in [Fri83], we expect the expression 2.2 of the *social_strength* request to be an accurate quantification of the tie strength between two indirectly connected users. However, more qualitative studies are needed to establish the user-perceived quality of the function's outcome.

2.3.6 Access Control Policies

Users can specify access control policies (ACPs) upon registration and update them any time thereafter. These policies, stored on each of the user's trusted peers, are applied each time an inference request is submitted to one of these peers to access social information for the particular user. For availability, the policies are encrypted with the group public key, signed with the data owner's private key, re-encrypted with the group public key and stored in the DHT, allowing a rejoining trusted peer to recover updated policies. The same mechanism used for updating the social graph is used to update policies. As future work, we plan to investigate the provision of strong consistency and conflict resolution for policies.

Currently, we consider ACPs that are comprised of two parts: (1) the social data object(s) to be accessed and (2) the specification(s) to be met before access is granted for the particular data object(s). They are defined as entries of the ACP_UID file in the following format:

<Social Data Object(s)> :: <ACP Specification(s)>

In Table 2.1 we present a list of access control policy definitions comprised of these two parts.

Table 2.1: Access control policy definitions for a user in Prometheus.

Social Data Object(s)	ACP Specification(s)
Social edge label α	Social distance ρ
Social edge weight χ	Social edge label γ
User location Λ	Social edge weight y
	Originator user B
	Originator peer P
	Intermediate user C
	Intermediate peer M
	Application S
	Originator's location λ

Following is a list of social data objects for which a user UID_1 (owner of the information) can allow access to an inference request originating from user UID_2:

- Social edge label α:
  A policy to control access to social edges of user UID_1 with other users, over label α, regardless of the weight.

- Social edge weight χ:
  A policy to control access to social edges of user UID_1 with other users, that have minimum weight χ, regardless of the label type.

- User location (Λ_1, Λ_2, Λ_3):
  A policy to control access to user UID_1's last stored location with latitude Λ_1, longitude Λ_2 and altitude Λ_3.

By design, ACPs are whitelists. To specify who is allowed to access these categories of social information, user UID_1 can set the following access policy specifications:

- Social distance ρ:
  User UID_2 must be within a maximum distance of r hops in the social graph from user UID_1.

- Social edge label γ:
  User UID_2 must be connected with user UID_1 over a social edge of label γ (directly or indirectly).

- Social edge weight y:
  User UID_2 must be connected with user UID_1 over a social edge of weight y, regardless of the label type.

- Originator user B:
  The inference request must originate from user B (=UID_2).

- Originator peer P:

  The inference request must originate from peer P.

- Intermediate user C:

  The Inference request must be forwarded from user C (as part of a multi-hop inference request).

- Intermediate peer M:

  The inference request must be forwarded from peer M (as part of a multi-hop inference request).

- Application S:

  The inference request must come from application S.

- User location $(\lambda_1, \lambda_2, \lambda_3)$:

  The inference request must originate from a user UID_2 with current location latitude λ_1, longitude λ_2 and altitude λ_3.

Each of these access policy specifications can be placed to fulfill the requirements to access any of the data objects stated earlier. These specifications are meant to establish the minimum rights needed from a request to access a particular piece of social information of the user-owner of the social data. Each of the data objects, as well as each of the specifications, can be combined with logical operators such as *AND*, *OR*, *NOT*, etc, to create more complex access policies. A default set of access policies for user *Bob* is shown in Figure 2.6.

$< \alpha_1 >::< B = Bob$ AND $P = Bob's\ peer >$
$< \alpha_2 >::< B = Bob$ AND $P = Bob's\ peer >$
...
$< \alpha_m >::< B = Bob$ AND $P = Bob's\ peer >$
$< x = 0 >::< B = Bob$ AND $P = Bob's\ peer >$
$< \Lambda_1, \Lambda_2, \Lambda_3 >::< B = Bob$ AND $P = Bob's\ peer >$

Figure 2.6: Example set of default access control policies of user *Bob*.

To verify the access rights, Prometheus may call its inference functions, when applicable. For example, to detect whether the originator of the request is within N-hops, the originator is checked against the result of a N-hop *neighborhood* inference. ACPs also allow for blacklisted users (and their peers) for convenience. These users and peers are blocked either because the owner of the social data doesn't want to share any social information with them, or the system has flagged them as malicious.

Figure 2.7 shows an example of the set of access control policies for user *Bob*. By defining the policy for label *work* with the specific requirements, *Bob* allows *work*-related social information to be given to requests coming from users within 2 social hops (i.e., friends and friends-of-friends), that are also connected with him over *work* label, or when these requests come in from the *CallCensor* application. Also, he allows his parents and his brother to access his exact location at any time. However, everybody else must be in the same approximate area with him (within a difference δ) to retrieve *Bob*'s location. If a neighborhood inference request on the *tango* label is submitted to *Bob*'s trusted peer, Prometheus checks his ACP in the order *Blacklist*→*labels*→*weights*. User *Alice* is excluded from all types of inferences and cannot receive any information about *Bob*.

$< \alpha = hike >::< \rho = 2 \text{ AND } \gamma = hike \text{ AND } y = 0.2 >$
$< \alpha = tango >::< \gamma = tango \text{ OR } \gamma = salsa >$
$< \alpha = work >::< (\rho = 2 \text{ AND } \gamma = work) \text{ OR } S = CallCensor >$
$< \alpha = school \text{ AND } \chi = 0.2 >::< (\rho = 1 \text{ AND } \gamma = school) \text{ OR } y = 0.25 \text{ OR } S = CallCensor >$
$< \chi = 0.3 >::< \rho = 1 \text{ AND } S = SofaSurfer \text{ AND } (C = Charles \text{ OR } C = Dane \text{ OR } C = Eve) >$
$< \Lambda_1, \Lambda_2 >::< \lambda_1 = \Lambda_1 \pm \delta, \lambda_2 = \Lambda_2 \pm \delta >$
$< \Lambda_1, \Lambda_2, \Lambda_3 >::< B = mom \text{ OR } B = dad \text{ OR } B = brother >$

$< blacklist >::< B = Alice \text{ OR } B = Gary \text{ OR } C = Alice >$

Figure 2.7: Example set of access control policies of user *Bob* in Prometheus.

2.3.7 Decentralized Inference Function Execution

We assume that applications interacting with Prometheus cache a number of peer IP addresses to bootstrap the interaction. A social inference request for user D submitted from B (e.g., *Neighborhood(D, football, 0.2, 1 hop, 5 sec)* in the social graph of Figure 2.4) can be sent to any Prometheus peer (e.g., peer 1). The request is encrypted with D's trusted group public key, signed with B's private key and re-encrypted with D's group public key.

The receiving random peer creates D's trusted peer list (i.e., peers $2, 3, 4, 5$ and 6, as explained in 2.3.3) and forwards the request to the D's network-closest (shortest latency) trusted peer (e.g., peer 5), which decrypts the request, verifies the submitter's identity via his public key and enforces D's access control policies for B. If the request is allowed, the peer fulfills the request by traversing the local social subgraph for the information requested by the application and then returns the result (i.e., *users* C, E and F) to the application using an Encrypt/Sign/Encrypt process.

For functions that can traverse the graph for multiple *hops* h (e.g., *Neighborhood(D, football, 0.2, 2 hops, 5 sec*), the peer (peer 5) submits secondary requests for information about other users to their trusted peers, as follows. A secondary request includes the UID of the original submitter (B) along with the UID of the intermediary user (D) producing the secondary request, in order for the receiving peer to verify each user's access rights. A time period of $T * (h - 1)$ seconds is given to the secondary peers (e.g., peer 2 for user C and peer 4 for user E, and locally at peer 5 for user F) to respond with their results (peers at each hop use independent clocks for T seconds). The peer submitting the secondary requests uses the Encrypt/Sing/Encrypt process.

Each receiving secondary peer authenticates the request and checks the access control policies for the requesting user B as well as the intermediary user D. If the request is granted, the result is returned to the requesting peer. If the request still needs more in-

formation, that peer (e.g., at hop k) repeats the same process and submits a secondary request with an adjusted $T * (h - k - 1)$ timeout. Finally, the original requesting peer recursively collects all the replies and submits the final result to the application.

Even though Prometheus contacts peers in parallel in each network hop, there can still be variations on the communication time between peers of the same hop, leading to extended delays. In the future, we plan to tackle this problem with the following greedy optimization. Using the trusted peer lists of users, a peer preparing to send secondary requests for the next hop could calculate the minimum set of secondary peers needed to cover all users for the next hop. For the example used earlier, if peer 5 sent the secondary request to peer 3, all users could be traversed within the same peer. This optimization can decrease request delays by reducing the number of peers to be contacted at each hop and thus the variability of request execution. Furthermore, the submitting peer can select not only the minimum set of needed peers, but also the network-wise closest set of peers. Potential conflicts between peer entries can be resolved using utility functions to model the gains from each level of optimization.

The possible scenarios of inference execution for a request of h hops ($h \in H = \{1, 2, \dots\}$) are illustrated in Figure 2.8. TA defines the time for an application to send or receive a request to a local or remote peer. TR defines the time for a request or reply to be sent between remote peers over 1 network hop. If the application is not running on the local peer, then we can assume that $TA \simeq TR$. If it does, then we can assume that $TA \simeq 0$. TD defines the time for a peer to parse a request submission or reply and act accordingly (i.e., create secondary requests or back-forward the results). We assume that TA, TR and TD are constant and independent of the number of hops a request will traverse the system. The overall delay given a number of network hops r ($r \in R = \{1, 2, \dots\}$) involved in the execution scenario is defined as $d(r)$, with $r \leq h$. Assuming a request that traverses the social graph for h number of social hops, $F(r, h)$ defines the fraction of social paths of this request that will force peers to traverse the system for r number of network hops.

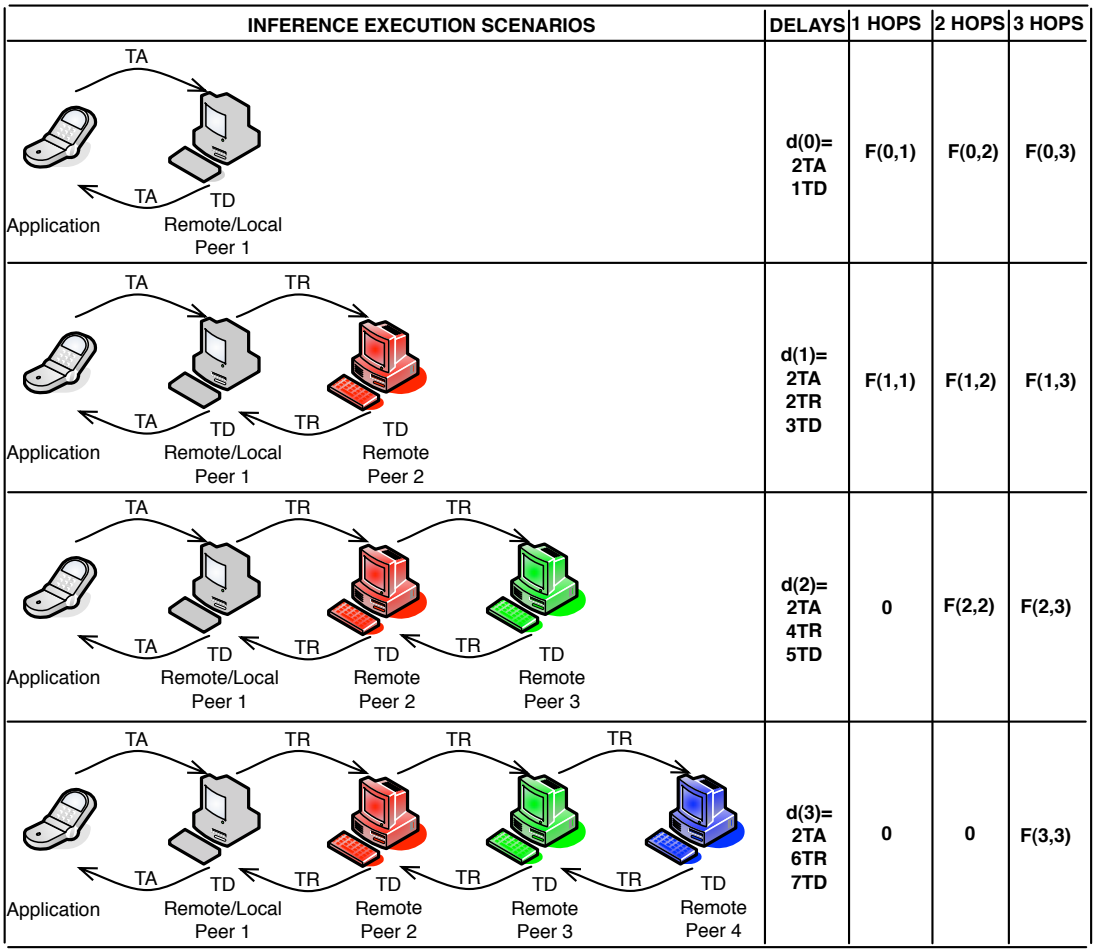

Figure 2.8: Possible scenarios of inference execution along with associated delays in the distributed infrastructure. TA defines the time for an application to send a request or receive a reply. TR defines the time for a request/reply to be sent between remote peers. TD defines the time for a peer to parse a request/reply. $F(r, h)$ defines the fraction of social paths of a request of h social hops that force peers to traverse Prometheus for r number of network hops. $d(r)$ defines the overall delay for a request that needs r number of network hops.

It would be tempting to model each inference execution scenario as a discrete-state Markov process with each network hop being a different state in the execution process. However, we cannot assume exponential distribution of 1) the arrival time of requests, 2) the arrival time of secondary requests, 3) the departure time of replies (service time). This is because the future state of the execution of an inference request highly depends on the

past states and what portion of the social graph traversal was fulfilled in each of these states. Therefore, the probability of transition from the i_{th} state (i.e. i_{th} network hop) to the next state $(i+1)_{th}$, i.e., $(i+1)_{th}$ network hop, or to the previous state $(i-1)_{th}$ is not constant and not exponentially distributed.

Using the notation of Figure 2.8, and given that the various secondary requests produced are executed in parallel, we can define the overall execution delay, $D(h)$, of a request of h social hops as follows:

$$D(h) = \sum_{r \in R} F(r,h)d(r) = \sum_{r \in R} F(r,h)(2TA + (2r+1)TD + 2rTR) \tag{2.3}$$

As mentioned earlier, an application request can enforce a timeout T for every social hop requested. Depending on how the social graph is decentralized on peers, this timeout may be enforced up to $h-1$ times, if $h = r$. For example, we notice that in the third scenario of Figure 2.8, peer 3 can lead peer 2 to a T timeout, and in the fourth scenario, peer 4 can lead peer 3 to a T timeout and peer 3 and/or peer 4 can lead peer 2 to T or $2T$ timeouts. Thus, the peer that will timeout can be at hop k, with $k < r$. We can assume that $Q(r,k)$ is the probability of a request of r number of total network hops to timeout at a particular peer at the network hop k. From the above, $Q(r,k) = 0$ for $k \geq r$. Using this probability, we can incorporate the timeout T in the equation 2.3 above, by adjusting appropriately the term $2rTR$ as follows:

$$D(h) = \sum_{r} F(r,h)\Big(2TA + (2r+1)TD + B\Big), where$$

$$B = \sum_{r} \Bigg(\sum_{k} Q(r,k)((r-k)T + 2kTR) + 2rTR\Big(1 - \sum_{k} Q(r,k)\Big)\Bigg) \tag{2.4}$$

2.3.8 Retransmission Policy

From our experience with Prometheus when deployed on a highly dynamic system such as PlanetLab [KFA+10a], we noticed that 10%–25% of the requests were dropped by remote peers, due to overloaded network interfaces, reduced computing resources, etc. To tackle this problem, we introduced in the decentralized inference execution a retransmission policy for sending messages (requests or replies) between peers. Under this policy, a peer can try to send a message to another peer up to three times, with each trial timing out at 1 second. Therefore, it can take from a little over 0 and up to 3 seconds for a message to be transmitted between two peers or permanently dropped. This policy helped reduce the drop rate (less than 5% drops were observed in our new experiments on PlanetLab), but also introduced an extra delay in the inference execution process. This delay forced the request end-to-end time to vary significantly but more predictably, depending on the number of message resends during the decentralized execution process.

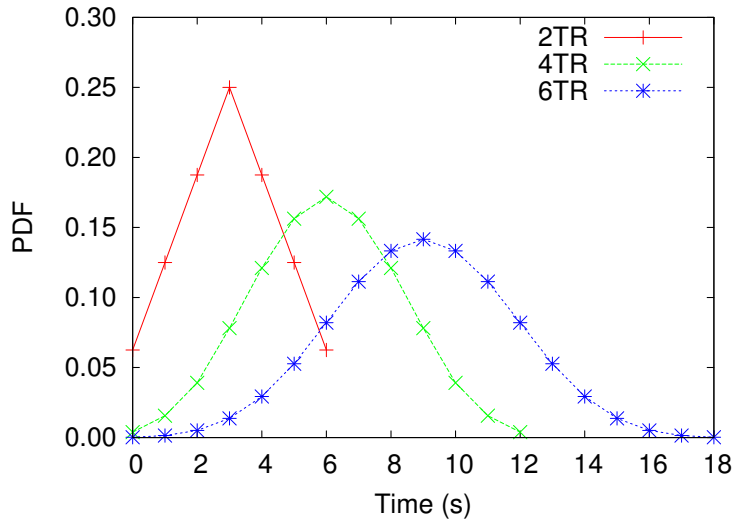

Figure 2.9: Probability distribution function of the time delay to send 2, 4 or 6 messages between peers, when the retransmission policy is used.

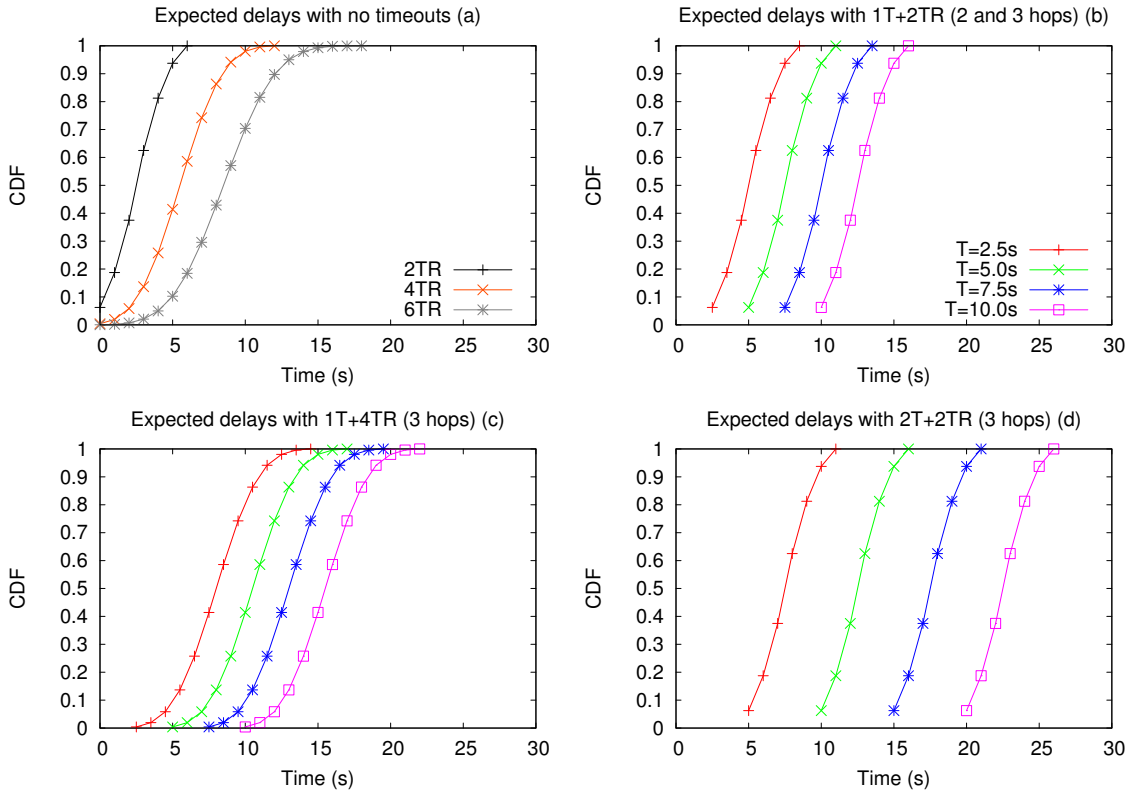

Figure 2.10: CDF of the time delay for a request with the following scenarios: (a) no timeouts, (b) 1 timeout and $2TR$ ($r = 2$, $k = 1$) , (c) 1 timeout and $4TR$ ($r = 3$, $k = 2$) and (d) 2 timeouts and $2TR$ ($r = 3$, $k = 1$), when the retransmission policy is used.

Figure 2.9 demonstrates how this delay varies probabilistically for the scenarios where a request involves 2, 4 or 6 messages between remote peers (i.e., $2TR$, $4TR$ or $6TR$ delays from equation 2.3), under all possible combinations of these messages and the times required to be transmitted. These are the simplest cases expected in inference requests of up to 3 social hops not including timeouts, assuming that the delays for computation and communication with the application are negligible in comparison to the network delays, i.e. $TD << TR$ and $TA << TR$ (as shown earlier in Figure 2.8 and in equation 2.3). We notice that an increase in the number of messages involved from 2 to 4 or 6, leads to an overall time delay normally distributed with an increasing mean from 3 to 6 or 9 seconds.

Figure 2.10 demonstrates the execution delay when we take into account the possible cases of timeout that could occur in the inference execution for 2 and 3 hop requests (as described in equation 2.4 and assuming $TD << TR$ and $TA << TR$) for timeout values $T = 2.5, 5.0, 7.5$ and 10 seconds. We observe that the expected delay for each scenario highly depends on the network hop that the timeout will occur (i.e., value of k in equation 2.4) and the number of network hops of the request (i.e., value of r in equation 2.4). Overall, the expected delay of a request will be a mixture of the delays shown in Figure 2.10, since an inference request could have different portions fulfilled in different peers, over multiple network hops (this was declared with the fraction $F(r, h)$), and with different probabilities for timeouts at each network hop (this was declared with the probability $Q(r, k)$).

Chapter 3: Performance Evaluation of Prometheus [1]

In this chapter, we present the experimental evaluation of the Prometheus performance in three different experimental setups. In the first two experiments, we deployed our prototype implementation of Prometheus on PlanetLab to test the system's performance in a real distributed environment under the same high-stress workloads, while varying inference request parameters such as number of social hops to be traversed, and timeout set by the application (Section 3.4). In the third experiment, we built a weighted multigraph from real collocation and Facebook traces and assessed the system's time responsiveness to social inference requests from a mobile application with real-time constraints (Section 3.5).

3.1 Implementation

Prometheus was implemented on top of FreePastry [Fre12b], a Java implementation of Pastry DHT [RD01a], which also provides API support for the functionalities of the components Scribe [CDKR02] and Past [RD01b]. The Keyczar library [Key12] was used for public/private key management and enforcement, as well as signing and authenticating messages when sent between peers or between application instances and peers. Furthermore, NetworkX library [Net12] was used for the social graph management.

[1] Portions of this work have been previously published in [KFA+10a] and are utilized with permission of the publisher.

We performed various optimizations and fine tuning from our earlier work [KFA+10a] to better handle the communication volatility and peer churn typically observed in a distributed infrastructure, as explained in Section 2.3. We also redesigned the API as it previously required applications and peers to exchange string-formatted messages with no extensibility on the number of request fields. The new API allows applications to submit a serializable class-based request using JSON [JSO12], by defining within the class object not only inference-specific parameters (such as weight and label) but also the new time-out parameter T, which declares the application waiting time per hop when requesting multi-hop inferences. Furthermore, applications and peers can define parameters such as request id and timestamp, error flags, etc. Overall, the new API design enhances extensibility for future inference parameters, portability across different computing platforms, and improves the peer-to-peer and peer-to-application communication during inference execution.

3.2 Testbeds

The Prometheus prototype was first deployed and extensively tested on the local cluster of the Distributed Systems Group (USF), which at the time was comprised of 10 servers with quad-core CPUs Intel(R) Xeon(R) x3220, 2.40GHz, and 4GB RAM and interconnected via a Gbit Ethernet switch. Later, the Prometheus prototyped was deployed and evaluated on the PlanetLab (PL) testbed [Pla12].

PlanetLab is a network of more than a thousand machines with a wide range of computing capabilities, contributed by more than 500 research sites around the globe, and used for research on distributed systems and networking at a planetary-scale. Each machine runs PLC, a Linux-based operating system that supports distributed virtualization. Different projects are allocated "slices" on the hardware and software of each machine, enabling them to deploy and test distributed services at the same time.

3.3 Experimental Setup for Performance Evaluation with Emulated Workloads

For the end-to-end performance and completion rate experiments under high-stress loads, we decentralized a social graph on selected PL machines and used emulated application workloads produced from studies on real social networks. The three metrics used in this evaluation were: (1) *end-to-end response time* to quantify the performance as perceived from an application point of view, (2) *percentage of completion* to quantify the tradeoff between request response time and level of request completion, and (3) *number of network messages* to quantify the service overhead.

3.3.1 PlanetLab Deployment

As mentioned earlier, the PL machines are shared among multiple research projects with variable workloads and daily patterns. Therefore, their highly heterogeneous network, computation and memory resources were scarce at times throughout our experiments. To increase the number of stable and resourceful PL machines included in the experiments, we used the CoMon tool [PP12] provided by the PL organization and polled the status of the machines before each run of our experiments using the metrics shown in Table 3.1. These strict status metrics reduced the potential set of peers to about 150. However, from previous experience [KFA+10a] with Prometheus on PL, we concluded that such peers are more appropriate for long-term experiments as they are typically more stable and well-provisioned. From this set of peers, we handpicked 100 PL peers placed in multiple countries around the globe for a good geographical coverage (Figure 3.1).

Our previous experience also showed that applications need to wait 10–20secs/hop for a high response completion when many peers are involved in a multi-hop inference re-

Table 3.1: Metrics used to select stable and resourceful PlanetLab peers for the Prometheus deployment and testing.

Resource	Availability
Total RAM	$\geq 2GB$
Free RAM	$\geq 90\%$
Free Disk Space	$\geq 20GB$
Cores per CPU	≥ 2
% of free CPU	$\geq 90\%$
Live Slices (allocated projects)	≤ 3
SSH & DNS Status	responsive in last 2h

Figure 3.1: Geographical distribution of PlanetLab sites used in the Prometheus' experiments.

quest. Therefore, for the end-to-end performance experiments, we set the application timeout to 15 seconds for every social hop in the graph traversed by Prometheus to fulfill an inference request at a high completion rate. This time is needed to tackle peer-to-peer communication delays due to long round-trip-times (RTT) (200–300ms on average), busy peer network interfaces (the PL peers were used by other researchers at the same time), peer churn (for maintenance or network disconnections) and retransmissions of requests to alternative peers. Prometheus uses this parameter by aggregating intermediate results until the per hop timeout T is reached, and then sends these results back to the

requesting peer. For the completion rate experiments we varied this timeout T to assess the trade-off between end-to-end response time and completion rate of a request.

3.3.2 Decentralizing the Social Graph

We used a bidirectional graph of $1,000$ users with initial edge weight of 0.1. The graph was created using a synthetic social graph generator described in [SCW+10] (a pseudocode of the algorithm used is presented in Appendix B), which consistently produces graphs with properties such as degree distribution and clustering coefficient similar to real social graphs. The properties of this graph are shown in Table 3.2.

As reported in [MMG+07] and [WBS+09], social networks such as Facebook, Youtube, orkut and LiveJournal exhibit power-law degree distributions with exponents in the range of 1.5 and 1.75. Furthermore, the average clustering coefficient of such networks is in the range of 0.16 and 0.33. The synthetic graph exhibits an average clustering coefficient and a power-law exponent close to the values found for these real social graphs.

Typical social graphs exhibit *small-world* properties [WS98], thus have small average shortest path legths, similar to random graphs of equal size, but have a significantly larger average clustering coefficient. If we compare the average shortest path length and average clustering coefficient of the synthetic graph with a random graph of equal size, we observe that the synthetic graph has a much larger clustering coefficient than the random graph (as expected [WS98]), but a slightly smaller average path length than the random graph. This can be attributed to the graph generator's focus on preferential attachment which increases connections between highly connected nodes [SCW+10], which leads to the overall reduction of the path lengths.

The graph size allowed for a realistic distribution of the $1,000$ users' data on the 100 PL peers. We distributed the social graph on peers in two ways. In the first way, users'

Table 3.2: Graph properties of the synthetic social graph used for the high-stress experiments.

Nodes	1,000
Edges	5,846
Radius	5
Diameter	8
Average Degree	11.692
Average Eccentricity	6.061
Average Clustering Coefficient	0.328
Average Shortest Path Length	3.545
Power-law Distribution Exponent	1.39
Average Clustering Coefficient of a random graph	0.0117
Average Shortest Path Length of a random graph	3.912

social data are stored on randomly selected peers. Consequently, groups of random users are mapped on the same peer. We will refer to this as the *random mapping* of users on peers. In the second way, we assume that groups of socially-connected users share the resources provided by a peer possibly contributed by a member of the group. Therefore, social groups of users are mapped on peers, reflecting a more realistic scenario of social data decentralization. We will refer to this as the *social mapping* of users on peers.

In our experiments, we created such a social mapping by using a modified version of the community detection algorithm introduced in [GN02] that allowed us to control the number of communities and their average size. The algorithm takes as input a social graph, the number of communities to be identified (which in our case is the number of peers in the system) and the minimum acceptable community size. The algorithm iteratively removes the social edge with the highest edge betweenness centrality [GN02] if by removing it a new community of the desired size is created. Removal of edges continues until the specified number of communities is met. Users from a given community are then mapped on the same peer.

However, even in the social mapping, neighboring communities in the social graph were mapped on random peers worldwide. This setup implies that geographically-close commu-

nities could store their information on random peers across the globe, which allowed us to examine a worst case performance of the platform with respect to network delays. Moreover, we assumed users defined ACPs that allowed access to all their data from all users. Therefore, any user could submit requests to access data of any other user and proceed over multiple hops, consequently stressing the system even further with the maximum graph traversal load.

The placement (mapping) of user data on peers introduces bias on the delays of requests between peers. Furthermore, although we did not apply any peer churn, and despite our efforts to pick stable nodes, random PL peers exhibited an average churn of up to 5% of the peers used during the experimental runs, which lasted 8 hours each. Therefore, splitting the application workload among initial peers (i.e., each peer submits requests only for a particular set of users) would have amplified this bias on the request performance. To minimize this bias, each peer probabilistically submitted the same application workload on behalf of the social graph's users and we report averaged results across all peers.

The average number of users mapped per peer, N, was set to 10, 30 and 50. The number of PL peers was kept constant to 100, forcing the user groups to overlap with each other for N>10 users/peer. This resembles the realistic scenario of overlapping social circles of users with some users participating in more than one circle (peer), and thus having multiple trusted peers. In effect, user's data were replicated on K=N/10 peers on average, hence K=1, 3 and 5 trusted peers/user.

3.3.3 Emulated Workloads

We emulated the workload of one social sensor and two social applications based on previous system characterizations [WBS+09, KGA08, GCX+05]. The emulated social sen-

sor tested the ability of the platform to manage and incorporate new social input with the existing data of users while under high-stress load. The emulated social applications tested the end-to-end performance of the *neighborhood* and *social_strength* requests.

3.3.3.1 Social Sensor Input

We emulated a Facebook social input based on a Facebook trace analysis [WBS+09]. The workload was characterized by the probability distribution function (PDF) for users to post comments on their friends' Facebook 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 (CDF) from Figure 8 in [WBS+09], as shown in Table 3.3.

Table 3.3: Probability distribution function for social inputs, based on a Facebook study. [WBS+09].

Social Degree Rank (Top%)	% Total Interaction (CDF)	Group assigned to user	Probability to choose user (PDF)
5	40.0	1	0.400
10	60.0	2	0.200
20	80.0	3	0.200
30	90.0	4	0.100
40	95.0	5	0.050
50	97.5	6	0.025
>50	100.0	7	0.025

To emulate a social interaction from user *ego* to user *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. User *alter* was randomly selected from *ego*'s direct social connections, to maintain the small-world properties of the graph. The weight of each input was kept constant to 0.01 for all users. Due to the fact that users were picked

based on their social degree, users with higher social degree probabilistically produced more input, leading to higher weights on their corresponding edges in the social graph.

3.3.3.2 Neighborhood Inference Requests

In order to create a workload for the neighborhood inference requests, we used an analysis of Twitter traces [KGA08]. Twitter [Twi12a] is a micro-blogging website that allows its users to share information at real-time. The sharing happens via small bursts of information called *tweets*, which can be at most 140 characters long. Each user can *follow* other users' stream of tweets that she finds interesting. Thus, she can be characterized by the number of followers she has over the number of users she follows, or the ratio R of *Followers/Following* and the number of tweets she produces.

According to the study [KGA08], we can classify the users into three main categories: 1) the *broadcasters* are heavily followed but they follow very few other users and publish a lot of tweets (e.g., news station, artists, etc), 2) the *acquaintances* are typical OSN users who tend to reciprocate the *follow* action of others (similar to "friendship" in Facebook) and they broadcast a moderate number of tweets, 3) the *spammers* are heavily spamming other users with following requests and do not tweet that often.

Depending on the content of a tweet, users may choose to repost it on their account status and this action can continue for multiple hops in the social graph, leading to a rapid spread of a tweet. Therefore, we can intuitively associate a tweet such as "Go Bulls! Let's meet for a drink after the football game", with a Prometheus neighborhood request (centered at the leader of the tweet) which traverses the social graph for a number of social hops h, over a label *football* and some minimum weight w.

We used the Twitter analysis to extract the probability distribution function of submitted neighborhood requests. We assumed that the users in our social graph were of the second

category (i.e. acquaintances) and thus had a ratio R close to one (which could also be reflected by the ratio in/out degree). From [KGA08] we observed that users with $R \simeq 1$ but with increased absolute number of followers and following (i.e. higher social degree) tend to tweet more. Therefore, using Figure 4 in [KGA08], each user was assigned to a group with a particular twitter count based on their degree.

Each user group was mapped onto a particular probability to be selected and submit a neighborhood request, as shown in Table 3.4. We assumed that users from the last group produced about half the tweets of users from the previous group, since numbers were not provided in the study. Once the group was selected, a user (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 and the weight was randomly picked from the set of values 0.01, 0.02, . . . , 0.10. Applying a weight threshold $\leq$0.1 on the inference requests produced a high-stress load since all edges had an initial weight of 0.1.

Table 3.4: Probability distribution function for the neighborhood requests, based on a Twitter study. [KGA08].

Degree	Number of tweets	Group assigned to user	Probability to choose user (PDF)
100-1000	1727	1	0.544
10-100	964	2	0.304
0-10	482	3	0.152

3.3.3.3 Social_Strength Inference Request

In order to create a workload for the social strength inference requests, we used an analysis of BitTorrent traces [GCX+05]. BitTorrent [Coh03] is currently the most popular P2P file-sharing protocol and responsible for about a third of the internet's traffic around the world [SM09]. It allows users to distribute files to other users in a decentralized way, therefore eliminating the need of a centralized server with bandwidth and computation

limitations. Via a cooperation mechanism in the protocol, each file is split into chunks, and peers (i.e., users) participating in the sharing of the file are organized into a swarm. Peers or *leechers* in a swarm may download chunks of the file from the other swarm peers (*seeds*), but also upload chunks they currently have, thus contributing back to the swarm.

A battery-aware BitTorrent application [KBI09] could run on mobile devices and users could rely on social incentives to be allowed to temporarily "free ride" the system (i.e., do not upload any chunks to other users) when low on battery. Members of the same swarm (i.e., they have interest in the same file) could check their social strength with the needy leecher to see if they want to contribute by uploading on her behalf. This translates into a series of social strength requests submitted to Prometheus by random users participating in BitTorrent swarms.

Therefore, for the social strength workload we assumed that users participated at random in BitTorrent swarms. Two users were randomly selected as the source and destination of the social strength inference request. The source user was associated with a total number of requests she would submit throughout the experiment. This number was extracted by the torrent request distribution from the analysis of BitTorrent traces (Figure 9b in [GCX+05]), as shown in Table 3.5.

Table 3.5: Probability distribution function for the social strength requests, based on a BitTorrent study. [GCX+05].

Number of torrent requests (i.e., social strength requests) to submit during experiment	Probability to choose user (PDF)
1	0.450
2	0.144
4	0.178
8	0.104
15	0.079
25	0.026
35	0.015
45	0.004

3.4 Performance Evaluation with Emulated Workloads

In the first two experiments, we deployed Prometheus on PlanetLab to test the performance of the system's functionalities in a real distributed environment. During these experiments, we used high-stress workloads from the two emulated social applications and the social input as described earlier, while varying inference request parameters such as users involved in the request and number of social hops to be traversed. The first set of experiments was performed under a constant, but relatively long, timeout per hop $T = 15$ seconds to allow high completion rate of inference requests. The second set of experiments was performed with a variable timeout per hop, as it would be selected by each application, to assess variability in the quality of results returned (completion rate) vs. end-to-end performance.

3.4.1 End-to-End Performance

The first set of experiments had two goals: (1) to measure Prometheus' performance over a widely distributed network such as PL, and (2) to assess the effect of socially-aware trusted peer selection on the system's overall performance. For every experimental run, more than 1 million *social_strength* and *neighborhood* requests and more than 100 thousand social inputs were submitted from the emulated applications and social aggregator. Figures 3.2– 3.4 show the cumulative distribution function of the end-to-end average response time for the *neighborhood* inference for different social hops and number of users per peer. We do not include the results for the *social_strength* inference as its performance is almost identical to the one for *neighborhood* for 2 hops. The reason is that this function has to verify all the possible paths between two users within at most 2 social hops of each other. We formulate the following lessons from this set of experiments.

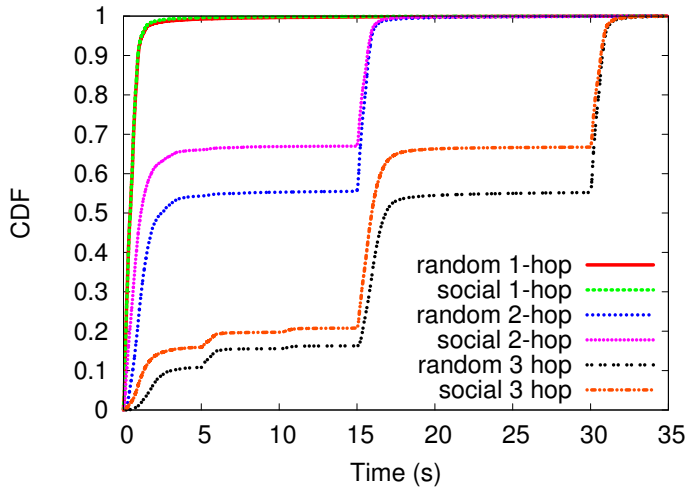

Figure 3.2: CDF of the average end-to-end response time of the *neighborhood* inference for the *random* and *social* mappings, different social hops for 10 users per peer, with $T = 15$ seconds. The distributions of the *social* mapping were found statistically different from the *random* mapping using the Kolmogorov-Smirnov test, with $p < 0.0001$ and test statistics $k = 0.0132$, $k = 0.2276$ and $k = 0.1641$ for 1, 2 and 3 social hop requests, respectively.

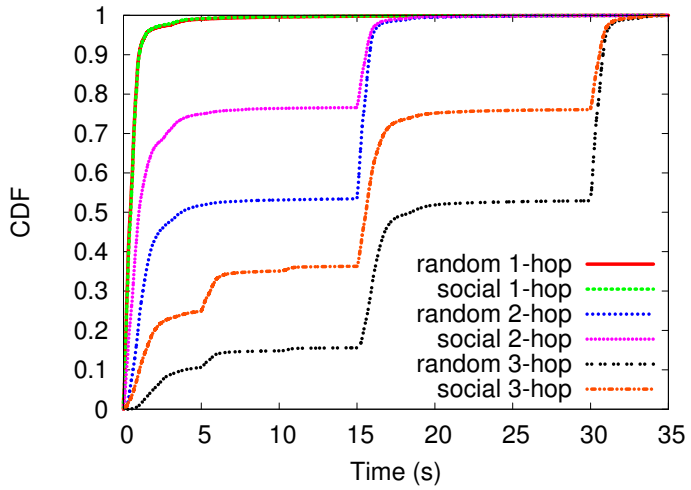

Figure 3.3: CDF of the average end-to-end response time of the *neighborhood* inference for the *random* and *social* mappings, different social hops for 30 users per peer, with $T = 15$ seconds. The distributions of the *social* mapping were found statistically different from the *random* mapping using the Kolmogorov-Smirnov test, with $p < 0.0001$ and test statistics $k = 0.0180$, $k = 0.2771$ and $k = 0.2918$ for 1, 2 and 3 social hop requests, respectively.

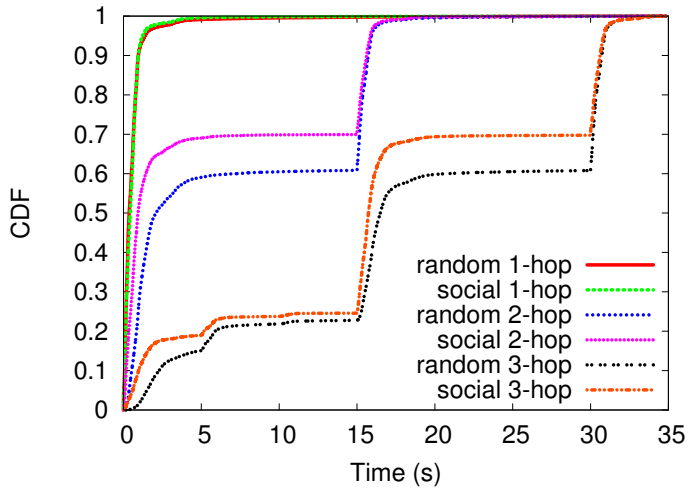

Figure 3.4: CDF of the average end-to-end response time of the *neighborhood* inference for the *random* and *social* mappings, different social hops for 50 users per peer, with $T = 15$ seconds. The distributions of the *social* mapping were found statistically different from the *random* mapping using the Kolmogorov-Smirnov test, with $p < 0.0001$ and test statistics $k = 0.0480$, $k = 0.2642$ and $k = 0.1710$ for 1, 2 and 3 social hop requests, respectively.

Lesson 1: *The social-based mapping of users onto peers leads to significant improvements in end-to-end response time and reduction in message overhead.* In all cases of users/peer, the social mapping leads to faster responses of requests than the random mapping. For the case of 30 users/peer, gains of up to 20–25% in response time over the random mapping are observed. The difference is more visible for 2 and 3 hops, as the 1 hop function is computed either locally at the submitting peer or the first available trusted peer of the source user. For the case of 50 users/peer, the system continues to perform better with the social than the random mapping, but the improvements are smaller than between 10 and 30 users/peer. This is because under this case, the average number of peers that must be contacted for information is reduced for both mappings. However, the reduction is more prominent in the random mapping, and thus we notice more improvement on the random than the social mapping.

In all cases, the social mapping outperforms the random mapping since it needs significantly less number of messages to fulfill the same requests. For 10 users per peer, the

system with social mapping has 38.4% reduction in message overhead over the random mapping. The average message overhead is increased in the case of 30 users per peer under the social mapping. This is because the increase in number of users per peer (N) from 10 to 30 also increases the number of peers per user (K) from 1 to 3, which adds alternative peers in the system for the users that their requests failed in the $N = 10$ case. However, the social mapping still outperforms the random mapping with 16.3% less message overhead. Increasing further the number of users per peer reduces the overall number of messages needed as the number of peers to be contacted is also reduced in both mappings. Still, the social mapping leads to 37.3% less message overhead than the random mapping.

Lesson 2: *Increasing the service availability through data replication and number of users per peer improves end-to-end response time and reduces message overhead.* Our experimental design has the following specifics: by increasing the average number of users who are mapped on a peer (N), we also increase the average number of trusted peers per user (K), and therefore, service availability for that user. In general, since inference requests for a user can be fulfilled by any of her trusted peers, we observe that increasing the availability of users' data by a factor of 3 to 5 times, and correspondingly increasing the number of users mapped per peer, improves the end-to-end performance by up to 25% (when K=3) and reduces the message overhead in the system by ∼30%, on average. This overall performance gain is due to the following two reasons. First, having more user data on each peer allows for a higher portion of each request to complete locally on the peer, and therefore fewer peers need to be contacted, i.e., fewer network peer hops. Second, given high peer churn and vulnerable P2P communication, more service availability per user means more alternative trusted peers to contact for an inference request to be fulfilled faster.

Lesson 3: *Creating the TPL can be an expensive operation.* A request for a user X can arrive at any random peer. This peer has to first create X's TPL, and then forward the

request to X's network-wise closest trusted peer (as explained in 2.3.3). This operation involves several time-consuming lookups in the DHT, which result in multiple peer traversals. The distribution of the overhead associated with the cold start of creating a user's TPL over the PL infrastructure had a 50^{th}, 90^{th}, and 99^{th} percentile of 1.05, 2.15 and 8.78 seconds, respectively (as shown in Figure 3.5). Thus, for the majority of the users this process can be fast, but for some unfortunate users it can take as much as 8–10 seconds. Similar delays (6–10 seconds) were reported in [SVCC09] for the group activities of the Vis-a-Vis system running over PL nodes. To mitigate this problem, Prometheus caches the TPL as explained in 2.3.3. The graphs in Figures 3.2– 3.4 show the performance using this caching mechanism, since we are interested in testing Prometheus' performance in inference execution and not in DHT lookups.

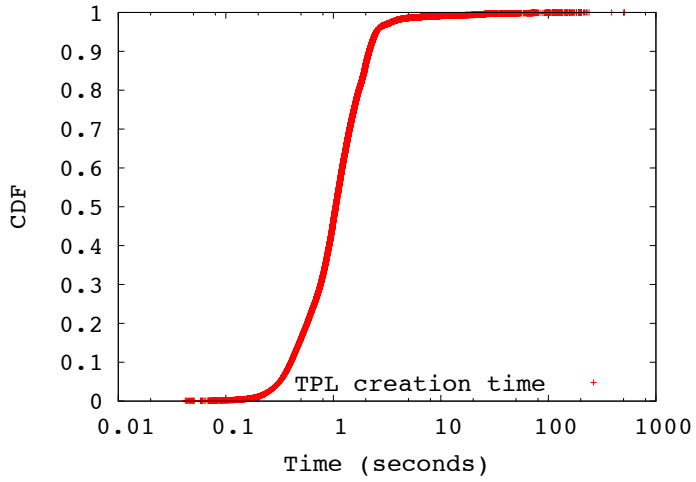

Figure 3.5: CDF of the time needed for a peer to create a trusted peer list of a user in PlanetLab.

Lesson 4: *The response time is relatively high due to the overloaded testbed, especially for* 2 *and* 3 *hops.* In our worldwide PL testbed, the average RTT was 200-300 ms. In addition, the testbed was generally loaded with other projects running along with ours, thus leading to about 1–2 seconds just to establish a reliable TCP connection to submit a request to a peer or receive an inference result from a peer. The response time greatly depends on the number of hops to be traveled by the request and the number of peers to

be contacted at each hop. Even though the aggregation of a request result is executed in parallel per social hop, multiple peers must be contacted per hop, with potentially long response delays to complete a request at the highest rate possible. For example, for 10 users/peer in the social (random) mapping, an average of 37 (48) peers have to be contacted to collect *neighborhood* results from users located 3 social hops away from the source user (the number of users returned is 350 on average). Such a request took about 30–35 seconds to reach ∼100% completion rate, given the 15-second timeout per hop.

Lesson 5: *Caching optimizations and geographic social graph decentralization allows better scalability.* By placing socially-close communities on random peers, we examined a rather pessimistic experimental scenario of longer than expected delays for request execution. However, in reality, we expect neighboring communities in the social graph being placed in geographically close peers (e.g., same country peers) instead of random, which could effectively reduce delays by one order of magnitude, since the average (median) RTT between PL peers of the same country was 25.5 (15.7) msec in our experiments for over 35 countries, as seen in Figure 3.6. To further reduce execution delays, 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 as the social graph rarely changes [Gol07] and will allow the inference execution to scale easier to thousands of peers.

3.4.2 Response Time vs. Completion Rate

In general, the longer an application is willing to wait, the more complete the information returned by the social inference is. In the previous experiments we assumed that a long timeout of T=15 seconds per hop offers a very high completion rate for requests fulfilled in the PL infrastructure. We designed a second set of experiments to measure the trade-off between end-to-end response time and response completion rate, when applications use

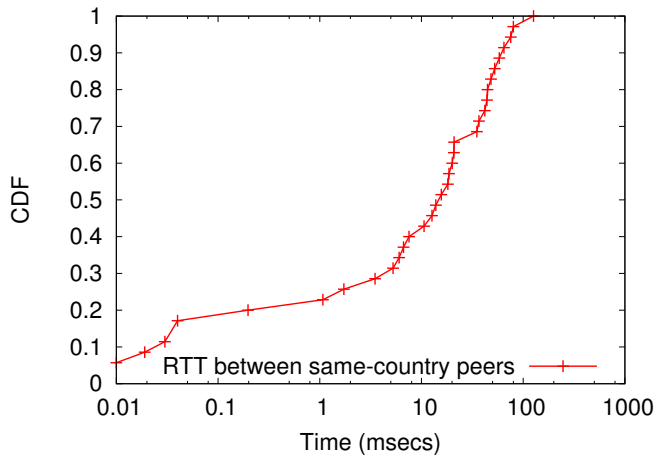

Figure 3.6: CDF of the round trip time (RTT) between same-country peers in PlanetLab.

variable timeout T=2.5, 5.0, 7.5, 10.0 seconds, using a social mapping of 30 users/peer for the 1000-user social graph.

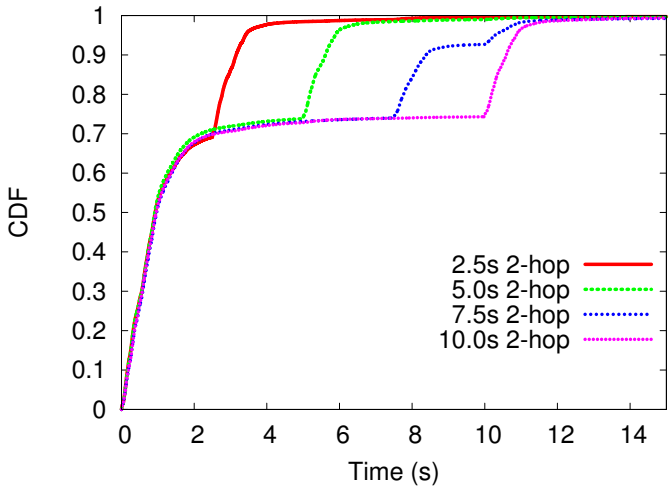

Figure 3.7: CDF of the average end-to-end response time of the 2-hop *neighborhood* inference for the *social* mapping, with 30 users per peer and varying timeout values.

Figure 3.7 and 3.8 show the CDF of the end-to-end response times for neighborhood requests of 2 and 3 social hops, respectively (a request of 1 social hop cannot timeout). For both request types, we can clearly see the effect of the timeout in the end-to-end response time performance. Furthermore, in the 3-hop requests, we notice a more promi-

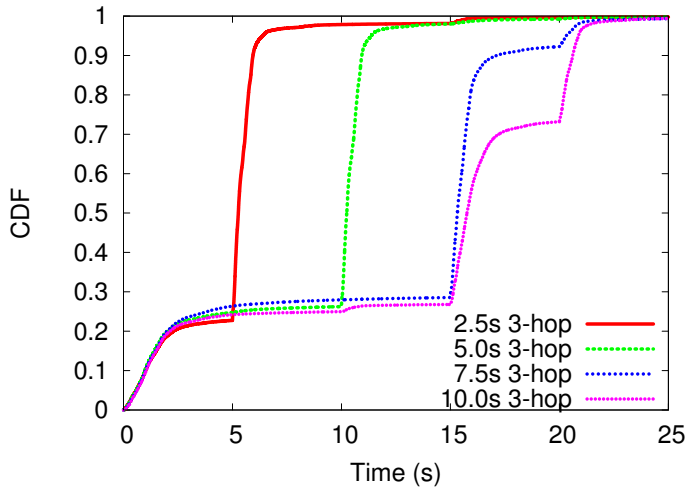

Figure 3.8: CDF of the average end-to-end response time of the 3-hop *neighborhood* inference for the *social* mapping, with 30 users per peer and varying timeout values.

nent use of the retransmission policy (also seen in Figures 3.2– 3.4 but in a lesser degree) and especially in the cases of 7.5 and 10 seconds. Using the theoretical results shown in Figure 2.10, we notice that case (c) with one timeout and 4 remote messages ($4TR$) is the most prominent for the 3-hop requests with one timeout and the case (d) with two timeouts and 2 remote messages ($2TR$) the next more prominent.

In a system such as PlanetLab, the time delays TA and TD are not truly negligible and the minimum time delay of one remote message (TR) cannot be zero, as assumed in the analysis of Section 2.3.8. Therefore, the requests exhibit an even longer delay to be fulfilled, and this delay presents a multi-step variability, depending on what portion of the requests was executed in 1, 2 or 3 network hops and at what network hop there were timeouts involved.

Figure 3.9 shows the cumulative distribution function of the average completion percentage of the 3-hop neighborhood requests under different application-set timeout values. The results for 1 and 2 hops showed practically 100% completion for all requests and timeout values and are omitted for brevity. Overall, we observe a clear trade-off between request completion rate and application waiting time for response. For example, when

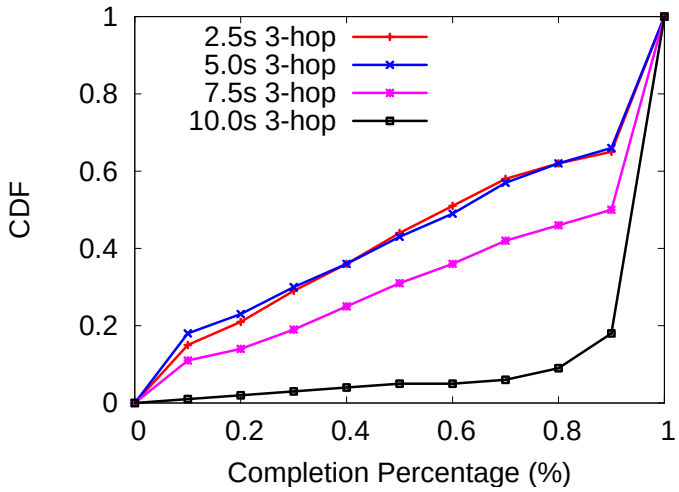

Figure 3.9: CDF of the average completion percentage of 3–hop neighborhood requests for varying timeout values.

T=7.5(10)seconds, about 50%(80%) of the requests have $> 90\%$ completion. A real-time social application—e.g., "using the *proximity* inference, invite my 2-hop football contacts for celebration of the team's victory"—could set a low timeout for quick, yet incomplete, results.

3.5 Performance Evaluation with Real Mobile Application

We validate the usability of Prometheus as a social data management service by developing a mobile social application, CallCensor, that utilizes the Prometheus inference functions through the exposed API, under real-time constraints. Past work (ContextPhone from University of Helsinki [ROPT05]) introduced an application called *ContextContact* which offers cues to the caller about the callee's social context such as location, collocation with other people, phone ringer status, etc. Our application builds on these lessons and allows the callee's phone to adjust the phone ring based on the owner's social context and the social relationship with the caller. We measured its end-to-end performance using a real multi-graph of 100 users.

The CallCensor application leverages social information received from Prometheus to decide whether or not to allow incoming calls to go through. In addition to the social information from Prometheus, this application also uses the phone location and collocation via *BT* with other phones, to infer whether the user is at work and in a meeting (e.g., in the boss's 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., don't allow personal calls while at work), the application decides if the phone should ring, vibrate or silence upon receiving the call. The application was written in Java for devices running Google Android OS and was tested on a Nexus One mobile phone from HTC (1GHz CPU, 512MB RAM).

There are multiple scenarios a caller can be connected to the phone's owner; we tested three: directly connected within 1 social hop, indirectly connect by 2 social hops, and connected with a high social strength. We tested each of these scenarios 50 times. For each of them, the *ego* and *alter* were randomly chosen, and the inference request was sent to a random peer.

We assumed users defined ACPs allowing access to all their data, thus enabling requests to proceed over multiple hops, and consequently stressing the system at maximum possible load. 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, and the processing time by the mobile application.

3.5.1 Social Multi-Graph from Real Traces

The social graph used in the CallCensor application experiments was based on data collected at NJIT [PBB11]. 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 provide their friend lists and participate in a survey.

The user set was small (100 users) compared to the size of the student body ($9,000$), 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 a subgraph of the 45 minute collocations. A summary of the graph properties (when flattened) is shown in Table 3.6.

Table 3.6: Graph properties of the multi-graph produced using real collocation and Facebook data from NJIT. [PBB11].

Nodes	100
Edges	469
Average Degree	9.38
Average Eccentricity	4.47
Average Clustering Coefficient	0.37
Average Shortest Path Length	2.50
Power-law Distribution Exponent	1.37

While the graph edges 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 mobile application 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 3.10 illustrates this graph and demonstrates one of Prometheus' features: using multi-edge graphs provides for better connectivity between users. Neither the "Facebook" nor the "collocation" graph is connected, but the graph containing both types of edges is. We equally distributed the $NJIT$ graph with a social mapping on three PL nodes in the US. Splitting this 100 user-graph on 3 PL peers implies a similar ratio of users/peer as before (about 30 users/peer), while testing 1 and 2 hop inference execution on a real multi-graph.

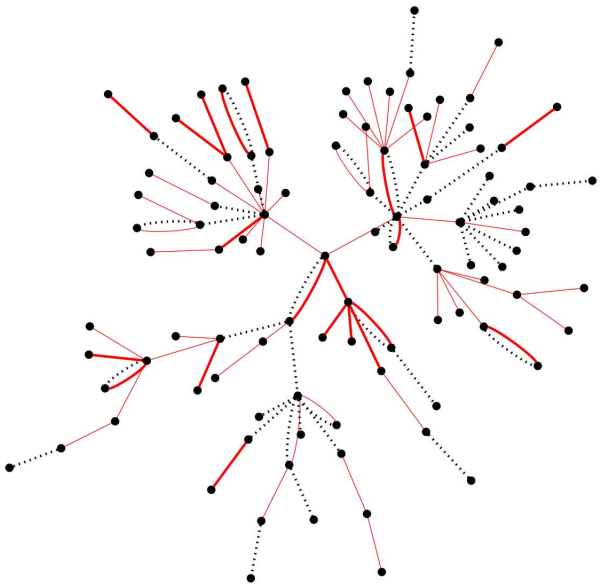

Figure 3.10: Real multi-graph with Facebook edges (black dashed lines) and collocation edges (red continuous lines). Line thickness demonstrates edge weight (for collocation edges).

3.5.2 Application Response Performance

Figure 3.11 presents the performance for the requests sent by CallCensor, for each of the three scenarios examined. The results show the time spent by the requests only in Prometheus and the overall time needed by the CallCensor to request and handle a response. We first observe that the results meet the time constraints of the application: the

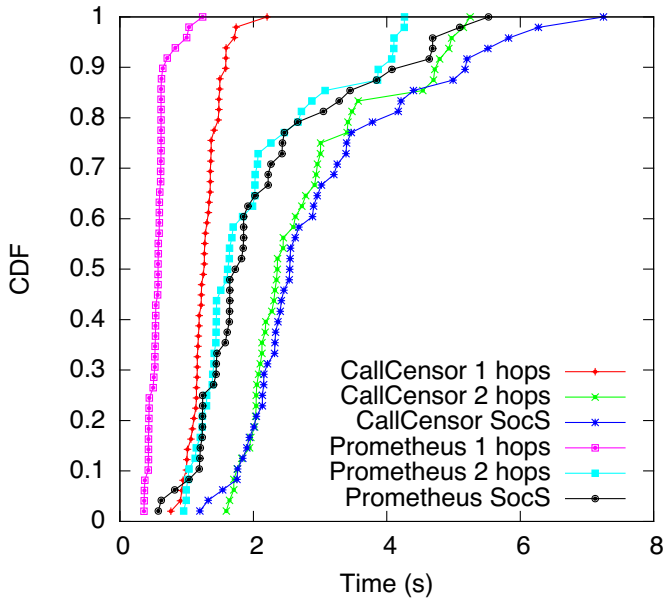

Figure 3.11: CDF of average end-to-end response time for CallCensor, under three social inference function requests: 1 and 2 hops *neighborhood* requests and *social_strength (SocS) requests*.

response must arrive and the ring must be adjusted before the call is forwarded to the voicemail of the callee (we used the default voicemail time setting).

Second, we notice that the application itself introduced a significant overhead: for example, as much as 100% in the 1–hop *neighborhood* and 50% in the 2–hop *neighborhood* and *social_strength*, due to both communication overhead and execution time on the mobile phone. Third, we confirm the similarity of the *social_strength* results with the *neighborhood* for 2 social hops, as found in the first set of experiments. Fourth, the 2-hop request results using this small social graph are similar in performance with the larger 1000-user graph used before (social mapping, 30 users/peer). In particular, more than 70% of requests finished within 2.5 seconds in both setups, which leads us to believe that similar performance of the mobile application could be expected in a larger social graph distributed over hundreds of peers.

Chapter 4: Resilience to Malicious Attacks in Prometheus

A social data management service can be the target of attacks at different levels of the system (Figure 4.1): 1) at the infrastructure level, 2) at the social graph level, 3) at the service level, and 4) at the application level. Depending on the architecture of the system (centralized or decentralized), defenses can be more effective and easier to implement. In this Chapter we discuss how the design characteristics of Prometheus enable the system to resist such attacks, or mitigate their effects on users.

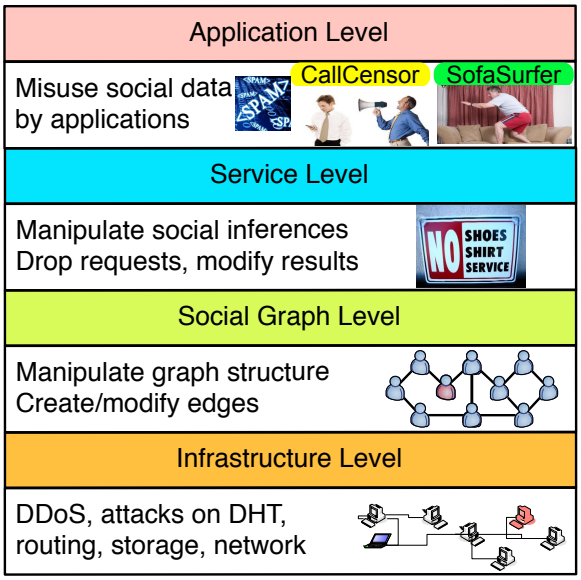

Figure 4.1: Attacks can target a social data management service at different system levels.

First, attacks at the infrastructure level such as denial of service attacks can target both the servers of a centralized system and the peers participating in a decentralized system.

A decentralized system such as a P2P network, however, can be the target of additional attacks on the infrastructure, with a goal to disrupt the communications (routing of messages between peers) or storage of content in the system. The main classes of attacks on DHT-based P2P systems are: 1) the Sybil attack, 2) the Eclipse attack, and 3) attacks on the routing and storage functionalities of the system. Solutions to such attacks are reviewed in [UPvS11]. Attacks at the infrastructure level are out of the scope of this work.

Second, attacks on the social graph aim to manipulate the graph structure by creating or modifying social edges between users. Centralized systems have complete knowledge of the social graph and monitor its changes due to user activity. Thus, it can be easier to detect when malicious users, independently or in collaboration, create artificial edges with honest users to gain access to their social data (e.g., Facebook Immune System [SCM11]). However, not all such defenses are successful [BMBR11] and more research is needed to address such attacks. Decentralized systems can also employ techniques (e.g., [YKGF06, YGKX08]) to defend against such attacks, but they are typically more expensive and less effective due to the distributed nature of the social graph. In Section 4.1 we discuss how the characteristics of the social graph maintained by Prometheus, i.e., a directional, labeled and weighted social graph, protect users against many variants of these attacks.

Third, attacks at the service level attempt to manipulate inferences on the social graph (drop requests, modify results, etc). A centralized system such as Facebook can claim trust in the social inferences on the social graph, since it controls the access and use of its servers. Such attacks are more effective on decentralized systems due to the different level of guarantees that distributed, user-contributed peers can provide. In Section 4.2 we discuss how the socially-aware design of Prometheus increases the system resilience to these attacks.

Finally, we consider attacks at the application level. Centralized systems expose the social data of users to 3rd party applications and services through APIs. However, users

typically have limited or no control on the exposure and use of their data from such 3rd party services, as seen by the numerous attacks on users with email spam and phishing campaigns [Sym11, BCF09], as well as collection of private information from online social aggregators such as Spokeo [Spo12]. Prometheus enables applications to mine the rich social graph stored in the system via social inference functions, but allows users to control access to their social data via fine-grained access control policies. In Section 4.3 we discuss how Prometheus can defend against attacks at the application level.

4.1 Attacks at the Social Graph Level

It has been shown that attackers are able to create social edges with honest users in social networks (e.g., Koobface [BCF09, Dan09, Vam08]), or bias them to reciprocate social edges [CM08]. Moreover, these attacks can be automated using socialbots [BMBR11] and current defense systems are not capable of detecting and stopping the majority of such attacks (e.g., Facebook [SCM11]).

The reciprocation of a social edge (e.g., Facebook, Twitter, etc) usually means partial or complete access to honest users' social data. These data could include online profiles and personal information and can be used for email spam and phishing campaigns [Sym11, BCF09], or sold to the black market for identity thefts. A more subtle attack could include efforts to manipulate public opinion on sensitive issues, for example with respect to public health [Mor09] or political elections [RCM+11, Mis11].

Because Prometheus maintains a directed social graph, manipulating it effectively is more challenging. While it is relatively easy to create an edge from the attacker to the victim, creating the reciprocal edge from the victim to the attacker is not in the attacker's control. Next, we examine these two scenarios in more detail.

4.1.1 Creating Social Edge from Attacker to Victim

Attackers could bias their social sensors to create directed edges towards victims to manipulate the social graph and access their social data. For example, *Alice* could repeatedly email a large number of users to create a social edge with them. These edges could be of different labels and weights, in an effort to cover a wide range of social activities that honest users interact with each other.

However, no matter how many edges she creates with various labels and weights, the *directionality* of these edges will always be from her to victims, thus restricting their effectiveness when it comes to affecting or even accessing honest users' social data. Therefore, even though she can include in her 1-hop neighborhood the entire social graph, she will not be part of any honest user's 1-hop (or n-hop) social neighborhood.

4.1.2 Reciprocating Social Edge from Victim to Attacker

An attacker *Alice* can try via social engineering [Gal11] or other means (for example, socialbots [BMBR11]) to convince an honest user *Bob* to reciprocate an edge to her. *Bob* can defend against such attackers using Prometheus inferences or SybilLimit-based techniques [YGKX08].

However, if this attack is successful, *Alice* (and her malicious 1-hop neighborhood) may be able to access part of his social data, depending on *Bob*'s access control policies. This could also lead to a more subtle attack on complex social inferences such as the *social strength*. In the next paragraphs we examine in more detail the two defense mechanisms that *Bob* can use to reduce the reciprocation of social edges to attackers, as well as the attack on the complex social inference requests.

4.1.2.1 Defense via Prometheus Inferences

Bob could consult his direct social neighborhood about *Alice* before establishing an edge with her. Individuals typically share a significant number of social contacts (i.e., people have common friends with each other) and this overlap of friends is an indication of the strength of their social relationship [OSH+07]. Therefore, *Bob* could query Prometheus for his 2-hop social neighborhood (i.e., the direct contacts of his contacts), to establish if *Alice* appears in his friends' 1-hop neighborhood, and if she does, under what edge label and corresponding weight.

Note that a similar mechanism has been used by Facebook [Fac12b] to allow users to recover their account password through 3 of their friends. However, the particular Facebook mechanism has also been shown to be vulnerable to Sybil attacks via social engineering [Gal11]. Instead, our proposed mechanism takes into account edge labels and weights, and allows for a more collective decision to take place between friends regarding a stranger, emphasizing Prometheus' socially-aware design.

4.1.2.2 Defense via SybilLimit-based Techniques

Bob could apply a SybilLimit-based technique [YGKX08] to traverse his social graph and investigate if this social edge is legitimate or not. This technique assumes that attackers such as *Alice*, with potentially multiple or *Sybil* identities in the system, are concentrated within a Sybil region and attack the honest users H via a specific set of social edges S_G, or *attack edges*. Such a defense mechanism provides guarantees that each honest user could reciprocate at most $O(logH)$ social edges to attackers in the social graph, per attack edge in the set S_G, even when $|S_G| = o(H/logH)$. Given the technique's decision, *Bob* can accept an edge or not. As an extension to this technique, if the social edge is not

accepted, it can be placed under "quarantine", until *Bob* has enough interaction history with *Alice* to reconsider his decision on the edge formation.

4.1.2.3 Manipulating Complex Inference Requests

Alice could partially influence complex requests such as *social strength*, that traverse the social graph over different types of edges and weights to reach users that are directly or indirectly connected.

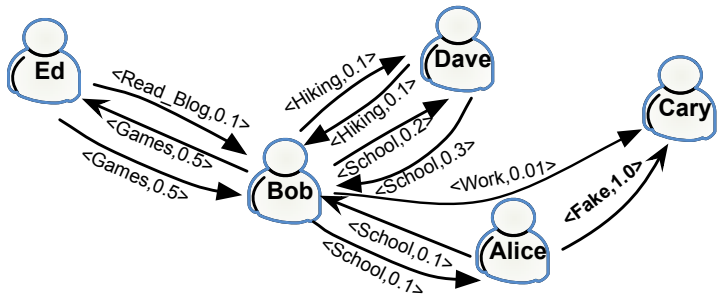

Figure 4.2: User attack on the social graph: *Alice* creates a fake edge to *Cary* with high weight to bias inferences from *Bob* to *Cary*.

Assume that *Alice* has a reciprocal edge with *Bob* and creates a fake edge with high weight to another attacker *Cary* who also managed to create a reciprocal edge from *Bob* to her, as in Figure 4.2. Given that *Bob* submitted a *social strength* request between him and *Cary*, the request will search the social graph to identify the path (direct or indirect) with the highest strength (as explained in Section 2.3). The fake edge between *Alice* and *Cary* can cause the reported social strength between *Bob* and *Cary* to be 0.1 whereas without the fake edge it would be 0.01. Thus, *Alice* can mislead such a request originating from *Bob* to utilize the indirect social path of *Bob* to *Cary through her*, instead of the direct path between the two users.

In general, an attacker *Alice* could cause any social strength request from a user directly connected to her to be artificially increased. In theory, this implies that *Alice* could enforce a minimum social strength to any other user within her 1–hop neighborhood, given that she is in their 1–hop neighborhood as well. In practice however, even though *Alice* can easily create outbound edges, she has little control over the creation of inbound edges, their label and weight, as well as if she will be granted access from *Bob* through his ACPs. The above example demonstrates the need for further investigation of attacks when considering the implementation of complex inferences such as the *social_strength*.

4.2 Attacks at the Service Level

Prometheus allows users to store their social data on particular peers based on out-of-band trust with the peer owners. Therefore, users can assume that these peers will not act maliciously while hosting their data and during the execution of an inference request. If such a peer acts maliciously, the peer and its owner can be flagged accordingly (blacklisted) in the ACPs of all honest users so that future requests initiated from the particular peer and user are dropped and requests are not sent to the specific peer for service.

As explained earlier in the inference function execution (Section 2.3.7), if a peer does not have the social data necessary to fulfill a request locally, a different peer must provide them. In fact, in a decentralized system such as Prometheus, many peers can participate in the execution of a single inference request. This depends on how the social graph is decentralized in the system and the type of request and the n-hop distance it will travel on the social graph, which can translate to multiple network hops in the P2P system.

Prometheus' design enables users to decentralize their social graph in a socially-aware manner, since socially connected users can store their data on the same peer, therefore reducing the number of peers needed to be queried during inference execution. However,

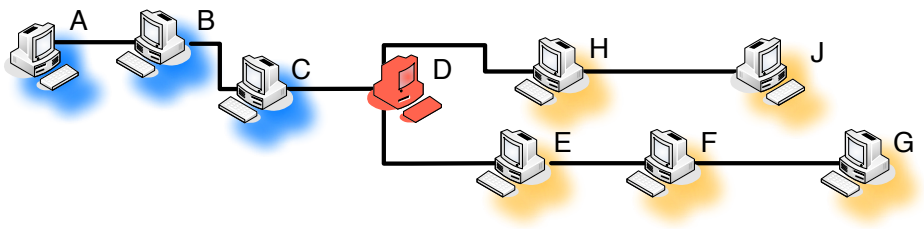

Figure 4.3: Example of peer influence. The red peer (D) can affect the requests submitted from the blue peers (A, B or C) that pass through peer D to reach peers E–J or results returned from these peers.

even with this design, multiple peers may need to be queried and in a system that protects user privacy, a requestor cannot distinguish how and from which peer any given item entered the result set.

Therefore, the intermediate peers serving a multi-hop inference request cannot be identified from the requester. Consequently, for *Alice* to mount a successful attack at the service level, she would have to control such intermediate peers in the system. These peers could have the opportunity to drop incoming requests, modify intermediate results sent to other peers, or change the parameters of secondary requests sent to new peers (Figure 4.3).

For example, if a malicious peer serves the ith hop of a n-hop request, it can "override" the results of the ith+1 to the nth hop of its leg of the request and remain undetected. Moreover, if malicious peers collude with each other (either because they are owned or have been compromised by the same attacker) they can increase the magnitude of the attack at the service level.

In this section we experimentally investigate how the socially-aware design of Prometheus increases the system resilience to such attacks at the service level, under different experimental setups.

4.2.1 Peer Influence

We studied system resilience to attacks on inference request execution by measuring peers' opportunity to influence results when serving a *neighborhood* inference request. We define a peer's *influence* on requests as the fraction of requests that the peer serviced over the total number of requests issued in the system, during a period of time t:

$$\text{peer influence} = \frac{\text{number of requests the peer serviced during period } t}{\text{total number of requests issued during period } t}$$

This fraction represents the overall opportunity of a peer to serve, and thus potentially manipulate the results of any given request issued.

A peer's influence on a request increases with the number of social hops the request traverses the graph, since, probabilistically, there are more chances the peer will participate in the request's graph traversal. We do not consider the first hop (i.e., the source peer) of a request as malicious, since if it is, no results can be considered legitimate. We consider the worst case scenario in which we do not restrict the weight of the social relationships used, all edges are reciprocal and users define ACPs allowing access to all their data, thus enabling requests to proceed over multiple peer hops.

Next, we extend our preliminary work [BKI11] and experimentally evaluate the influence malicious peers have over social inference execution on a distributed social graph. We performed three sets of experiments and extracted lessons regarding the system's resilience with respect to inference execution. During the experiments, an n-hop neighborhood request was performed for each *ego* (user) in the social graph. This request was submitted to the peer the user was mapped to, i.e. peer P_0, which could fulfill requests regarding information *only* about users mapped to P_0. For users in subsequent social hops from *ego* that P_0 did not have social data, P_0 randomly chose one of the peers stor-

ing the particular users' data and submitted secondary requests to be fulfilled by those peers. Each time a peer served a secondary request, we increased the peer's *influence*.

4.2.2 Peer Influence on a Synthetic Graph

The first set of experiments was performed on the 1000-user synthetic graph (same graph used in Section 3.4) distributed on 100 peers. In this set of experiments we studied the peer influence by identifying and mapping communities onto peers using the algorithm from Section 3.3.2. This algorithm allowed us to control the size and number of communities as well as the average replication factor of users' social data (K). Therefore, multiple copies of a user's data were allowed and thus multiple trusted peers for each user.

To eliminate any bias introduced by the random peer selection, we performed 100 iterations of each experimental configuration and report the average influence. Since we reach nearly 100% of the users in the 1000-user social graph within 4 hops, we only measured influence for $2, 3$, and 4 hop requests. To produce random mappings while preserving the community size distribution, we used the community sizes produced by the social mappings and shuffled the users randomly between communities.

Figure 4.4 plots the cumulative distribution function of the average influence of peers in neighborhood requests for the synthetic graph, for different combinations of users per peer and number of hops per request and for social and random mappings of users onto peers. From these results we formulate the following lesson.

Lesson 1: *The socially-aware mapping of users onto peers reduces the average opportunity of peers to influence requests.* From the results in Figure 4.4 we observe that the random mapping leads to an overall steady influence rate on peers, depending on the number of hops. On the other hand, the social mapping leads more than 85% of peers to have less

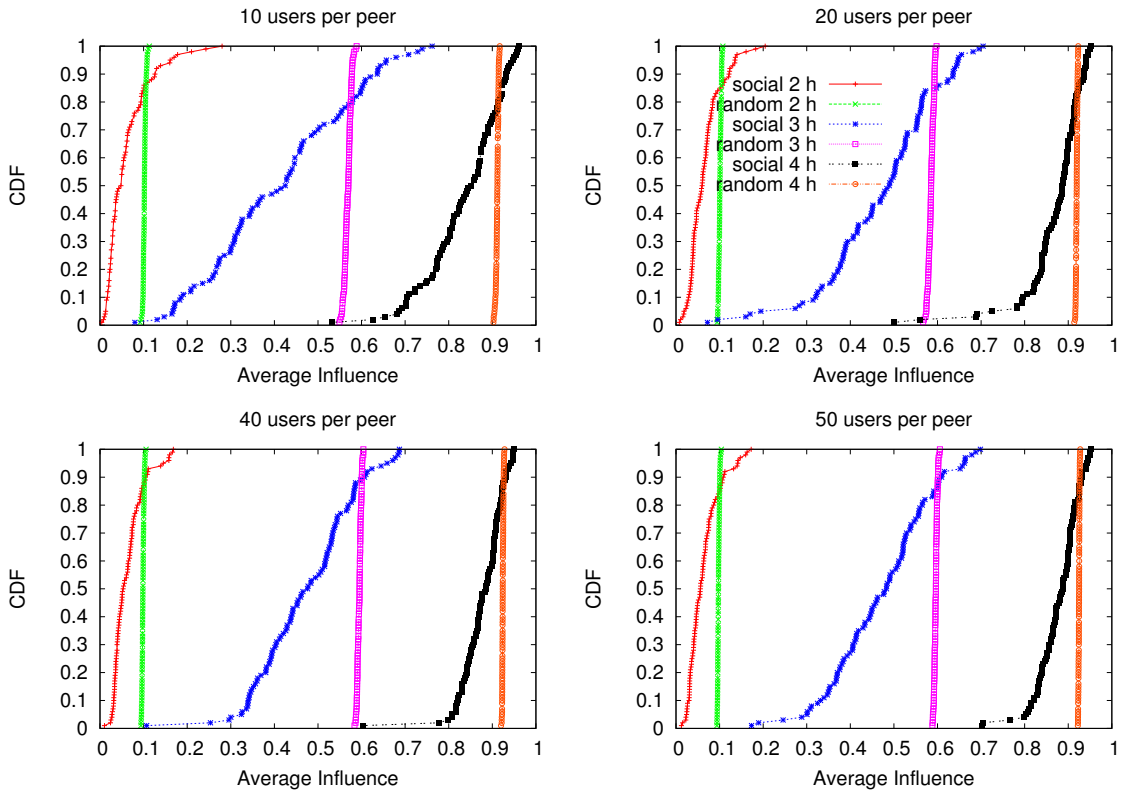

Figure 4.4: CDF of average peer influence for random and social mappings of the synthetic graph, for combinations of users per peer and number of hops per request.

average influence than peers in the random mapping, even though this influence increases and is also depended on the number of hops.

We can explain this result if we consider that in a social mapping, socially-close users are mapped to the same peers, increasing the likelihood that more hops are served locally when compared to randomly mapped users on peers. This also implies that a social mapping results in fewer secondary requests sent in the system (regardless of the way communities were identified and mapped on peers), and therefore supports our performance results (Chapter 3) which showed inferences on a socially-mapped graph to execute faster than on a randomly-mapped graph.

4.2.3 Peer Influence on Real Graphs

In our second set of experiments we studied the peer influence on five larger graphs based on real traces from diverse application domains, such as file sharing using peer-to-peer protocols (Gnutella, [RIF02]), email communications between company employees (Enron, [Les12]), trust on consumer reviews (Epinions, [Les12]) and friendships in a news website (Slashdot, [Les12]). We considered all networks undirected and unweighted and used only the largest connected component from each graph to ensure reachability between all pairs of users.

The time complexity of the community detection algorithm used in Section 3.3.2 is very high and unsuitable for networks larger than a few thousand nodes. Thus, to produce social mappings of users to peers for the real graphs, we identified social communities with the modified algorithm used in [KI11] (recursive-based Louvain method, Section 7.1.2), for average community sizes of N=10, 50 and 100, while maintaining the replication factor to one (K=1). Table 4.1 presents a summary of these networks and the communities found. We describe in more detail these networks and the community detection algorithm in Section 7.1.

Table 4.1: Summary information of the real networks used in the experimental study for the influence in Prometheus requests.

Network	Users	Edges	Communities (avg size N)			Source
			N=10	N=50	N=100	
Gnutella04	10,876	39,994	1,088	218	109	[RIF02]
Email-Enron	33,696	180,811	6,256	1,246	619	[Les12]
Gnutella31	62,561	147,878	3,370	674	337	[RIF02]
Epinions	75,877	405,739	7,564	1,485	727	[Les12]
Slashdot	82,168	504,230	8,207	1,607	794	[Les12]

Figure 4.5 plots the average influence of peers on the real graphs used. Figure 4.6 shows the difference of the average influence between random and social mapping for the real

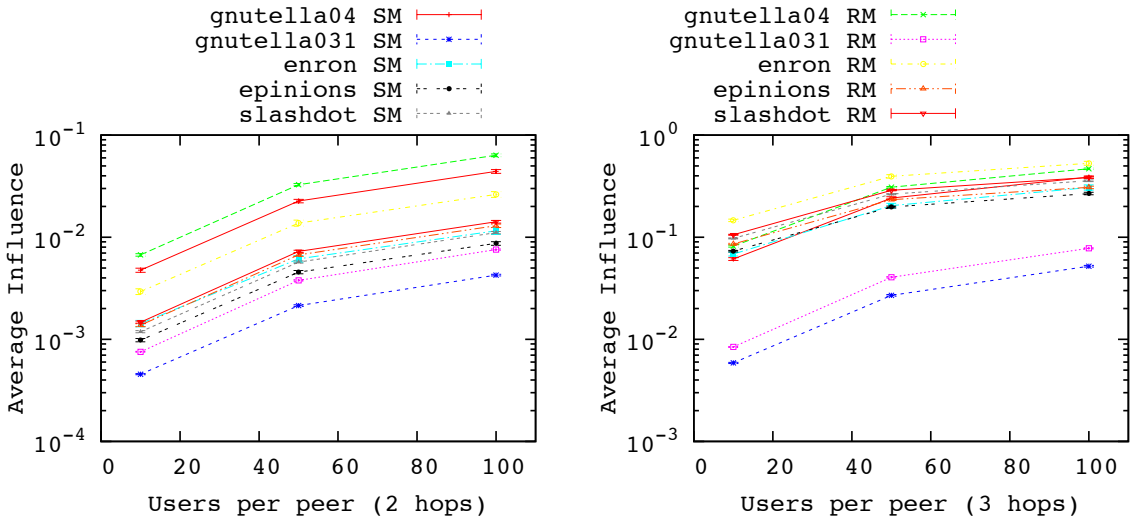

Figure 4.5: Average peer influence for random and social mappings for real graphs, for combinations of users per peer and number of hops per request. (Note: y-axis in log scale).

graphs, for clarity. Figure 4.7 plots the CDFs of the average influence of peers. From these results we formulate several lessons.

Lesson 2: *Increased number of users per peer and decreased social graph size directly increase peer influence on requests.* The results on real graphs (Figures 4.5, 4.6 and 4.7) verify the intuition that more users mapped on a peer increase the peer's influence on requests, regardless of the type of mapping (random or social). However, the difference between the two mappings (as seen in Figure 4.6) also increases with larger communities mapped on peers, as well as increased number of hops traversed. This means that socially-aware mappings increase the peer influence in a smaller rate than random mappings, further supporting Lesson 1. Moreover, malicious peers are more effective in small networks, since they can serve and thus influence a larger portion of requests. For example, *Gnutella04* exhibits a higher average influence than *Gnutella31*, which even though has the same topological characteristics (as also seen by the peer influence profile in Figure 4.7), it's 6 times larger in size and thus peers have less opportunity to serve (portions of) requests.

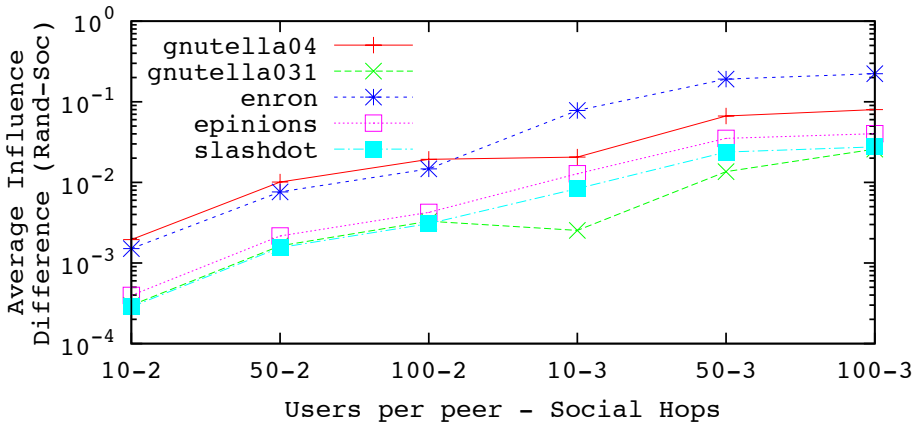

Figure 4.6: Average difference of peer influence between random and social mappings for real graphs. I.e., difference between random and social mapping lines in Figure 4.5), for combinations of users per peer and number of hops per request. (Note: y-axis in log scale).

Lesson 3: *The topology of the social graph affects the peer influence.* The size of the decentralized graph is not always an accurate predictor of the opportunity of peers to influence requests. It also depends on the particular application domain and topology of the network. For example, *Gnutella31* exhibits similar peer influence profile with *Gnutella04*, even though it has overall lower absolute values due to its larger size (as explain in Lesson 2). On the contrary, *Gnutella31* is smaller in size than *Slashdot* and *Epinions*, and yet exhibits overall lower peer influence as seen in the different peer influence profile in Figure 4.7.

Lesson 4: *The socially-aware distribution of a social graph onto peers allows the formation of "hot-spot" peers.* The maximum influence of each mapping is higher than its corresponding average for all networks (synthetic or real), and all average community sizes N and all n-hop requests. This reveals highly influential peers that control more requests flowing through the P2P topology than the average peer. We especially notice these peers for the social mapping of the small 1000-user graph. This is because users of high social degree centrality are closely connected with each other and more likely to be mapped together on the same peer [SBAG08] in the social than in the random mapping.

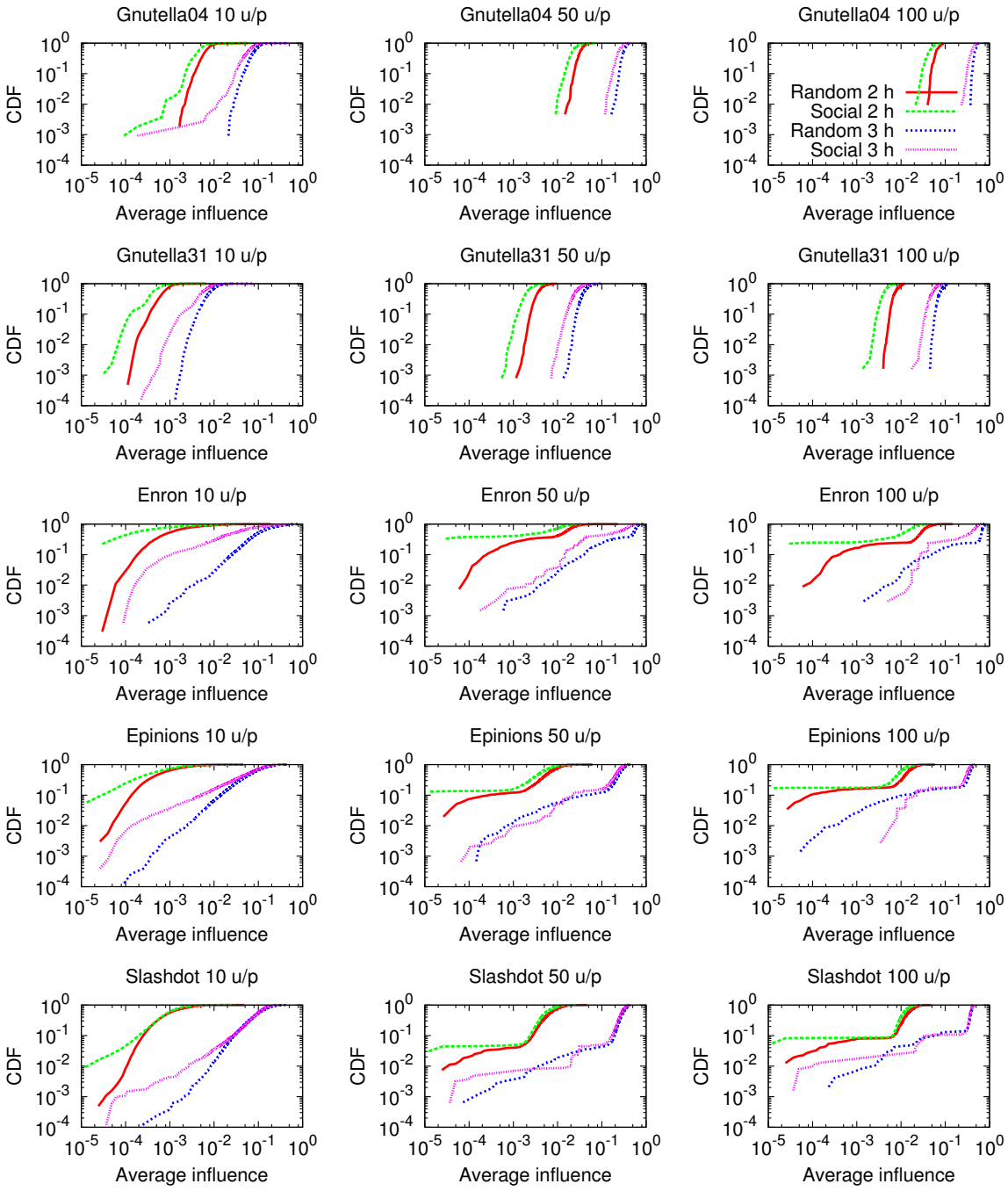

Figure 4.7: CDF of average peer influence for random and social mappings for real graphs, for combinations of users per peer and number of hops per request. (Note: axes in log scale).

Further, due to the small size of the graph, peers participate in more requests and the variability of peer influence is higher (especially for 3 hop requests). These peers can be identified based on the users mapped on them [KI11] and targeted for quarantine in the early stages of a malware outburst or used to disseminate faster and more efficiently security software patches to handle a malicious attack.

4.2.4 Peer Influence under Peer Collusion

In the third set of experiments we investigated the peer influence when peers collude with each other, i.e., they are controlled by the same attacker. We used the largest network *slashdot* and the social mappings created in the second experimental setup for average community size of N=10 users per peer. The two collusion types examined where the following: 1) a social-based collusion of neighboring peers (i.e., their mapped users are connected with each other over social edges), 2) a random-based collusion of random peers.

We seeded the collusion by selecting a random set of peers amounting to 1% of the total peers in the system. Then, we iterated over these peers expanding their collusion set depending on the type of collusion (social or random), until the overall malicious peers across all collusion sets amounted to a specific portion of the total network. We varied the overall fraction of peers colluding C in the range of 10% to 50% of the total peers.

Figures 4.8 and 4.9 plot the average influence of peers when colluding in a social or random way, respectively, to cover a portion C of the total number of peers in the system. Figure 4.10 plots both types of collusion on the same graph for easier comparison (note that peer influence is on the x-axis). We also compare average influence of the peers when colluding or not colluding, when the social graph is distributed with a random or social

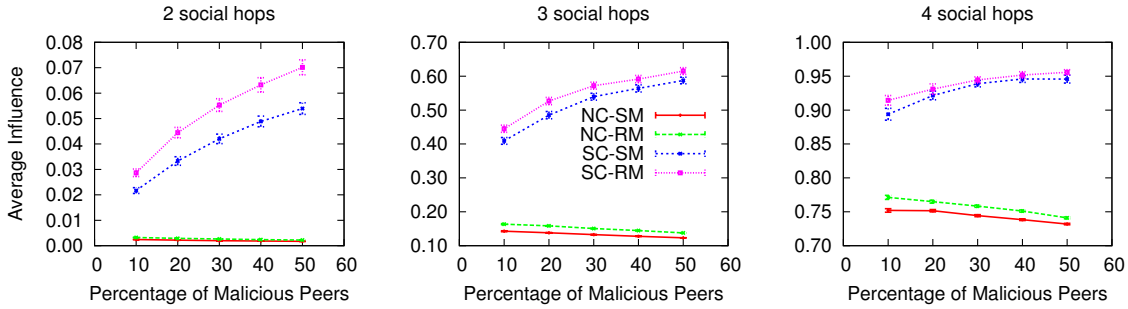

Figure 4.8: Average peer influence for random (RM) and social mapping (SM) of the *slashdot* network, for 10 users per peer and different number of hops per request. We vary the percentage of *social* collusion (SC) and compare with the no-collusion (NC) scenario. Error bars show 95% confidence intervals on reported average peer influence.

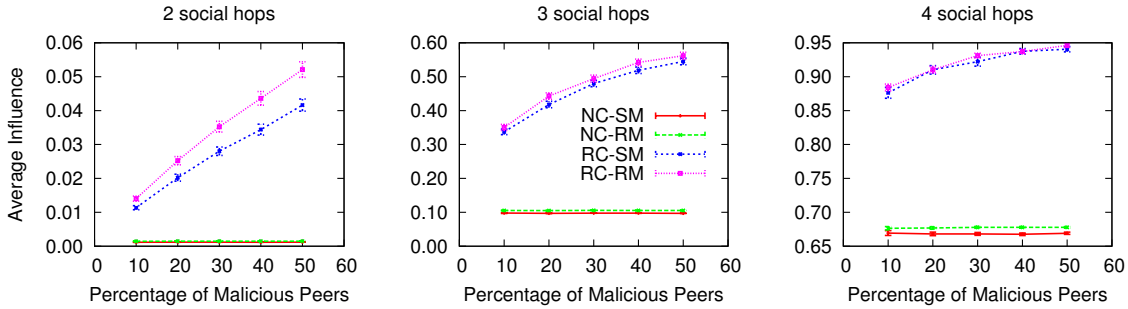

Figure 4.9: Average peer influence for random (RM) and social mapping (SM) of the *slashdot* network, for 10 users per peer and different number of hops per request. We vary the percentage of *random* collusion (RC) and compare with the no-collusion (NC) scenario. Error bars show 95% confidence intervals on reported average peer influence.

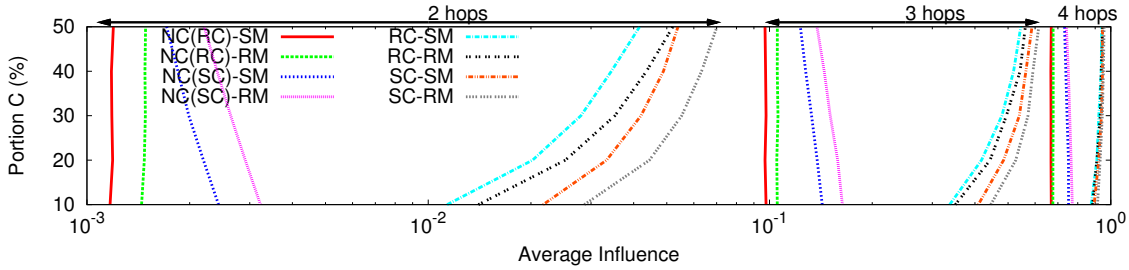

Figure 4.10: Average peer influence for random and social mapping of the *slashdot* network, for 10 users per peer and different number of hops per request. We vary the percentage of social and random collusion and compare with the no-collusion scenario.

mapping and for neighborhood requests of different number of hops. From these results we formulate several lessons.

Lesson 5: *Peer collusion attacks are less effective on socially-aware mappings.* Collusion of peers into groups increases their individual effectiveness when attacking the system. As seen from Figures 4.8, 4.9 and Figure 4.10, the average influence measured on collusion groups (SC or RC) is always higher than the average influence of their individual peer members when not colluding with each other (NC). However, a random distribution of the social graph onto peers (RM) forces requests to access data from more peers than in a social distribution (SM), and thus allows peers to control and influence a higher portion of inference requests, either if they are colluding (SC and RC) or not colluding (NC).

Lesson 6: *Random peer collusions are less effective on socially-aware mappings.* Generally, the average peer influence increases dramatically with the number of social hops of the neighborhood request: from less than 0.1 for 2 hops to more than 0.9 for 4 hops (as also shown in the previous two sets of experiments on the synthetic graph and real graphs). In the social collusion SC, the attack targets neighboring peers (i.e., their users are directly connected in the social graph). In this case, the attacker can achieve higher peer influence on requests than if the attack targeted random peers (random collusion RC). However, the difference between social and random collusion is limited to about 0.02 in 2 hop requests, increases to about 0.1 in 3 hop requests and decreases to about 0.05 in 4 hop requests.

Lesson 7: *Network hops affect more the peer influence than collusion size.* In either of the two collusion types and graph mappings, increasing the number of colluding peers increases the average peer influence of an attacker. However, the peer influence rate (change of peer influence) depends more on the number of social hops the request will traverse

the graph, than the size of the collusion set. Thus, we observe highest gains on the peer influence rate for a malicious attacker at 3 social hop requests than 2 or 4 social hops.

4.3 Attacks at the Application Level

Prometheus exposes the social graph to applications and services via an API implementing social inference functions. These inferences request access to users' social data while being executed on each peer and at each hop. Users in Prometheus can control access to their social data via fine-grained ACPs.

Let us assume an honest user *Bob* who deploys a default policy allowing access to his data only from his 1-hop (or direct) social neighborhood. Also assume a spammer *Alice* who does not have a reciprocal edge from *Bob* to her. If *Alice* submits a n-hop neighborhood request to *Bob*'s trusted peer to gain knowledge of his social relations within n social hops and spam them, she will not be able to receive any data regarding *Bob*. Such an ACP can protect *Bob* even in the case that his friends of friends' accounts (i.e., 2-hops from *Bob*) have been compromised by *Alice*. If *Alice* has a reciprocal edge from *Bob* to her, *Bob* could modify his ACP to allow partial access to his data which are relevant to the edge's label, and only if the edge has a minimum weight w.

However, being very strict to protect a user's data, such a draconian ACP comes with an unwanted tradeoff as it limits Prometheus' usability from applications seeking for social data of individuals across multiple hops ($\geq$ 2 hops). Therefore, honest users could carefully relax their ACPs, allowing inference requests originating from users outside their direct neighborhood to access portion of their social data only when the requests are forwarded through specific trustworthy direct friends and their owned peers. Such configuration allows the Prometheus service to become even more socially-aware, since

these contacts acquire the role of *gatekeeping* the social data of their friends and protect them from attacks.

In addition to applying defenses from the users' side using ACPs, application designers can make use of the Prometheus inference functions to reduce attacks on their users' data. For example, a spammer *Alice* could use the neighborhood request to retrieve the list of users within n hops from her and try to spam them with emails. An email application could filter spam by allowing incoming emails only when the receiver is reciprocating a social edge to the sender and with appropriate label and weight. Furthermore, the application could use knowledge from the 2-hop neighborhood of the receiver to decide if the incoming email is spam or not [GKF+06].

Chapter 5: Projection Graphs [1]

In a P2P system such as Prometheus, a social graph (SG) is decentralized among the available peers in the system. This decentralization leads to the emergence of the projection graph (PG) [KI11]. In order to provide better intuition on the projection graph formation, we present two application scenarios different than Prometheus (Section 5.1). These scenarios, among others, motivate our study of the projection graph as a general tool for improving application and system performance. We then present a formal model for the projection graph that is used throughout the rest of this dissertation (Section 5.3).

5.1 Motivating Scenarios

People naturally form social communities with strong social bonds [WF94, FLGC02], that could translate into incentives for resource sharing within the community [AG07, IRSNF11]. In our study, we assume that there exists a community peer (e.g., provided by a member of the social community) that stores the social data of all community members.

The projection graph emerges when the social data of all users in the graph are distributed among the available peers in the system. We describe below two motivating scenarios of social graph decentralization and the emergence of the projection graph. These scenarios

[1] Portions of this work have been previously published in [KI11] and are utilized with permission of the publisher.

help formally define the projection graph data structure and motivate the study of its network properties and how they compare to the distributed social graph's properties.

5.1.1 Civilian Networking in Large-Scale Disaster

After a natural disaster of large proportions, most of the communications and IT infrastructure is destroyed. Survivors cluster around small communities such as families, local organizations and community centers in small villages, etc. To help with the dissemination of emergency information as well as the organization of search-and-rescue operations, the authorities equip these communities with some basic IT and communication equipment (e.g., commodity servers, GSM/Wifi networks).

Community members input on these machines (peers) their health status, as well as the status (e.g., alive, injured, deceased or unknown) of their close family and friends living with them or in close distance in the area of the disaster. This information represents the edges of a social graph that connects users in the same geo-localized community. Individuals may also input information about friends located in other communities along with any useful detail regarding their status (e.g., last time seen, etc).

The community machines are connected with each other over wired and wireless networks and form a rudimentary P2P network. In this way, a basic social network of civilians is formed and their social information (along with others that might still be missing) is stored on these machines in a decentralized fashion.

5.1.2 Player Networking in Online Games

Online gaming platforms built for multi-player games (such as Steam [Val12], Battlefield [Bat12] and Minecraft [Min12]) allow players from around the world to run servers (peers) that host multi-player gaming sessions and individual players or teams can participate in each session. Players typically choose a server as their favorite, due to low network delays or the player community on the server.

Many of the in-game interactions and social interactions between players are stored on the server as meta-game data. Occasionally, gamers play on different servers, for example when their "home" server is offline or overloaded. In this way, they also form social connections with players from other communities. Consequently, the meta-gaming social graph connecting players is distributed on thousands of gaming servers around the world.

5.2 Projection Graph Emergence

In both scenarios, a projection graph emerges when a social graph is partitioned into social communities and distributed across the P2P network on community-owned peers. Social edges connecting users of the same community remain *within* the community peer, whereas social edges connecting users of different communities correspond to edges *between* peers in the projection graph.

In effect, peers acquire particular network properties in this graph based on the network properties of the users storing their social data on them. But how do these network properties of the peers in the projection graph compare with the network properties of the users in the social graph? And how can they be used to improve the performance of an application traversing such a decentralized social graph? These questions motivate the rest of this work.

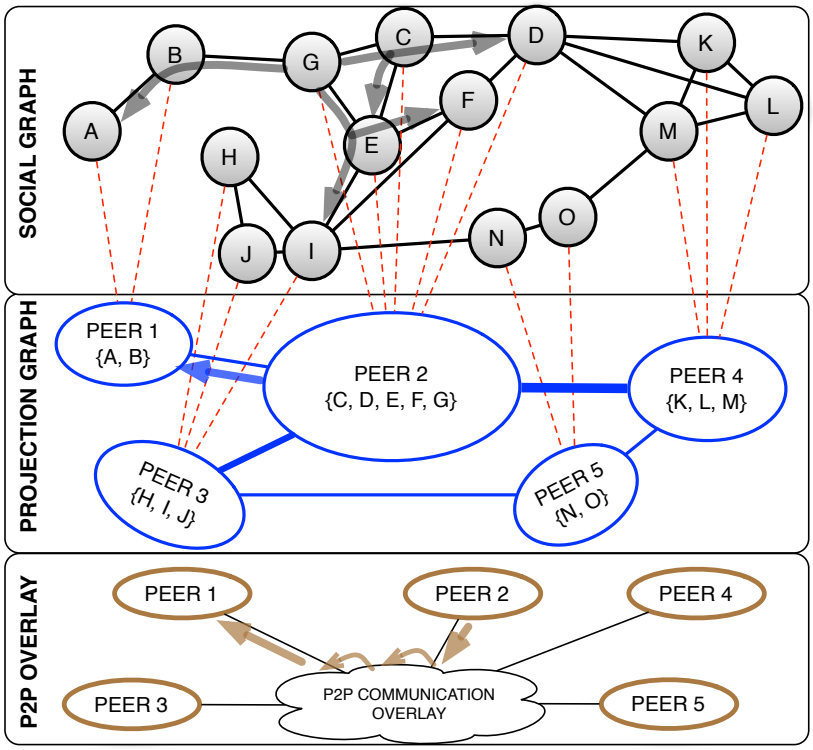

Figure 5.1: An example of a social graph distributed on a set of peers which are organized in a P2P overlay. Users A–O are shown in small grey circles, peers 1–5 are shown in large blue circles in the PG and brown circles declare the peers organized in a P2P overlay. Users are connected with social edges illustrated with black lines, blue lines correspond to P2P edges with different weights (declared by their width) and red dashed lines correspond to mappings of users onto peers storing their data. ©2011 IEEE.

5.3 Projection Graph Model

We consider a social graph as an undirected and unweighted graph $SG = (V_G, E_G)$, where V_G is the set of users and $E_G \subseteq V_G \times V_G$ is the set of edges that represent the social ties between users (top layer of Figure 5.1). The existence of an (unweighted, undirected) edge between two users u and v is denoted by $e(u, v) = 1$.

The projection graph in a P2P system emerges when the social graph is distributed on the P2P network (middle layer of Figure 5.1). The projection graph is an undirected, weighted graph whose nodes are peers responsible for a set of users in the social graph

and whose edges represent the social ties between the users mapped on different peers. We refer to a user u as "mapped" on a particular peer when the peer stores u's *social data* (the set of all social graph edges originating from u). The weight of an edge in the projection graph is given by the number of social graph edges that connect the users mapped on the end peers.

Formally, a projection graph is represented by $PG{=}(V_P, E_P)$. V_P is the set of peers in the P2P network. For each peer $P_i{\in}V_P$, Q_i is the set of users mapped on P_i. $E_P \subseteq V_P \times V_P$ is the set of edges in the projection graph, to which we refer to as P2P edges. A P2P edge between peers P_i and P_j, where $i \neq j$, is formally defined as follows:

$$(P_i, P_j) \in E_P \text{ iff } \exists u \in Q_i, \exists v \in Q_j, i \neq j \text{ s.t. } (u, v) \in E_G \tag{5.1}$$

A P2P edge (R_{ij}) represents the set of social edges between the users mapped on peer P_i and the users mapped on peer P_j, or more formally,

$$R_{ij} = \{(u, v) \in E_G | u \in Q_i, v \in Q_j, i \neq j\} \tag{5.2}$$

The weight of an edge between two peers P_i and P_j is denoted by $e(P_i, P_j) = |R_{ij}|$, and $e(P_i, P_i) = 0$ by definition.

Figure 5.1 presents a scenario in which users *A*–*O* store their data on peers 1–5, and each peer has access to all its users' data. The P2P edge R_{24} has the weight $e(2, 4){=}3$ given by the social edges (D, K), (D, L) and (D, M).

In this model, a user's social data is stored on *at least* one peer, and each peer stores *at least* one user's social data. Each peer maintains the union of the social data of the users mapped on it. Depending on the social relationship of these users, this union can be

anywhere from a disjoint set of edges, as proposed in [BSVD09, SVCC09, CMS09], to a connected subgraph, as proposed in [KFA+10a].

The projection graph is independent from the P2P overlay, as explained in the following example. Assume an application wants to find the users in the 2-hop social neighborhood of user G (i.e., friends and friends of friends). The application could search for these users by traversing the graph over the social connections this user has with the rest of the users (black arrows following social graph edges, top layer in Figure 5.1). Since the social graph is distributed on top of a P2P network, these requests will be routed from peer to peer in a manner informed by the topology of the social graph.

Therefore, the traversal of the social graph dictates peer 2 sending a message to peer 1 (blue arrow in the projection graph, middle layer in Figure 5.1) to request information regarding the 1-hop connections of user B. This application request might translate into multiple routing hops between the peers in the P2P communication overlay (e.g., DHT) before the destination peer is located and the request is delivered (brown arrows in the P2P overlay, bottom layer in Figure 5.1). We call such P2P systems *socially-informed* because the communication pattern between peers is determined by the social graph topology and how it is projected on the peers, and can be seen independently of the P2P overlay.

Chapter 6: Social Network Centrality Measures in Projection Graphs [1]

In this Chapter, we formally define the degree centrality, node betweenness centrality and edge betweenness centrality for a social graph and its corresponding projection graph. These three classical social network centrality measures reveal many important social properties of a network under study:

- *Degree Centrality* of a node quantifies its visibility to the rest of the network and opportunity to influence and be influenced directly, as shown in Figure 6.1 (Section 6.1).

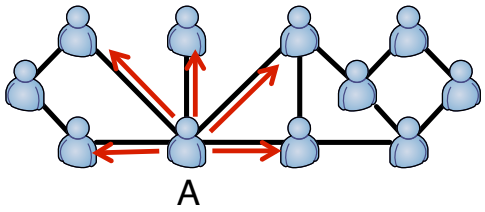

Figure 6.1: The degree centrality of user A is higher than other users in this example social graph.

- *Node Betweenness Centrality* quantifies a node's potential to control flows of information between pairs of nodes connected in the network over indirect paths, as shown in Figure 6.2 (Section 6.2).

[1] Portions of this work have been previously published in [KI11] and are utilized with permission of the publisher.

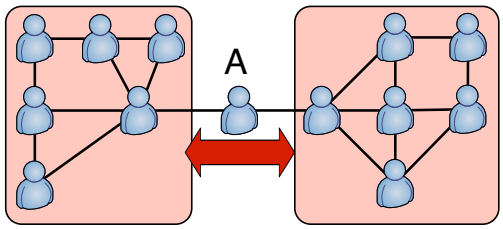

Figure 6.2: The node betweenness centrality of user A is higher than other users in this example social graph.

- *Edge Betweenness Centrality* quantifies an edge's potential to control flows of information between otherwise separate or distant parts of the network, as shown in Figure 6.3 (Section 6.3).

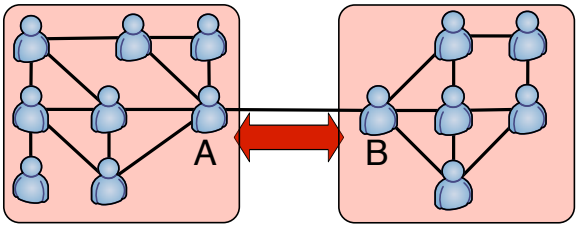

Figure 6.3: The edge betweenness centrality of the social edge connecting users A and B is higher than other edges in this example social graph.

For each measure, we study the connection between the social and projection graph and formulate research questions that we answer experimentally in the rest of this work. In the following, we assume that multiple users can be mapped on the same peer and a user can be mapped only on one peer.

6.1 Degree Centrality

The degree centrality [Nie74, Fre79] $C_D(n)$ of a node n in a graph is the number of edges that n has with other nodes. The degree centrality of a user u mapped on peer P_i can be expressed as the edges that u has with users on different peers than P_i, and the edges

that u has with users mapped on the same peer P_i:

$$C_D(u) = \sum_{\substack{v \in Q_j, \\ P_j \neq P_i \in V_P}} e(u,v) + \sum_{u \neq v \in Q_i} e(u,v), \forall u \in Q_i \tag{6.1}$$

We can express the degree centrality of a peer P_i as a function of the sum of the degree centralities of the users mapped on P_i, the sum of edges between users mapped on P_i and the sum of edges between users of peers P_i and $P_j, \forall P_j \neq P_i \in V_P$:

$$C_D(P_i) = \sum_{u \in Q_i} C_D(u) - \sum_{u \neq v \in Q_i} e(u,v) - \sum_{P_j \neq P_i \in V_P} (\sum_{\substack{u \in Q_i \\ v \in Q_j}} e(u,v) - 1) \tag{6.2}$$

Equation 6.2 allows us to analytically calculate the exact degree centrality of a peer if the peer can access its users' social connections and infer its P2P edges with other peers. However, it is generally difficult to determine the exact degree centrality of a peer when the peer is granted access to view only a user's degree centrality score but not the user's connections. Thus, a research question is the following:

Question 1: Can a peer estimate its degree centrality in the projection graph based only on the degree centrality score of its users in the social graph?

6.2 Node Betweenness Centrality

Betweenness centrality [Ant71, Fre77] $C_{NB}(u)$ of a user $u \in V_G$ is the sum of fractions of shortest paths between users s and t that pass through user u, denoted by $\sigma(s,t|u)$, over all the shortest paths between the two users, $\sigma(s,t)$:

$$C_{NB}(u) = \sum_{s \neq t \in V_G} \frac{\sigma(s,t|u)}{\sigma(s,t)} \tag{6.3}$$

Betweenness centrality $C_{NB}(P_i)$ of a peer $P_i \in V_P$ is the sum of fractions of *weighted* shortest paths between peers P_j and P_k that pass through P_i, denoted by $\lambda(P_j, P_k|P_i)$, over all the *weighted* shortest paths between the two peers, $\lambda(P_j, P_k)$:

$$C_{NB}(P_i) = \sum_{P_j \neq P_k \in V_P} \frac{\lambda(P_j, P_k|P_i)}{\lambda(P_j, P_k)} \tag{6.4}$$

When users are mapped on peers, the shortest paths between the users can be expressed as a combination of four basic categories, as illustrated in Figure 6.4. The first category reflects the shortest paths between two users s and t that pass through u and each user is mapped on a different peer. The second category (and its omitted symmetrical for $u \in P_V(i)$, $t \in P_V(i)$, $s \in P_V(k)$) reflects the shortest paths between s and t, when one of them is mapped on the same peer as u. The third category reflects the shortest paths between s and t when they are mapped on the same peer P_j, but different from u. The forth category reflects the case that all three users are mapped on the same peer P_i.

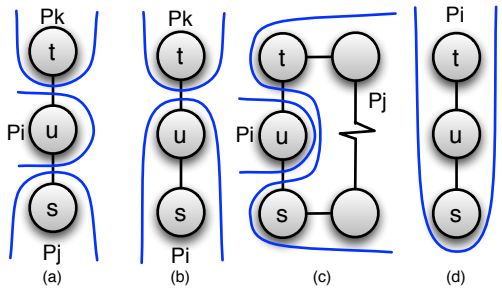

Figure 6.4: The four categories of shortest paths between two users s and t through u, when users are mapped on peers. ©2011 IEEE.

Thus, we can express the betweenness centrality of user $u \in Q_i$ as a combination of these main categories of shortest paths, as follows:

$$C_{NB}(u) = \sum_{P_j \neq P_k \in V_P} \Big(\sum_{\substack{s \in Q_j \\ t \in Q_k}} \frac{\sigma(s,t|u)}{\sigma(s,t)} + \sum_{\substack{s \in Q_i \\ t \in Q_k}} \frac{\sigma(s,t|u)}{\sigma(s,t)} + \sum_{\substack{s \in Q_j \\ t \in Q_j}} \frac{\sigma(s,t|u)}{\sigma(s,t)} + \sum_{\substack{s \in Q_i \\ t \in Q_i}} \frac{\sigma(s,t|u)}{\sigma(s,t)} \Big) \tag{6.5}$$

As demonstrated in eq. 6.5, it is difficult to analytically determine the peer betweenness centrality with respect to the centrality of its users due to the various types of shortest paths in which users participate, which are highly dependent on the way the social graph is decentralized onto peers. Also, the peer might not be granted access to traverse the P2P topology and calculate its exact betweenness centrality in the projection graph, for example due to user access policies on other peers or unavailability of peers. Assuming a peer is granted access to its users' betweenness centrality scores, a research question is:

Question 2: Can a peer estimate its node betweenness centrality in the projection graph, based only on the node betweenness centrality score of its users in the social graph?

6.3 Edge Betweenness Centrality

Betweenness centrality [GN02] $C_{EB}(e)$ of an edge $e \in E_G$ is the sum of fractions of shortest paths between two users s and t that contain e, denoted by $\sigma(s,t|e)$, over all the shortest paths between the two users, $\sigma(s,t)$, or more succinctly:

$$C_{EB}(e) = \sum_{s \neq t \in V_G} \frac{\sigma(s,t|e)}{\sigma(s,t)} \tag{6.6}$$

Betweenness centrality $C_{EB}(E)$ of a P2P edge $E \in E_P$ is the sum of fractions of *weighted* shortest paths between two peers P_i and P_j that contain E, denoted by $\lambda(P_i,P_j|E)$, over all *weighted* shortest paths between the two peers, $\lambda(P_i,P_j)$:

$$C_{EB}(E) = \sum_{P_i \neq P_j \in V_P} \frac{\lambda(P_i,P_j|E)}{\lambda(P_i,P_j)} \tag{6.7}$$

As with the node betweenness, the shortest paths between users that contain edge e can be divided into five categories, as illustrated in Figure 6.5. We omit the illustration of the symmetrical cases for (a), (c) and (d).

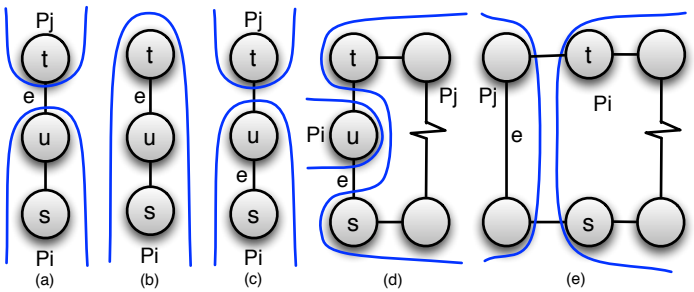

Figure 6.5: The five categories of shortest paths between two users s and t through e, when users are mapped on peers. ©2011 IEEE.

Based on the intuition of Figure 6.5, we can express the betweenness centrality of an edge e as follows:

$$C_{EB}(e) = \sum_{P_i \neq P_j \in V_P} \Bigg(\sum_{\substack{s \in Q_i \\ t \in Q_j \\ e \in R_{ij}}} \frac{\sigma(s,t|e)}{\sigma(s,t)} + \sum_{\substack{s \in Q_i \\ t \in Q_i \\ e \in R_{ii}}} \frac{\sigma(s,t|e)}{\sigma(s,t)} + \sum_{\substack{s \in Q_i \\ t \in Q_j \\ e \in R_{ii}}} \frac{\sigma(s,t|e)}{\sigma(s,t)} + \sum_{\substack{s \in Q_j \\ t \in Q_j \\ e \in R_{ij}}} \frac{\sigma(s,t|e)}{\sigma(s,t)} + \sum_{\substack{s \in Q_i \\ t \in Q_i \\ e \in R_{jj}}} \frac{\sigma(s,t|e)}{\sigma(s,t)} \Bigg) \tag{6.8}$$

Using similar argumentation with the node betweenness centrality, a research question is:

Question 3: Can we estimate the edge betweenness centrality of an edge in the projection graph based only on the edge betweenness centrality scores of its edges in the social graph?

Chapter 7: Experimental Study of Projection Graphs [1]

In this Chapter we study the network centrality properties of the projection graphs. For this, we construct in Section 7.1 projection graphs from real social networks by decentralizing them onto multiple peers using a social-based mapping. This mapping takes into account the naturally formed communities of users and assigns one community per peer.

We study the centrality measures of the projection graphs formed and how they relate to those of the social graphs distributed on peers (Section 7.2). Furthermore, we investigate how well we can predict top centrality peers from top centrality users (Section 7.3).

7.1 Projection Graphs From Real Networks

In order to answer the research questions from Chapter 6, we used five real networks that represent application domains where the social data of users is assumed to be distributed on a P2P network, as proposed in Section 5.3. This assumption enables us to extract projection graph topologies from these real networks, with varying number of users per peer.

[1] Portions of these results have been previously published in [KI11] and are utilized with permission of the publisher.

7.1.1 Network Description

The five real networks used to generate projection graphs cover diverse domains, such as file sharing (gnutella) (from [RIF02]), email communications of company employees (enron), trust on consumer reviews (epinions) and friendships in a news website (slashdot) (from [Les12]) and have sizes between $10K$ and $100K$ nodes.

The *Email-Enron* is an email communication network generated from emails sent within the Enron company. Vertices are email addresses in the data set and undirected edges represent at least one email exchange between the end vertices.

The *P2P-Gnutella04 and P2P-Gnutella31* are two time-snapshots of the Gnutella peer-to-peer file sharing network on August 4th and August 31st, 2002. Nodes represent hosts in the Gnutella network topology and edges represent connections between the Gnutella hosts. Even though the Gnutella networks are not social networks like the other three, we use them because they exhibit social properties [RIF02] and we have two instances of different sizes, enabling us to study the variation of the social network metrics with the network size.

The *Soc-Epinions1* is a *who-trusts-whom* online social network of the general consumer review site "Epinions.com". Members of the site can decide whether to "trust" each other. Nodes represent the members and edges represent the trust relationships among members. All the trust relationships interact and form the "Web of Trust," which is then combined with review ratings to determine which reviews are shown to the user.

The *Soc-Slashdot0922* is a network containing friend/foe links between users of "Slashdot.org", a user-contributed, technology-related news website. Users can tag each other as friends or foes using the "Slashdot Zoo" feature.

We consider all networks undirected and unweighted and used only the largest connected component (LCC) from each graph to ensure reachability between all pairs of users and peers. Table 7.1 presents a summary of the networks' size with respect to the full network and the largest connected component found.

Table 7.1: Summary information of the real networks used in the projection graph experimental study. ©2011 IEEE.

Network (abbreviation)	Number of Users LCC (original)	Number of Edges LCC (original)	Source
P2P-Gnutella04 (gnutella04)	10,876 (10,876)	39,994 (39,994)	[RIF02]
Email-Enron (enron)	33,696 (36,692)	180,811 (183,831)	[Les12]
P2P-Gnutella31 (gnutella31)	62,561 (62,586)	147,878 (147,892)	[RIF02]
Soc-Epinions1 (epinions)	75,877 (75,879)	405,739 (405,740)	[Les12]
Soc-Slashdot0922 (slashdot)	82,168 (82,168)	504,230 (504,230)	[Les12]

7.1.2 Mapping Users onto Peers

To study the properties of projection graphs, we first identified social communities on each social graph and then mapped each community to a peer. Social edges between communities were transformed into weighted P2P edges. Communities were identified by using a modified algorithm of the Louvain method [BGLL08] for fast community detection in large networks.

This method first splits users into very small communities, and then iteratively reassigns users to other communities and merges them in order to improve the overall *modularity* score. It stops this process when the modularity score doesn't improve between two consecutive iterations.

The modularity Q [NG04] of a partition in a graph measures the density of the links inside communities as compared to links between communities, and is defined as follows:

$$Q = \frac{1}{2m} \sum_{i,j} \left[A_{ij} - \frac{k_i k_j}{2m} \right] \delta(c_i, c_j) \tag{7.1}$$

where A_{ij} represents the weight of the edge between users i and j, $k_i = \sum_j A_{ij}$ is the sum of the weights of the edges attached to vertex i, c_i is the community to which vertex i is assigned, the δ-function $\delta(u, v)$ is 1 if $u = v$ and 0 otherwise and $m = \frac{1}{2} \sum_{ij} A_{ij}$. The value of the modularity lies in the range $[-1, 1]$. It is positive if the number of edges within groups exceeds the number expected on the basis of chance.

The Louvain method detects communities very fast, even for graph sizes in the order of millions of users, with a very wide range of community sizes, i.e., a lot of small groups of 2–10 users, as well as very large groups in the order of 1000s users. The largest community usually represents the *core* of the network whereas the smallest ones, loosely connected to the core, reflect the *whiskers* of the network [LLDM08]. Instead, we would like to have communities with a size in a realistic range, avoiding these large outliers and reducing bias on the calculation of social network properties of the P2P topologies.

One way to split these large communities is to modify the Louvain method to reassign users to communities only if the resulting communities are below a *max-size*. However, this option interferes with the way the Louvain method works, and prohibits the modularity from reaching the best possible value. An alternative way is to consider the communities exceeding the *max-size* as individual subgraphs and recursively call the Louvain method on these subgraphs. We call this the "Recursive-Louvain" method. Even though this method also partitions the graph into communities which lead to sub-optimal overall modularity score, it allows for the optimal modularity to be reached within sub-communities.

We tested the Recursive-Louvain method with values for the *max-size=10, 100, 500, 1000* and decided to use *max-size*=100 as it offered a local minimum for the standard deviation

of size of communities. The value of *max-size*=100 supports the findings in [LLDM08] which found that the best communities with respect to *conductance* are relatively small with sizes up to 100 users per community.

Since we consider that a community is mapped on a peer contributed by a user, it would be unrealistic to map a very large community on one peer. Thus, we consider the communities exceeding a *max-size* as individual subgraphs and recursively apply the Louvain method on these subgraphs. We call this technique "Recursive-Louvain" and tested it with values for the *max-size*=10, 100, 500 and 1000. We used *max-size*=100 as it offered a local minimum for the standard deviation of size of communities. The value of *max-size*=100 supports the findings in [LLDM08] according to which the best communities with respect to *conductance* are relatively small, with sizes up to 100 users per community.

We compare in Table 7.2 the summary statistics of the formed communities with the Louvain and Recursive-Louvain methods for *max-size*=100. Using the Recursive-Louvain method we successfully split most of the larger communities into smaller ones (4 to 50 times smaller) and improve the overall standard deviation of the size of communities formed from each network (some cases 6 to 30 times smaller).

Table 7.2: Summary statistics for communities identified with the Louvain (L) and Recursive-Louvain (RL) methods on the real networks used. ©2011 IEEE.

Social Network	Number of Communities L / RL	$\overline{\lvert Q_i \rvert}$ L / RL	Standard Deviation L / RL	Min/Max L (RL Max)
gnutella04	2384/3013	4.0/3.6	23.0/3.5	2/1299 (89)
enron	2434/4303	11.9/7.6	139.1/15.7	2/4845 (1204)
gnutella31	13425/14385	4.4/4.3	3.0/2.8	2/3594 (97)
epinions	8481/16404	7.1/4.6	196.4/7.9	2/15770 (484)
slashdot	6879/18846	9.5/4.3	225.2/6.9	2/17012 (358)

This process effectively reduces the variability of the community size. However, if we take into account the new maximum sizes, the range is still reasonably large: communities with size of more than 400 users are still available and could reflect a large group of users utilizing the same computing resources (e.g., a highly populated gaming server). At the same time, the minimum and average community size remains either constant or within the same order of magnitude, for all social networks. This is a clear indication that the Recursive-Louvain method successfully reduces the number of outlier large communities without affecting the overall variability of the community size.

7.1.3 Community Size and Degree Variability

In our study, we investigate how the average number of users mapped per peer affects the estimation of the three social network measures, while using only local information on peers and P2P edges. Thus, we vary the average size of communities mapped on peers. Using the Recursive-Louvain method we identified a set of communities with fairly small average size (about 4–5 users), which were used as a baseline for our experimentation with increasing average size of communities.

To produce communities with increasing number of users, we incrementally merged the smallest, socially-connected communities, until we reached the desired average number of users per community (and thus peer) in the range of $10, 20, ..., 1000$ users/peer. During this process, a lot of very small communities of 2-5 users were gradually merged to form larger communities of average size in the range of $10, 20, ..., 1000$ users per peer. This merging process finds support from [LLDM08] where it is suggested that small communities can be combined into meaningful larger ones.

In order to avoid the case of a small community merging with an already large community, we enforced that the communities to be merged, as well as the newly formed merged

community, cannot be larger than the *max-size* of users. The parameter *max-size* was gradually increased as the average size of users per peer increased, to allow for larger average size of communities.

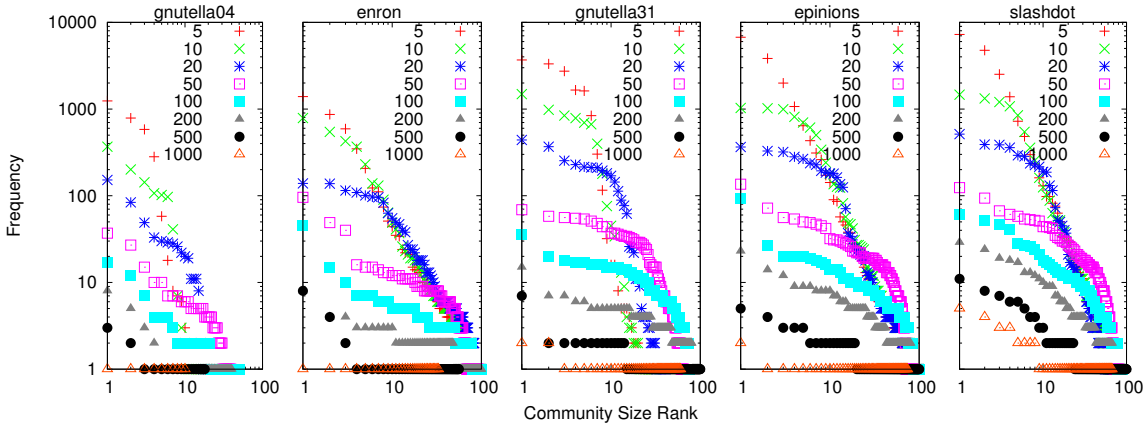

Figure 7.1: Distribution of the community size rank vs frequency observed in the different average size of communities and different real networks. ©2011 IEEE.

Figure 7.1 presents the rank distribution of the size of communities formed by this process, for the five networks studied and for various average community sizes, ranging from 5 to 1000 users/peer. The average community size for all networks for the range of 5–100 users/peer exhibits a *Zipf* distribution with two main exponents. The first one describes the size of community among the top 10 ranked sizes. The second one describes the rest of the sizes ranking lower. The *Zipf* distribution applicable in this range of community sizes shows that the communities formed maintain a power-law structure [CNM04]. When the average community size is increased above 100–200 users/peer, the *Zipf* distribution is no longer applicable, as the communities become more uniform.

Figure 7.2 presents the rank distribution of the peer degree in the projection graph for different average size of community, for each of the five networks examined. The user degree rankings of the networks (points marked as "1") follow a *Zipf* distribution demonstrating a power-law nature (especially the larger networks *epinions* and *slashdot*).

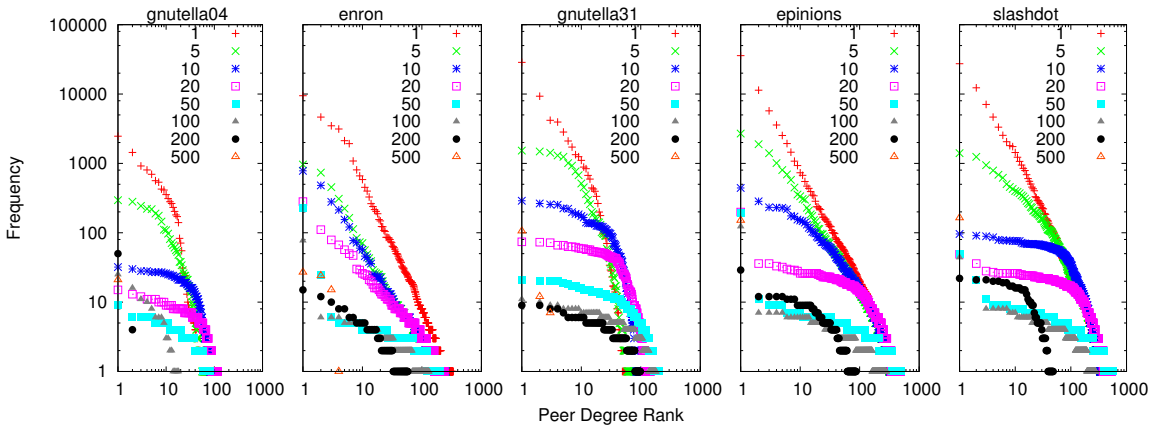

Figure 7.2: Distribution of the peer degree rank vs frequency observed in the different average size of communities and different real networks. ©2011 IEEE.

Similar to the community size rankings, the networks exhibit a strong 2-exponent Zipf distribution when the size of communities increases from 5 to about 20–50 users per peer, meaning that the topologies inherit social structure from the social graph distributed on the peers. Beyond a community size of about 100 users, the topology becomes significantly uniform: most peers exhibit a similar degree, thus degree rankings show similar frequency. This means they all have about the same number of connections and they form tightly connected groups or cliques. This effect intensifies as the average community size increases to 1000 users per peer.

7.2 Centrality Measures in Social vs. Projection Graph

As demonstrated by the expressions in Chapter 6, it is not easy to answer analytically the questions stated because of the various terms intercorrelated in the calculation of each centrality metric and the way users are mapped on peers.

In this section we experimentally examine how each of the three centrality metrics for a peer depends on the number of users mapped on the peer and how it correlates and if

it can be estimated by their cumulative centrality metric. We study these associations on the five real graphs and their extracted topologies, as explained in Section 7.1. In particular, we are interested to identify within what range of number of users per peer this estimation maintains high accuracy.

7.2.1 Estimation of Centrality Measures

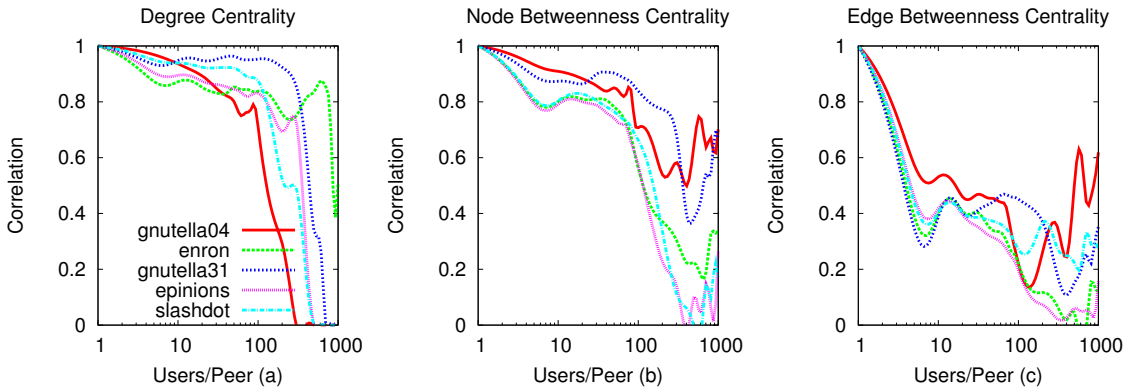

Figure 7.3: Correlation of cumulative normalized centrality scores of users vs normalized centrality scores of peers for Degree, Node Betweenness and Edge Betweenness Centrality. ©2011 IEEE.

Figure 7.3 presents for each metric the Pearson correlation of the scores of peers and cumulative scores of users per peer, with respect to the average number of users per peer. Specifically, we calculate the correlation for each metric based on the tuple $\{A, B\}$ of scores per peer (or edge): (A) the cumulative centrality of users (or social edges) mapped on a peer P_i (or P2P edge R_{ij}), and (B) the centrality of the corresponding peer P_i (or P2P edge R_{ij}) in the resulting topology. More concisely,

$$\text{Degree Centrality: } \{\sum_{u \in Q_i} C_D(u), C_D(P_i)\} \tag{7.2}$$

$$\text{Node Betweenness Centrality: } \{\sum_{u \in Q_i} C_{NB}(u), C_{NB}(P_i)\} \tag{7.3}$$

$$\text{Edge Betweenness Centrality: } \{\sum_{e \in R_{ij}} C_{EB}(e), C_{EB}(R_{ij})\} \tag{7.4}$$

The correlation is calculated by taking into account the tuples across all peers (or P2P edges) in the network, given a particular ratio of users per peer.

We observe that, for most of the networks, the correlation of the degree and node betweenness centrality remains fairly steady and high overall (> 0.8) for communities of less than 100-200 users. From that point on, the correlation decreases rapidly. This trend is generally consistent across all sizes and types of real graphs, but some networks present an outlier behavior.

For degree centrality, *gnutella31* maintains a high correlation up to 300 users/peer before the steep drop. For node betweenness centrality, the two *gnutella* networks demonstrate some extended high correlation up to 300 users/peer. The edge betweenness centrality drops significantly with the increase in community size, demonstrating that it is more sensitive to this parameter than the degree or node betweenness centrality. Next, we elaborate on the details behind the correlation performance of each centrality metric.

Figures 7.4, 7.5 and 7.6 compare the average degree centrality (DCP), node betweenness centrality (NBCP) and edge betweenness centrality (EBCP) for peers, with the respective cumulative centrality metric for users mapped on peers, respectively. Figures 7.4 and 7.5 show that the degree and node betweenness centrality of peers increase with respect to the average size of community.

This means that adding more users to a peer directly affects the centrality of the corresponding peer according to these metrics. We explain this as follows: by increasing the

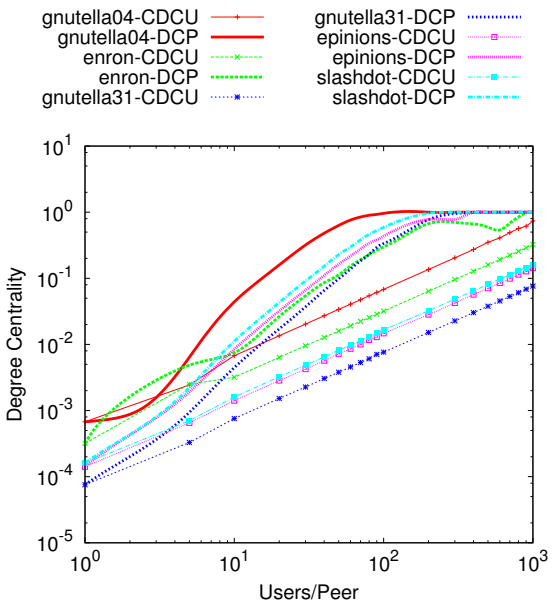

Figure 7.4: Comparison of cumulative normalized scores of users (point lines) vs average normalized scores of peers (smoothed lines) for Degree Centrality. ©2011 IEEE.

size of the communities, we reduce their number. In effect, more users mapped on a peer translates to more social edges with new peers, thus possible more P2P edges to other peers (i.e., higher degree centrality), as well as opportunity of the peer to participate in more shortest paths (i.e., higher betweenness centrality).

Figure 7.6 shows that the cumulative edge betweenness centrality of social edges between peers does not change for a range of size of communities. This is because when increasing the community size from 1 to about 50 users per peer, more social edges are mapped *within* the peers instead of *between* peers (as demonstrated in Figure 7.7). Within this range, the weighted edge betweenness centrality of the P2P edges decreases: the number of peers is reduced, and new edges between peers distribute the betweenness centrality of P2P edges across multiple paths, thus the P2P edges lose importance (in terms of edge betweenness).

As the number of users mapped on the same peer increases, the degree and node betweenness centrality of peers reach a maximum point. For the degree centrality this can

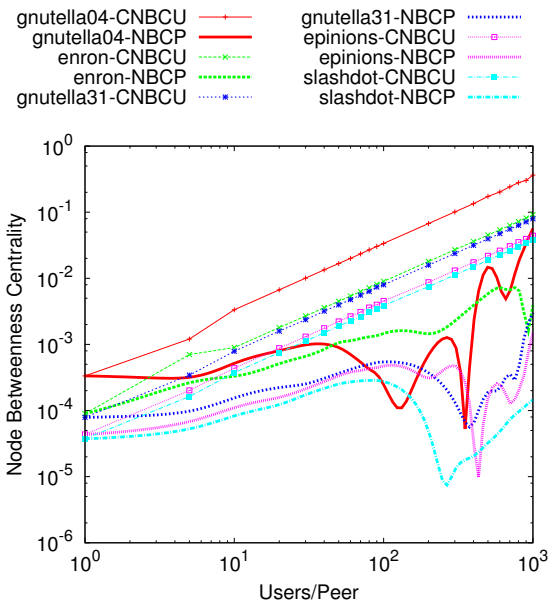

Figure 7.5: Comparison of cumulative normalized scores of users (point lines) vs average normalized scores of peers (smoothed lines) for Node Betweenness Centrality. ©2011 IEEE.

be seen at the point where the slope is steepest (e.g., the *gnutella04* topology reaches a maximum average degree centrality at about 55 users per peer, whereas the *slashdot* topology at about 90 users per peer). From equation 6.2, when the average size of community per peer $\overline{|Q_i|}$ increases, the second and third terms increase as well but at a different rate than the first term, thus the difference of them becomes least at this maximum point.

At this point, the network is optimally divided in communities mapped on peers which exhibit highest average degree and node betweenness centrality. For the edge betweenness, this is a turning point: between 50 and 100 users per peer, more social edges are mapped on P2P edges (i.e., between peers), in effect reversing the decline observed in Figure 7.6.

Increasing further the average community size decreases rapidly the peer degree centrality to very small values (also verified by the flat distribution of peer degrees in Figure 7.2).

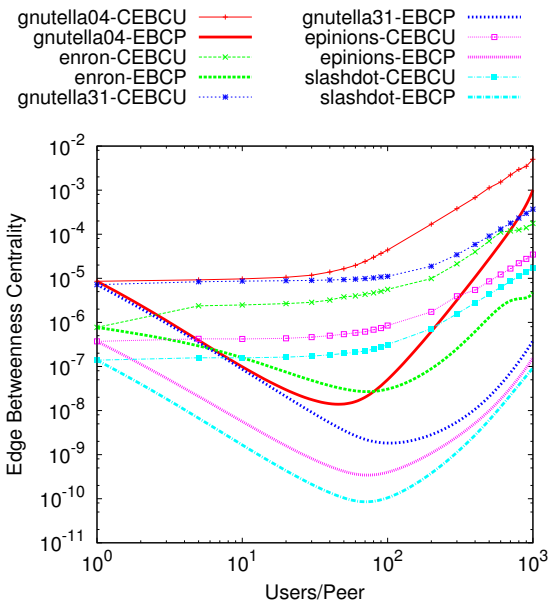

Figure 7.6: Comparison of cumulative normalized scores of user edges (point lines) vs average normalized scores of peer edges (smoothed lines) for Edge Betweenness Centrality. ©2011 IEEE.

In addition, the opportunity to influence information flows (due to high node betweenness) is distributed uniformly across all peers since they start forming a small, tightly connected graph. For the smallest network *gnutella04*, this drop takes effect quickly at about 60 users per peer, whereas for larger networks, like *epinions* and *slashdot*, at about 200–500 users per peer.

At the same time, even though the betweenness centrality of P2P edges increases, the opportunity to influence information flows (due to high edge betweenness) is distributed evenly across very few P2P edges. Eventually, by increasing even further the community size, the peer degree reaches 0, since at that point all the users are mapped on one peer and this peer has no inter-peer edges. It is important to note that depending on the application domain, the network properties of the topology may vary, even for seemingly small networks such as the *enron* email graph in comparison to *slashdot* or *epinions* graphs.

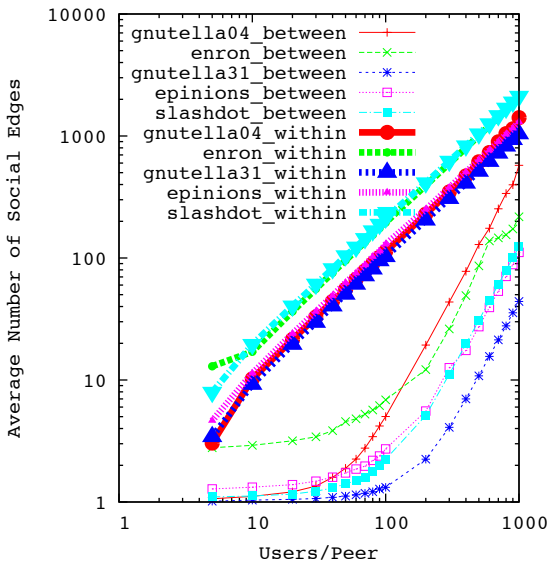

Figure 7.7: Average number of social edges found within a peer (thick lines) or between two peers (thin lines).

Figures 7.4– 7.6 help explain the correlation performance of Figure 7.3. Up to the maximum point for degree and betweenness centrality, the values of each pair of metrics increase with the addition of more users on each peer, and thus, the correlation is high overall. After this point, there is rapid decrease in the centrality scores of peers but not for the cumulative scores of users, and this reverse relationship causes the steep drop in the correlation of the respective measures. For the edge betweenness, the correlation drops early, as there is a high deviation between the P2P and social edge betweenness centrality scores (as explained earlier).

7.2.2 Applicability of Results

From our experiments on these networks we formulate the following lessons on peer centrality in projection graphs.

Lesson 1: *Increasing the community size directly effects the centrality of peers.* The increase of the average community size has an immediate effect on the topology and thus on the social network measures of each peer. From our experiments we identify a turning point where the degree and node betweenness centralities of peers reach a maximum. Before this point is reached, the projection graph resembles more closely the social graph it projects. Thus, the correlation of social network metrics between users and peers is highest and the two measures for peers can be estimated with good accuracy by the cumulative scores of users. When this point is reached, the peers gain maximum opportunity to influence the information flows passing through them. After this point, the topology loses any social properties, becomes a highly connected network and the peers acquire equal opportunity to participate in social graph traversals.

Lesson 2: *The cumulative centrality of users mapped on a peer can be used to estimate the peer's centrality.* Users mapped on a peer reflect their importance in the social graph onto their peer in the projection graph topology in two ways: either directly by connecting their peer with other peers (degree centrality), or indirectly by situating their peer on multiple shortest paths between other peers (betweenness centrality).

For small and medium size communities, we observe high correlation between users and peers for both of these centrality metrics. Thus, the centrality scores of users acquired from local information available to peers are good predictors of the importance a peer will have in the network. In effect, this means a peer can estimate with high accuracy its importance in the projection graph without the need to traverse the P2P network, which might be difficult due to network size, peer churn and user data access policies on other peers.

Lesson *3: The cumulative centrality of social edges between two peers is not a good estimator for the centrality of their P2P edge.* There are high betweenness social edges in the social graph that control significant information flow between different parts of the

network. However, the importance of a social edge reflected on the P2P network depends on the way the social edges are mapped in the topology. From our experiments we observe that when more social edges are mapped within peers instead of between peers, the importance of social edges between peers becomes less significant and the estimation of edge betweenness centrality of P2P edges is less accurate.

7.3 Estimating Top Centrality Scoring Peers

In very large social networks, calculating the exact node betweenness centrality of users can be intractable. Instead, we could use approximate measures such as κ–path centrality [ATK$^+$11, KAS$^+$12], to identify a small percentage of users that exhibit high betweenness in the social graph.

Next, we experimentally study if we can identify peers that exhibit high centrality in the projection graph, given a set of users with relatively high degree or node betweenness centrality in the social graph. We investigate this proposal with two methods on the projection graphs constructed from the real networks.

In the first method, we pick users in *top-N*% centrality (degree/betweenness), and create a unique set U of their peers (set U, size k=$|U|$). Then, we pick k peers with the highest centrality (set P, size k=$|P|$). Finally, we compare the sets U and P to find the overlap of same peers.

In the second method, we pick communities in *top-N*% cumulative score of centrality (degree/betweenness), and create a set C of their peers (set C, size q=$|C|$). Then, we pick peers in *top-N*% centrality (set P, size q=$|P|$). Finally, we compare the sets C and P to find the overlap of same peers.

Figure 7.8 shows for all the networks used the percent overlap of peers for the degree centrality using the two methods described. We repeated the same experiments for the node betweenness centrality and report in Figure 7.9 the percent overlap of peers for the node betweenness.

The results for the degree centrality are very similar to the node betweenness centrality for all networks and for both methods. We attribute this to the high correlation between the two metrics, which is common in social networks. Depending on the network, using the first method and *top*-5% of users, we can achieve 60–100% accuracy in identifying important peers for communities which are within the range of 5–100 users per peer. Above this range, the projection graph topology becomes too densely connected and the centrality of peers is uniformly distributed across a small number of peers.

Also, users of high importance are usually socially close to each other [SBAG08] and since the first method identifies only those users, there is a higher probability these users are mapped together on just a handful of peers. The overlap shows we can accurately predict the corresponding top peers within a range of users per peer.

The second method achieves high accuracy but for a tighter range of users per peer. In comparison to the first method, the second method degrades in performance when increasing the community size. This means the cumulative score of users is more sensitive to the size of grouping of users per peer than the individual scores of users.

From these experiments we formulate the following lesson. Top degree or betweenness centrality peers affect system performance and security. However, deterministically identifying high betweenness peers in projection graphs is infeasible not only because of the networks' large scales but also because of their dynamic nature caused by high peer churn. Our experiments show we can identify with high accuracy the peers with high degree or betweenness centrality, by knowing the *top* 5% degree or betweenness centrality users in the social graph.

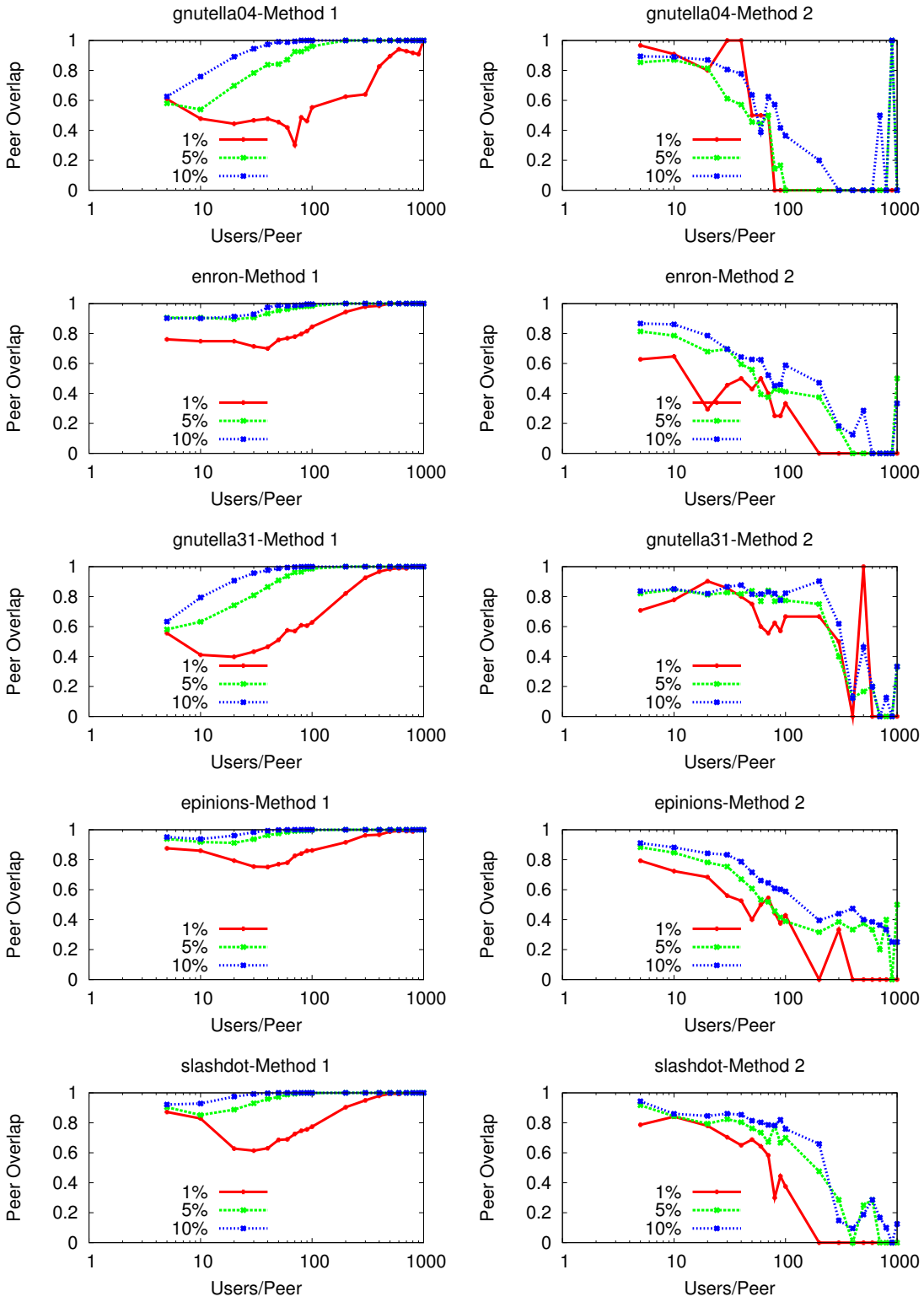

Figure 7.8: Percent overlap of peers for top $N\%$ degree centrality in the networks used.

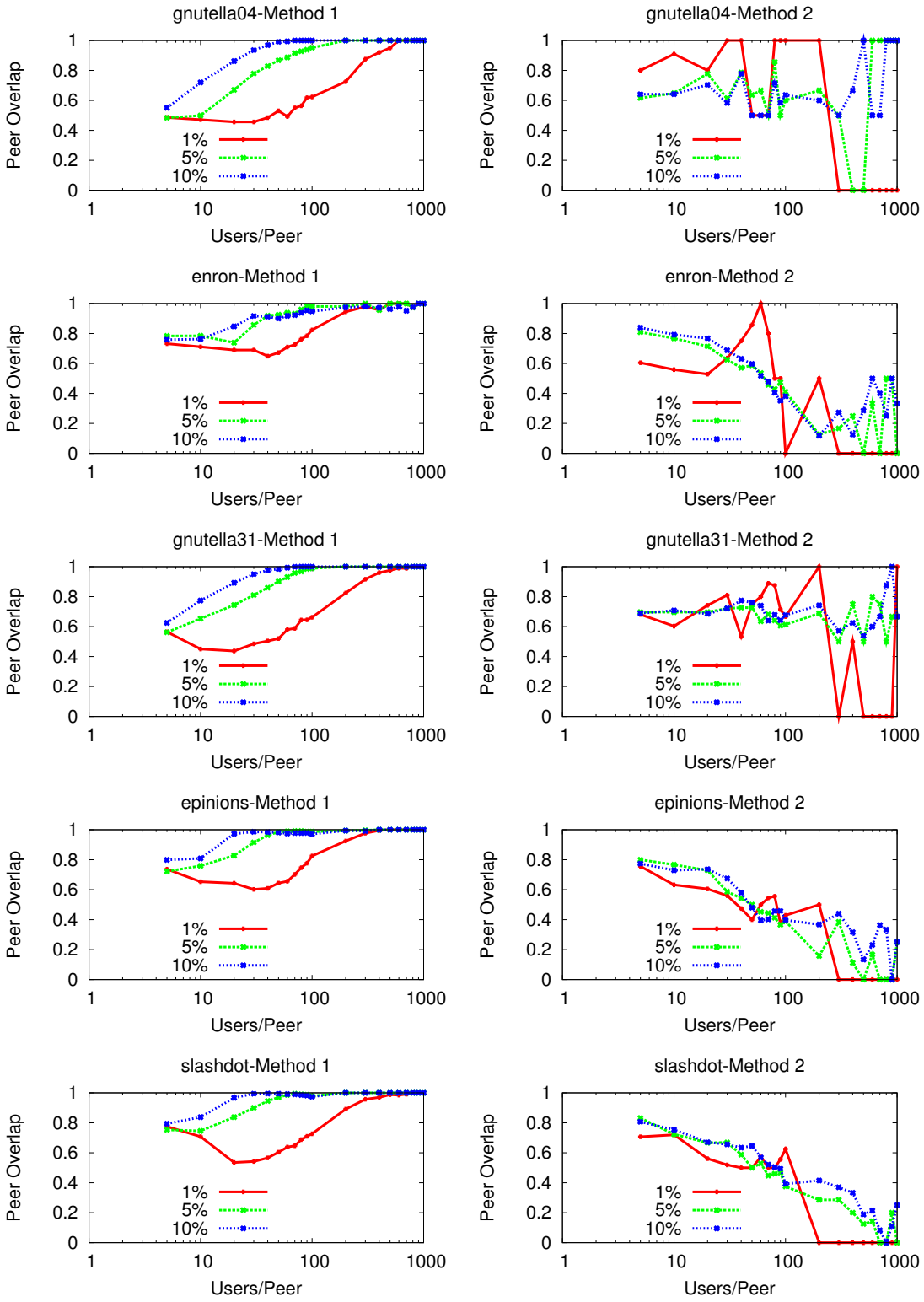

Figure 7.9: Percent overlap of peers for top $N\%$ node betweenness centrality in the networks used.

Chapter 8: Leveraging the Projection Graph in Application and System Design

Previous studies have used centrality of nodes in social graphs to improve performance of applications such as search [ALPH01] or dissemination of software patches for worm containment in social networks [ZCZ+09, NXT10]. We propose that projection graph centrality properties (either estimated as proposed earlier, or calculated exactly) can also be used to improve performance of applications such as social data search or dissemination of emergency messages. In the following sections, we focus on social data search.

We study two different search workloads (Section 8.1) and investigate techniques that use the projection graph model and its centrality properties at the application level (Section 8.2), and the overlay level (Section 8.3) for improved performance. We summarize our findings in a set of lessons (Section 8.4).

8.1 Application Workloads and Experimental Setup

In this Section we describe two applications inspired from the motivating scenarios presented earlier in Section 5.1. These applications are focused on social data search and impose two different search workloads in the P2P system. We further describe the details of the experimental setup used by the next two sections.

8.1.1 Application Workloads

The first application is *person-finder*: the relatives of a missing person in the disaster search the civilian network distributed on the P2P network. Social information is highly geo-localized in the system. The search could start from a remote node close to the last known location of the missing person. The search traverses the social graph to find time-relevant information about this person, either stored by him or by others. Many community peers could be potentially visited before any information about the person is found. A smart search technique should strive to limit the peers visited and thus the overall communication in the system, while maximizing the success rate.

The second application is *team-builder* in online gaming, a service that builds teams by matching players based on their gaming characteristics such as play statistics or level of experience. Server administrators occasionally instantiate such a service for competitions or simply for increased fun. Such a service aims to find D players from at least $\sqrt{D}$ distinct communities (for diversity in playing style) in order to form N teams with C players each (D=CxN). The service traverses the meta-gaming social graph in search for the right combination of players, potentially visiting hundreds of servers, with each server storing data of tens to hundreds of players.

These two applications represent the following search workloads: a) Starting from a random user s in the social graph, find a specific user d. b) Starting from a random user s in the social graph, find any D users from $\sqrt{D}$ different communities.

8.1.2 Experimental Setup

For these experiments we used the two largest graphs, *slashdot* and *gnutella*31. *Slashdot* has about double the graph density than that of *gnutella*31, which leads to shorter aver-

age path lengths (4.07 vs. 5.94 hops). We constructed their projection graphs considering communities of about five users (a typical family server) with maximum sizes of 358 and 97 users for *slashdot* and *gnutella*31 respectively (closer to gaming server size). This average size also offers the best estimation of peer centrality from the cumulative scores of users, while allowing the formation of large-scale peer networks ($> 10K$ peers).

We performed the *person-finder* search for a number S of different pairs of source and destination users. This number was set to 10% of the whole social graph (i.e. $S = 8216$ pairs for *slashdot* and $S = 6256$ pairs for the *gnutella*31). We performed the *team-builder* search for the same number of starting users S.

D was set to 1% of the users of each social graph, to force search queries to traverse each social graph for more than 2 social hops (i.e., to visit at least the friends of friends of a source user): on average, for $\sim$4 ($\sim$2.5) hops for *gnutella*31 (*slashdot*), with user average degree 4.7 (13.2).

We did not apply any constraints on the number of hops traveled from the source user, to study the highest possible success rate with respect to the incurred communication cost. However, we maintained the history of the previously visited users/peers and stopped the search when either the search goal had been met, or all neighboring users/peers had been visited. We measured the query success rate, the number of social graph and projection graph hops traversed, and the percentage of system peers accessed (P2P communication overhead).

8.2 Leveraging the Projection Graph at the Application Level

In the first approach, we inform the search not only with social graph topology properties [ALPH01], but also with projection graph properties which peers acquire within the system. The intuition is that a search query should be forwarded to users who being

mapped on central peers are likely to be connected to other central users mapped on the same central peers [SBAG08], and this should lead to improved search performance. In this section we investigate how the projection graph properties can inform various social search techniques to increase the success rate and reduce overhead.

8.2.1 Social Search Techniques

We investigate the following four search techniques for traversing the social graph. The first three techniques assume that during the social graph traversal, a user forwards the search query to its neighboring users mapped on peers with 1) degree centrality in the top$N$% of neighboring peers, 2) betweenness centrality in the top N% of neighboring peers, or 3) to neighboring users whose peers connect over projection graph edges with betweenness centrality in the top N% of neighboring projection graph edges. These techniques allow an application to utilize peer centrality to inform its graph traversal when specific user centrality is not available (e.g., due to privacy settings).

We compare their performance with a baseline technique (4) which even though still utilizes the same social graph topology, it does not take into account projection graph topology properties but randomly selects the same number N of neighboring users to forward the query. We also investigated the same techniques using centrality calculated in the social graph, but we exclude these results for brevity of discussion, as they show similar performance to the projection graph centrality techniques. We tested all the above techniques for N=20 and N=50.

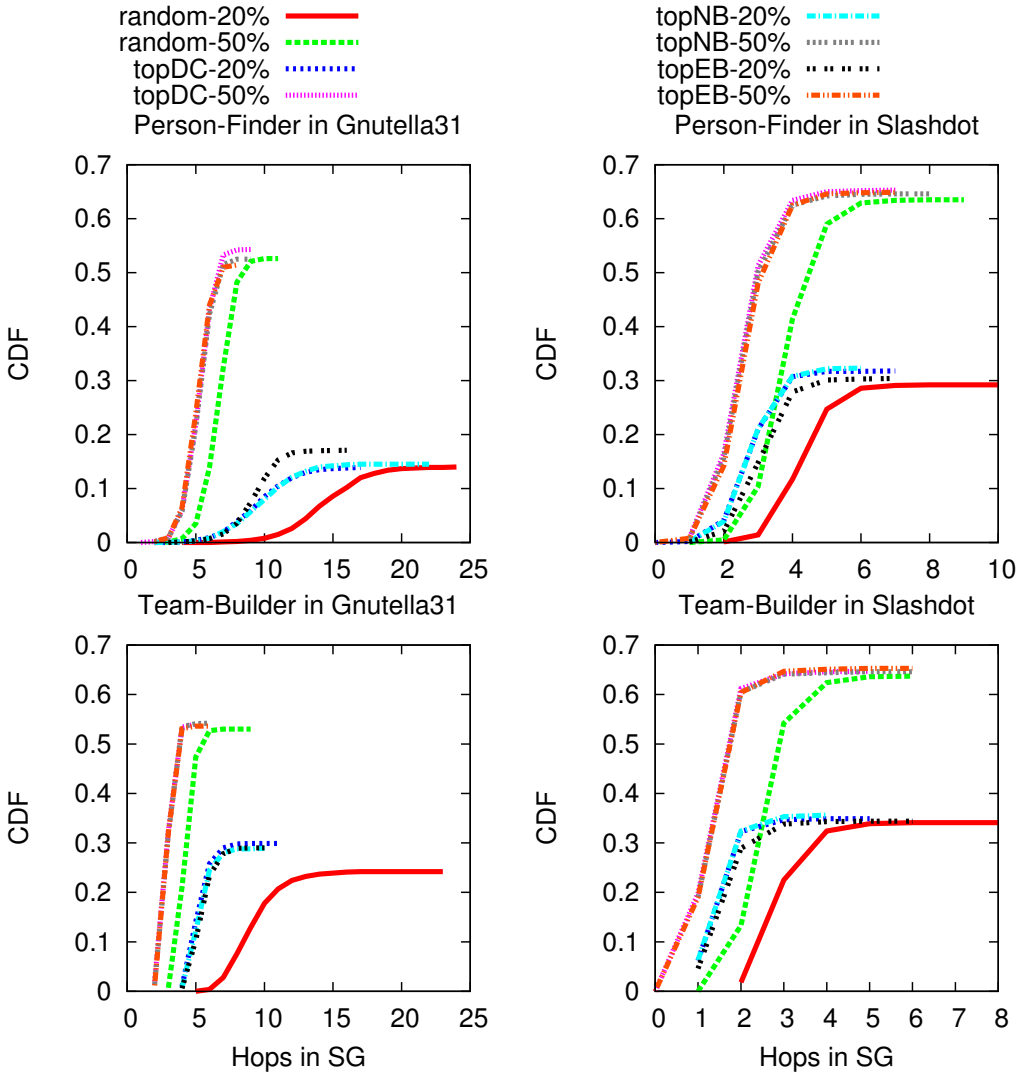

Figure 8.1: CDF of the number of social graph hops for successful queries in social graph traversals. This is the success rate for successful *person-finder* and *team-builder* queries in the social graph, for different search techniques and portion of edges used, for the networks *gnutella*31 (62561 nodes, 147878 edges) and *slashdot* (82168 nodes, 504230 edges).

8.2.2 Experimental Results

Figure 8.1 presents the success rate of the *person-finder* and *team-builder* searches as a function of the number of *SG* hops traversed for the four search techniques. For the

person-finder search over the *gnutella*31 (*slashdot*) network, all techniques converge to a maximum of about 55% (65%) of query success rate when 50% of social edges are used and about 17% (30%) when 20% of social edges are used.

For the *gnutella*31 (*slashdot*) network, we notice that more than 50% of the queries finish within 7 (3) hops when centrality techniques are used in comparison to about 9 (4.5) hops for the random technique. Thus, even though the random technique uses the same number of edges on the same graph as the centrality techniques, the random selection of which edges to follow in the search leads to longer walks on the SG and lower success rates.

Note that while traditional DHTs are better at finding "the needle in the haystack", they are impractical in this application scenario for the following reasons. First, identical names can exist within the same geographical region (and even the same family) and thus ambiguity can be introduced as to which person's social data were returned. Second, DHTs do not easily exploit the geographic locality implicit to this search type, which has to follow the social graph edges within each community and geographic location to find the correct person from the appropriate community.

In comparison to the *person-finder* search, the overall success rates reported for *team-builder* search are 5–10% higher, with the queries finishing within shorter walks. This is expected as the *team-builder* search is satisfied with *any* users discovered in the social neighborhood of the randomly selected source, given they come from $\sqrt{N}$ different communities, as opposed to finding a specific user. We notice that within 4 (2) social graph hops in *gnutella*31 (*slashdot*), more than 50% (60%) of the queries finish using centrality techniques, whereas only about 20% (15%) using the random technique.

Figure 8.2 shows the overhead as the percentage of peers accessed. We compare the random technique only with the *topN*% peer degree centrality technique as the other centrality techniques perform similarly in success rate. The *person-finder* search on both

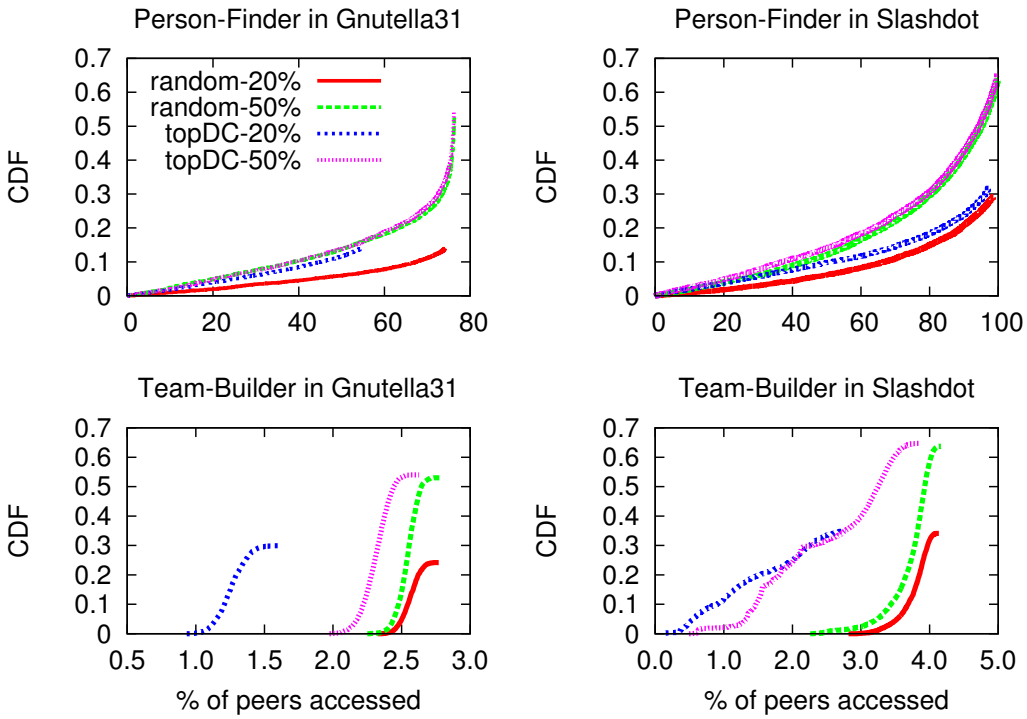

Figure 8.2: CDF of the percentage of peers accessed in the system in social graph traversals. This percentage means system overhead of successful *person-finder* and *team-builder* queries in the social graph, for different search techniques and portion of edges used, for the networks *gnutella*31 (62561 nodes, 147878 edges) and *slashdot* (82168 nodes, 504230 edges).

networks has a similar overhead in both types of techniques, especially when using 50% of edges. However, the *team-builder* search with the centrality technique has 0.25–2 times less overhead than the random technique.

As mentioned in the beginning, in addition to the experiments using projection graph properties, we tested techniques that used the centrality properties in the social graph (not shown here for brevity). These experiments showed that the centrality techniques that use social graph centrality scores perform similarly with the centrality techniques that use projection graph centrality scores. This observation was true when using degree centrality or node betweenness centrality.

This can be attributed to the high correlation between user and peer scores for the same centrality measure: correlation 0.9–0.95 (0.8–0.9) between degree (node betweenness) centrality of users and peers, as well as between different centrality measures: correlation 0.85–0.91 (0.71–0.85) between degree and node betweenness centrality of users (peers). Furthermore, global metrics that require knowledge of the whole social or projection graph, such as node betweenness, do not add much gain in the search performance, so an application can effectively use local information instead, such as degree centrality.

8.3 Leveraging the Projection Graph at the P2P Overlay Level

A P2P system could be organized into either an unstructured overlay (e.g., Gnutella) or a structured overlay (e.g., a DHT). Leveraging social graph knowledge has been applied to both structured [MGGM04] and unstructured [PCT04] P2P overlays. In this section we investigate the benefits of informing routing decisions in the P2P network with *PG*-specific information. We focus on unstructured overlays, leaving the structured overlays for future work.

8.3.1 *PG*-Based Unstructured P2P Overlays

By definition, a projection graph is the accurate representation of the social graph mapped on the P2P system. We propose an unstructured P2P overlay that exactly mimics the projection graph: the routing tables in the P2P network consist of (a subset of) the projection graph edges that connect different peers. This overlay reflects well the social relationships between users (thus best supporting socially-aware applications), and, implicitly encapsulates geographical (and network) locality and clustering, since social relationships are usually geographically close [SNLM11].

However, the power-law nature of the node degree in the projection graph translates into high-degree peers maintaining unrealistically many connections. Therefore, we propose that the projection graph edges (E_P) are considered as *potential* communication connections between peers in the overlay, but only some of them are implemented into *active* communication connections (E_A), i.e., $E_A \subseteq E_P$.

We investigate the same four techniques but this time in overlay routing. In the first three techniques, a peer forwards the search query to its neighboring peers with 1) degree centrality in the *topN*% of the neighboring peers (set D), 2) betweenness centrality in the *topN*% of the neighboring peers (set B), or 3) that are connected over projection graph edges with betweenness centrality in the *topN*% of the neighboring projection graph edges (set E). The fourth technique is for baseline comparison: a peer forwards the query in the projection graph topology to the same number N of randomly selected neighboring peers (set R, $|R|=|D|$).

The difference from the application-level techniques (Section 8.2.1) is that a search query traverses the projection graph instead of the social graph. Thus, instead of forwarding a message along social graph edges and potentially bouncing multiple times between the same peers, the message is forwarded along the projection graph edges and thus reducing redundant communication.

To apply these techniques, the following assumptions are made: First, a query can access all user data stored on a peer. Second, as in all P2P systems, peers regularly update information regarding peers in their routing table, such as availability, but also *PG*-based centrality metrics. Third, peers rank their neighbors based on the projection graph metrics, and depending on the heuristic applied, they select the *topN*% subset as their active connections.

Therefore, depending on the search technique used (t=1,...,4), an active connection between two peers P_i and P_j will be included in the set of active connections of P_i (set $E_A^{P_i}$)

and can be defined as follows:

$$(P_i, P_j) \in E_A^{P_i} \text{ iff } \exists P_i \in V_P, \exists P_j \in V_P \text{ s.t.}$$

$$(P_i, P_j) \in E_P \text{ and } \begin{cases} P_j \in D & \text{if } t = 1 \\ P_j \in B & \text{if } t = 2 \\ (P_i, P_j) \in E & \text{if } t = 3 \\ P_j \in R & \text{if } t = 4 \end{cases} \tag{8.1}$$

Consequently, the total set of active connections in the P2P network E_A is the union of the sets $E_A^{P_i}$ for all peers:

$$E_A = \bigcup_{\forall P_i \in V_P} E_A^{P_i} \tag{8.2}$$

8.3.2 Experimental Results

We tested the four techniques by varying $N\%$, the portion of edges used. Since the search query has access to all the users' data stored on a particular peer, we expected the search to finish with higher success rate and in shorter walks than when the search traversed the social graph edges (Section 8.2).

By varying the portion of edges used from 1% to 50%, our experiments (not shown here for brevity) revealed that using ∼20% of available projection graph edges leads to almost maximum success rate for both *person-finder* and *team-builder* searches; above 20% there is mostly increase in the message overhead with minor gains in success rate. Using below 10% of projection graph edges leads to low search performance for both search types regardless of the technique used, but with the random technique performing the poorest. Next, we compare the techniques using 10% and 20% of PG edges.

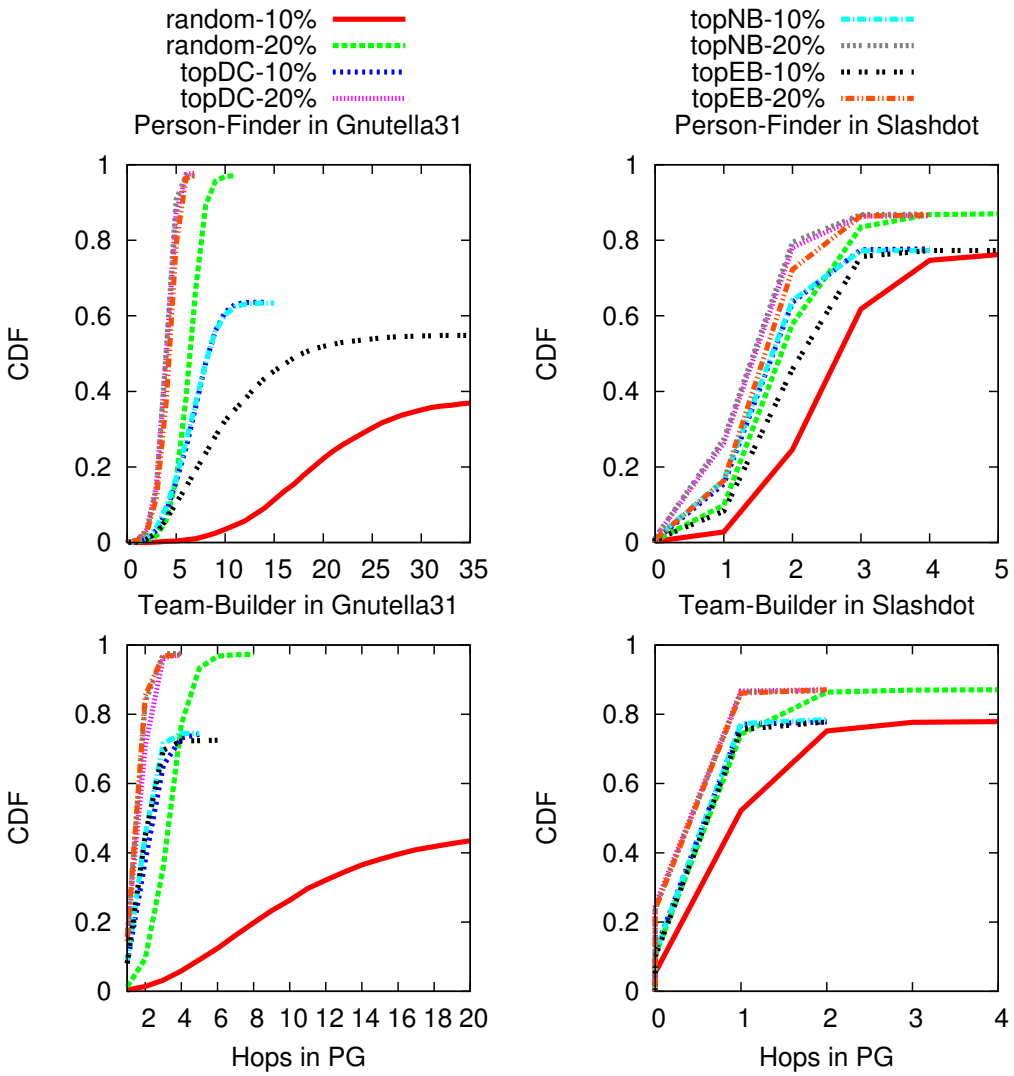

Figure 8.3: CDF of the number of projection graph hops for successful queries in projection graph traversals. This is the success rate for successful *person-finder* and *team-builder* queries in the social graph, for different search techniques and portion of edges used, for the networks *gnutella*31 (62561 nodes, 147878 edges) and *slashdot* (82168 nodes, 504230 edges).

Figure 8.3 presents the success rate of the *person-finder* and *team-builder* searches as a function of the number of projection graph hops. For the *person-finder* search and the *gnutella*31 (*slashdot*) network, all techniques converge to a maximum of about 98%

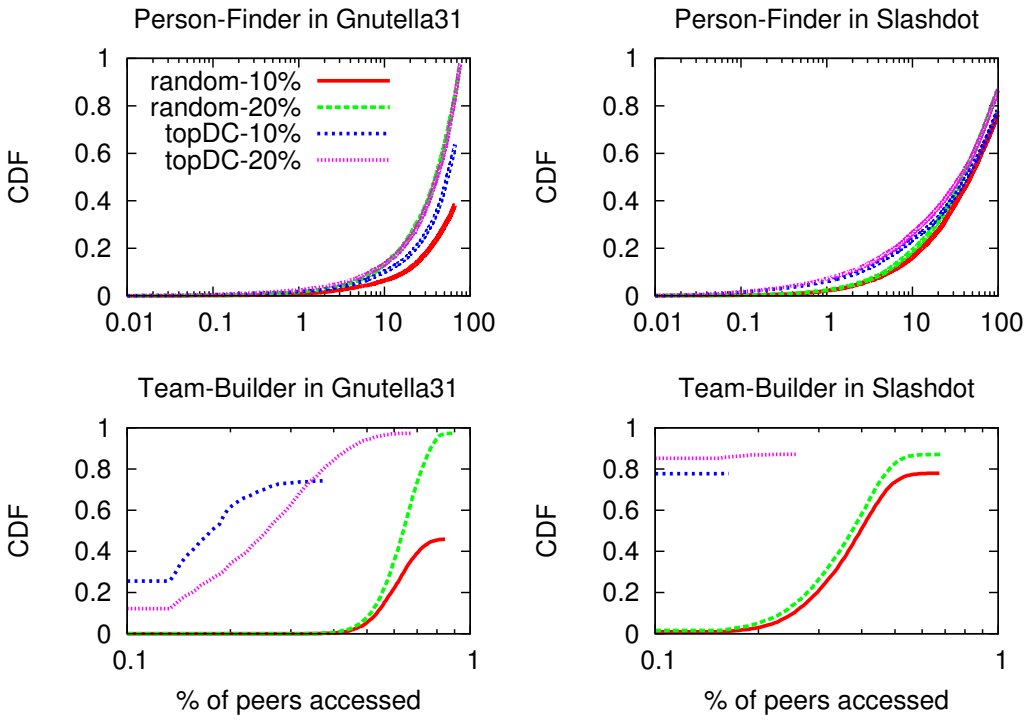

Figure 8.4: CDF of the percentage of peers accessed in the system in projection graph traversals. This percentage means system overhead of successful *person-finder* and *team-builder* queries in the projection graph, for different search techniques and portion of edges used, for the networks *gnutella*31 (62561 nodes, 147878 edges) and *slashdot* (82168 nodes, 504230 edges). Note the x-axis in log scale.

(87%) of query success rate when 20% of projection graph edges are used and about 64% (78%) when 10% of projection graph edges are used.

A centrality technique with 20% projection graph edges has 20% more success within 2 P2P hops than using the random technique. This difference is amplified for the *gnutella*31 network, where within 5 hops the centrality techniques achieve over 60% more success than the random technique. For the *team-builder* search, similar maximum success rates with the *person-finder* search are reported for both networks, but with the queries finishing in shorter walks by 3 (2) fewer P2P hops for the *gnutella*31 (*slashdot*) network.

The gain in success rate and with fewer P2P hops is reflected on the system overhead presented in Figure 8.4 as the percentage of peers accessed. We compare the random technique only with the *topN*% peer degree centrality technique, as the other centrality techniques perform similarly in success rate. Overall, the centrality technique leads to similar or lower system overhead than the random technique. The *person-finder* search has a similar overhead in both techniques, especially when 20% of edges are used: up to 76% (99.4%) of peers were accessed in *gnutella*31 (*slashdot*) to reach maximum possible success rate.

For the *team-builder* search, the random technique has about 3–4 times more overhead than the centrality technique, due to longer walks in the projection graph (as seen in Figure 8.3). Thus, the technique needs to access a larger portion of peers to satisfy the *team-builder* queries.

8.4 Applicability of Results

The previous experimental results on social search from both the application layer and the P2P overlay layer allow us to formulate the following lessons.

Lesson 1: *The centrality techniques lead to higher success rates within fewer hops (in the social or projection graph) than the random technique.* In particular, we observe that even though the random technique is also socially-aware as it utilizes the same social and projection graph topology construction as the centrality techniques (but randomly selects to which users or peers to send the query), it still requires about 1–3 more hops to reach the same success rate as the centrality techniques, thus imposing higher overhead in the system.

Lesson 2: *Search techniques that use social graph (user) or projection graph (peer) centralities perform similarly.* This means that an application could select which of the cen-

trality techniques to use based on the available centrality information for each user or peer. If it cannot access the individual score of Alice in the social graph or her full data to calculate it (e.g., due to privacy settings), but can access her peer's score (an aggregate metric for a user group), it can achieve the same performance by routing queries through the social graph edges using the projection graph peer centralities.

Lesson 3: *Search techniques that use local or global information perform similarly.* Search techniques that use global centralities calculated over the whole graph (i.e., node and edge betweenness centrality) perform similarly to the ones calculated using local information (i.e., degree centrality). Therefore, an application can utilize the degree centrality of users or peers to inform the forwarding decision of the search query.

Lesson 4: *It is feasible and more efficient to construct PG-based P2P overlays.* A P2P system can leverage a centrality technique that uses local information such as peer degree centrality to construct the set of active connections E_A used by peers during a search. Furthermore, a small set of active connections $E_A^{P_i}$ per peer is enough to ensure high performance and low communication overhead.

This fraction of edges would mean for the most connected peer of *gnutella*31 and *slashdot* a maximum of 80 and 392 active connections respectively, which is well below the maximum connections of deployed unstructured overlays (Gnutella V0.6 had peers with more than 500 connections [RSR06]).

Lesson 5: *Traversing the projection graph could bring significant improvements in search and reduced overhead.* In comparison to traversing the social graph edges, leveraging the projection graph topology provides access to social information of more users and thus, on average, increases the success rate by 10%–25%, reduces the walk length by 1–2 hops and decreases the percentage of peers accessed by 40% for the $person\text{-}finder$ search and by 2.5% for the *team-builder* search. Thus, socially-aware applications and services could

be developed to take full advantage of the available information for enhanced application search and overlay performance.

Chapter 9: Related Work

In this chapter we present a literature review of previous studies related to this dissertation. First we cover studies related to the Prometheus system [KFA+10b, KFA+10a], and its various design features, and then we cover studies related to the projection graph topologies [KI11] and peer centrality in such graphs.

9.1 Socially-Aware Applications and Services

Socially-aware applications and services have so far leveraged out-of-band social relationships for objectives such as recommending email recipients or filtering spam based on previous email activity [KRS+06], filtering restaurant recommendations based on reviews by friends (e.g., Yelp [Yel12]), improving security [YKGF06, YGKX08], inferring trust [MRG+05], providing incentives for resource sharing [TCL08, LD06], or building overlays [PCT04] 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 [GMD06], to favor socially-connected users in a BitTorrent swarm [PGW+08], and to reduce unwanted communication [MPDG08]. Social information has also been leveraged in conjunction with location and collocation data to provide novel mobile applications such as Loopt [Loo12], Foursquare [Fou12] and Google's Latitude [Goo12a].

In all of these cases, social knowledge has been mined in the context of a single application or from a single source of information. Our approach on the Prometheus system differs from these studies in four important ways.

First, we collect information from multiple sources. Two systems that aggregate social information from multiple sources are SONAR [GJS+08] and SocialFusion [BGX+10]. 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. Similar to our approach, SocialFusion proposes an integration of diverse mobile, social and sensor input streams, but focuses on the class of context-aware social applications, whereas our work enables a larger variety of social applications.

Second, we extract knowledge from the social graph by accessing larger portions of the network, not only a user's direct neighborhood. Somewhat similarly, RE [GKF+06] uses 2–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 [RF08].

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 [KN09]. Prometheus differs in its generality of the graph—its edges can represent interactions collected from different sources.

9.2 Social Data Management

There have been various system architectures proposed in the past for managing social information of users. Centralized architectures have been primarily used from companies operating online social networks for regular users (e.g., Facebook, Google, Twitter [Twi12a], etc), academics and researchers (e.g., Academia.edu [Aca12]), professionals (e.g., LinkedIn [Lin12]), etc. Our literature review focuses on distributed architectures including mobile systems and peer-to-peer networks.

9.2.1 Mobile Systems for Social Data Management

System architectures to store and manage social information over *mobile devices* have been proposed in systems such as [MMC09, POL$^+$09, SRA10, TPI11].

In [MMC09] a middleware service is described for social networking via mobile devices. Devices are logically divided to clients, i.e., nodes with social-based information regarding their owners' preferences and who request recommendations from brokers, i.e., nodes who are responsible for gather social information from multiple clients, propagating social links and providing recommendations to clients.

Mobiclique [POL$^+$09] is a mobile social networking middleware that allows mobile users to tap on existing online social networks such as Facebook and extend their interactions in real life through opportunistic encounters with other users. Mobiclique devices bootstrap from their owners' OSN profiles and alert them when they share some pre-defined relationship like friendship or interests with nearby users.

A distributed tuple space system is presented in [SRA10] that can support services usually provided by online social networks such as calendars, video and audio communication

and instant messaging. Data are stored in, or retrieved from the system in the form of tuples (key/value pairs). In addition it can support these functionalities in offline mode using multi-hop ad hoc networks.

Yarta [TPI11] is a middleware for supporting mobile social ecosystems, which are sets of rich interactions between users, over different types of activities, especially when within physical proximity. It allows exchange of social information between users and applications and enforces access control to users' data. It also implements a set of components for data extraction from social sensors, storage of the data within the users' devices, and retrieval from mobile applications, whose code can be generated using automatic tools.

All these approaches explore the possibility of storing and sharing user social information on systems of mobile devices. However, 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 support up-to-date inferences; and computational resources, and more importantly energy, are likely to be scarce, as pointed out by many of the above studies. Also, mobile phones might not be allowed to store social information of users other than their owner, thus incurring excessive network overhead to fulfill a multi-hop query in the social graph.

9.2.2 Peer-to-Peer Systems for Social Data Management

In [BD09] and [DBV+10] a set of research challenges are identified when decentralizing core functionalities of social networks and how a peer-to-peer infrastructure can be used to realize a decentralized online social network. Security, user control of data, storage, topology, search, and update management are some of the topics discussed within this realization.

Prometheus leverages a P2P architecture within the social hourglass infrastructure [IBK12] to provide user-controlled social data management and social inferences for services and applications. P2P management of social information has also been addressed in systems such as PeerSoN [BSVD09], Vis-à-Vis [SLCC08, SVCC09, SLC+11], Safebook [CMS09, CMS10], LotusNet [AR12] and LifeSocial.KOM [GGS+11].

PeerSoN [BSVD09] is a two-tier system architecture: the first tier is used to lookup for metadata of users, current location/IP, notifications, etc; the second tier is used for the actual contact between peers and users and the exchange of data (files, profiles, chat, etc). The lookup service is implemented with a DHT. The second tier is allows for opportunistic and delay-tolerant networking. Users can exchange data either through the DHT storage or directly between their devices. Prometheus differs from this approach in two significant ways: first, by enabling users to select trusted peers which are independent from the users' devices, to store and exchange social data, and second, by providing social inference functionalities to applications and services.

Safebook [CMS09, CMS10] is a decentralized OSN that aims to solve privacy issues in current centralized OSNs by using a 3-component architecture: a trusted identification service to provide each peer and user unambiguous identifiers, a P2P substrate for lookup of data, and matryoshkas, concentric rings of peers around each member's peer that provide trusted data storage, profile data retrieval and obscure communication through indirection. Profiles of users in Safebook are replicated in the innermost layer of a user's matryoshka to increase availability of the data. Similarly, Prometheus users employ multiple trusted peers. Also, Prometheus similarly forwards messages between peers based on social relations and inference requests, while restricting a large scale view of the graph through P2P decentralization. Prometheus differs dramatically, however, in the exposure of the graph as a first class data object through inference functionality.

In [SLCC08, SVCC09], three schemes for decentralized OSNs were presented, where users store their personal data in their own machines or Virtual Individual Servers (VISs). The VISs use peer-to-peer overlay networks, one per social group, to allow their owners to share information with other users. The first scheme deploys the VISs on a virtualized utility computing infrastructure in the cloud, the second scheme on desktop machines augmented with socially-informed data replication, and the third again on desktop machines with standby virtual machines in the cloud when the primary VIS of a user becomes unavailable. In later work [SLC+11] the authors focused on providing location-based networking services using a cloud infrastructure than a P2P network, since trust of the compute utilities is warranted given the companies' business models and terms of use with their clients. While similar to the trusted peer concept in Prometheus, VISs are hosted by a centralized cloud computing provider to tackle peer churn, while Prometheus provides a truly decentralized infrastructure—making use of social incentives to reduce churn on user-supplied peers.

LotusNet [AR12] is a DHT-based online social network system that binds user identities with their overlay nodes and published resources for increased security. User data such as social information, content, etc, are stored encrypted on the *Likir* DHT [AMRS08]. Users control access to their data by issuing signed grants to other users applied during retrieval of data. Services such as event notifications, folksonomic content search and reputation management are demonstrated.

In LifeSocial.KOM [GGS+11], user information is stored in the form of distributed linked lists in a DHT and is accessible from various plugin-based applications, while enforcing symmetric PKI to ensure user-controlled privacy and access. However, the data of a user are isolated from other users and peers access them individually, incurring significant system overhead.

Generally, in all these studies (PeerSoN, Safebook, LotusNet, LifeSocial.KOM), the social graph is fragmented into 1-hop neighborhoods, one for each user, and distributed across all peers, with potentially multiple fragments stored on the same peer that cannot be combined. Also, there is no peer functionality that utilizes the combination of the accessible units for simple social inference functions such as neighborhood search. In contrast, in Prometheus, a peer can locally mine the collection of social data entrusted to it by a group of (possibly socially-connected) users.

Commercial efforts such as [Ent12, Fre12a, Dia12] implement distributed social networking services and allow users to share content in a decentralized fashion. Our work with the Prometheus service significantly differs from the previously noted approaches (commercial and academic) 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 various socially-aware applications. This approach shares ideas with the MobiSoc middleware [GKBB09]; however, the MobiSoc middleware is logically centralized leading to "big brother" concerns similar to existing OSNs.

9.3 Privacy and Security in Decentralized OSNs

In Safebook [CMS09, CMS10] the concentric matroysha layers of nodes around the data of each user offer privacy and allow users to control access to their data via three levels: public, protected or private. However, there is an inherent tradeoff between privacy due to the number of layers around user data, and data availability and delays for data retrieval due to the recursive nature of the Safebook forward protocols. Instead, Prometheus enables users to select multiple trusted peers to increase data availability and define fine-grained access control policies applied at each trusted peer to provide specific data to specific applications or users. Using the ACPs, Prometheus users can also require inference requests to be initiated or forwarded through particular users and/or peers in the 1-hop

neighborhood of the user owning the data, therefore enabling stronger data protection than the matroyshka layers, and without the overhead expected in Safebook [CMS10].

Persona [BBS+09] combines attribute-based encryption with traditional PKI cryptography and supports flexible and fine-grained access control for user data. In the future, Prometheus could leverage this system to provide more flexible access control policies while preserving inference functionality.

Lockr [TSGW09] is an access control system that can be used either in centralized or decentralized OSNs, by decoupling the content to be shared between users, from the list of users participating in the sharing. This is achieved by allowing users-data owners to distribute signed social attestations of their relationships to their contacts that will want to access the particular content. Instead, in Prometheus, users that want to allow access to particular data need to define specific ACPs for that data only once, and their trusted peers will apply them multiple times thereafter every time there is an incoming inference request. Similarly to Lockr, the ACPs can include social relationships that data owners have with the data requesters, which can be either direct or indirect social contacts.

In [UPvS11] the authors provide a detailed review of system-level attacks that a DHT-based system such as Prometheus could be the target of, and ways to defend against them. We briefly discuss Sybil attacks [Dou02] which are easy to stage and more effective in a P2P-based social data management system.

In a Sybil attack, a malicious user introduces to the system multiple nodes he directly controls, with different identity for each node, so that he can disrupt system protocols such as routing, storage, etc, by artificially creating majority of colluding nodes in the P2P system. Prometheus could defend against such attacks by employing Sybil-resistant DHT solutions such as [DLLKA05] or [LLK10]. Also, defenses using existing security techniques (e.g., [YKGF06, YGKX08, TLSC11]) can be used to perform decentralized

Sybil-resilient admission control of new users in the system, or to control formation of new social edges between users and how their social data are stored on peers.

9.4 Application Programmable Interfaces for Social Information

APIs for accessing social information have been offered by several OSN companies. Facebook allows 3rd-party application developers and websites to access the social information of millions of users through the Open Graph API [Fac12a].

Similarly, OpenSocial [Goo12b] is a public specification used by many OSNs. It defines a component hosting environment (container) and a set of common APIs for web-based applications to access social information stored on any of the supported social networks: Hi5 [Hi512], MySpace [MyS12], Orkut [Ork12], Friendster [Fri12], Yahoo! [Yah12], etc.

Twitter offers two APIs. The Rest API [Twi12b] offers information about users and their tweets, messages between users, friends and followers of users, favorites tweets, geo-related information, arising trends, etc. The Streaming API [Twi12c] offers near-realtime access to various topics of tweets as being produced by users and websites.

It has been shown that social networks constructed from online gaming activities between players can exhibit social properties (e.g., Pardus [ST10] and Steam [BSL+11a, BSL+11b, BSK+12]) and therefore collecting and analyzing social information from such networks can be of importance to application developers. Valve, a company that creates and distributes online games to millions of players, provides a web API [Ste12] enabling developers to access gaming and social data from profiles of gamers interacting within Steam (its in-house online gaming social network) and its supported online games.

Such APIs do not provide direct complex inferences such as "friends of friends" or "social-strength" as Prometheus does, but instead an application must explicitly crawl the graph and infer such information.

9.5 Projection Graphs in Existing Peer-to-Peer Systems

In the previously mentioned P2P systems (PeerSoN [BSVD09], Safebook [CMS10], Vis-à-Vis [SLCC08, SVCC09, SLC+11], LotusNet [AR12] and LifeSocial.KOM [GGS+11]) as well as Prometheus, the decentralization of the social graph of users onto multiple peers leads to the emergence of a *projection graph* between peers, as proposed in this work, regardless of the way peers are organized in the P2P architecture (e.g., in a structured or unstructured overlay).

In such systems, a peer can calculate the cumulative centrality score of users in the social graph mapped on the peer, and infer the peer position and thus importance in the socially-informed topology. Therefore, the projection graph model can be applied for studying and improving the performance of a socially-informed application using the social graph distributed in such systems.

There are other systems that utilize the projection graph topology by directly reflecting the social graph topology in the P2P overlay organization. Turtle [PCT04] uses trust relationships between users to build overlays for private communication and preserve the anonymity of its users. F2F [LD06] uses social incentives to find reliable storage nodes in a P2P storage system. Sprout [MGGM04] enhances the routing tables of a Chord DHT with additional trusted social links of online friends, to improve query results and reduce delays. Tribler [PGW+08] allows users that are socially connected and participate in the same BitTorrent swarms to favor each other in content discovery, recommendation and downloading of files.

In other studies such as [YZLT09], peers are organized into social P2P networks based on similar preferences, interests or knowledge of their users, to improve search by utilizing peers trusted or relevant to the search. Similarly, in [LCTC07] a social-based overlay for unstructured P2P networks is outlined, that enables peers to find and establish ties with other peers if their owners have common interest in specific types of content, thus improving search and reducing overlay construction overhead. In [WS08], P2P social networks self-organize based on the concept of distributed neuron-like agents and search stimulus between peers, to facilitate improved resource sharing and search. In such systems, the peers form edges over similar preferences of their owners or search requests, reflecting the P2P edges in the projection graph model. Thus, they implicitly use this model to organize the peers into P2P social networks.

9.6 Peer Centrality

Relevant to our work on the centrality of peers in the projection graph, is the notion of the *group reduced graph*, as sketched in [EB99], where groups of users are replaced by a single "super" vertex whose neighborhood is the union of the neighborhoods of all group members. This technique allows for easier construction of group centrality measures, but a basic challenge, as stated in [EB99], is to justify the removal of internal social links in a group. However, in our instance of the reduced model, this is not a problem: members of a group can store their data on the same peer which has complete knowledge of the social subgraph formed by the union of individual users' social data.

Studies such as [EB99] and [KCB09] analytically discuss the betweenness centrality of a group of nodes by computing shortest paths between nodes outside the group, that pass through at least one node in the group [EB99] or all the nodes in the group [KCB09]. Similarly, we study the betweenness centrality of peers representing groups of users. However, we assume that *all* users are mapped on peers (groups) and compute the peer be-

tweenness centrality based on shortest paths between users mapped on different peers only.

Chapter 10: Lessons on Designing Socially-Aware Distributed Systems

Our work on the design and experimental evaluation of Prometheus, as well as our experimental analysis of projection graphs, enables us to extract several lessons that are applicable to previous studies on social-based P2P systems and can provide guidelines for the design of future socially-aware distributed systems and applications.

10.1 Socially-Aware Decentralization of the Social Graph

Enabling users to select where to store their social information can lead to a socially-aware distribution of users' data in the system. Therefore, groups of socially-connected users could store their data on the same peers, and this could lead to significant system improvements:

- Reduce end-to-end response time for applications mining the social graph.
- Reduce message overhead between peers during inference execution and social graph traversal.
- Increase success rate in social graph search.
- Reduce the average opportunity of peers to influence inference requests executed in the system, and thus their ability to manipulate requests.

These improvements allow the system to fulfill faster any application inference requests and increase resilience to attacks, either staged by independent peers or by colluding peers.

10.2 Peer Organization and Centrality Estimation

We have shown that a projection graph with social properties can emerge in a system that distributes the data of users in a socially-aware manner (e.g., groups of socially-connected users, or communities of users, store their data on the same peer). In our projection graph experiments, increasing the average community size from 5 to 1000 users per peer inevitably decreased the number of peers in the system (from thousands to tens). This parameter allowed us to study how the degree of social data decentralization affects the network properties, and in particular the centrality of the nodes in the system.

Socially meaningful communities of 50–150 *users lead to optimal peer organization in terms of degree and node betweenness centrality.*

Our study of projection graphs led, via a different methodology, to a previous result known as the Dunbar number. Dunbar [Dun98] predicted that on average individuals can manage a meaningful social circle of about 150 friends, no matter how sociable they are. Also, people tend to self-organize in groups of around 150; above this number, social cohesion begins to deteriorate as groups become larger. In [LLDM08] the best communities with respect to conductance were found to be relatively small with sizes only up to about 100 users, thus agreeing with Dunbar's number. Above this size, the quality of communities is questionable as they start blending more with the core of the graph.

We found that, for a particular range of the average number of users per peer, the projection graphs follow closely the properties of the social graph distributed on the P2P network. Independently of the network tested, within a range of about 50–150 users per

community, the organization of the projection graph topology reaches an optimal point with respect to average degree or node betweenness centrality. This range agrees with Dunbar's number and leads us to believe that there is a high correlation between the way people organize within the social graph into social groups or communities and how the properties of such social organization can be embedded in the emerging projection graph topology.

Smaller social communities can allow better estimation of peer centrality which can be used to improve application and system performance.

Large-scale systems such as mobile phone or P2P networks decentralize the users' social data on thousands of nodes and allow each device to access social data of a small set of users. The smaller this set, the higher the association is between the users' centrality in the social graph with their device's centrality in the projection graph. Therefore, the projection graph inherits social properties from the social graph stored in the system. Since peer centrality in the projection graph can be estimated using the cumulative centrality scores of its users in the social graph, an application can use either centrality score (user or peer) to effectively route search queries.

Furthermore, an application or system overlay can identify with high accuracy the peers with top degree or betweenness centrality, by knowing the *top*-5% degree or betweenness centrality users in the social graph. Since peer centrality can be used to improve application and system performance (Chapter 8), peers could enforce upper bounds on the amount of user data to store (e.g., up to 100 users), to maintain high correlation between their centrality and the users' centrality.

However, storing more user data per peer decreases end-to-end response time for an application traversing the social graph. In fact, considering centralized company systems with a few hundred machines, each node can access social data of thousands of users. By distributing the social data on centralized machines in a socially-aware manner (e.g., as

in [PES+10]), our experiments revealed that the degree and node betweenness centrality of peers should be similar, not highly correlated with the users' centrality, and that all peers should have an equal opportunity to be queried for social data. Thus, there are no real benefits in using the centrality of peers in such centralized setups.

Therefore, trade-offs must be considered between performance improvements and the ability to use peer centrality in application or system design when decentralizing user social data on multiple machines and with a variable community size.

10.3 Leveraging Peer Centrality in the Application Design

Central peers can be used for performance improvements in social search.

Search techniques that use social or projection graph centralities perform similarly, and always with higher success rates within fewer hops (in social or projection graph) than random-based search techniques. Furthermore, using degree centrality (local information) has the same benefits for a search technique as using node betweenness centrality (global information).

Therefore, if an application was not granted access to the individual centrality scores of users in the social graph, e.g., due to user privacy settings, but only to the peer scores in the projection graph, it can still route search queries through the social or projection graph, using the peer degree centralities. These results are applicable to previous works such as [GMD06, YZLT09, LCTC07], where social search can be informed using an estimation of the peer degree centrality in the projection graph.

High betweenness peers can be used for application monitoring and trace collection.

These peers are situated on many shortest paths between other peers (and their users) and a high portion of the workload from social applications will pass through them. There-

fore, social applications can collect usage statistics or better provision resources on these peers for improved performance. This can be applied in systems that manage social information of users and can expose it to user applications and services (e.g., Prometheus).

Central P2P edges can be used for data caching and dissemination.

P2P applications can make use of high betweenness edges to 1) monitor information flow between different parts of the network, 2) place caches of data on the edges' end peers for faster dissemination and 3) create caches of search results and indexes of data location for speedup of access and search. This result could be used by systems such as Tribler [PGW+08] to advance its mechanisms for caching metadata of user activity, content location and data placement for downloads between several communities of users.

10.4 Leveraging Peer Centrality in the System Design

Central peers and P2P edges can be used to construct trusted, search-efficient and socially-aware P2P overlays.

There are benefits in building an unstructured P2P overlay by leveraging the projection graph topology and selecting P2P overlay links using centrality metrics. Our experiments on social search showed that overlay communication overhead can be reduced if peers construct their routing tables using projection graph edges to neighboring central peers. However, such P2P network paths can be used frequently from any type of application traversing the social graph and not only from social search. Thus, these paths should be explicitly defined and used in the construction of the P2P overlay.

This idea could be embedded in P2P systems such as Prometheus, Turtle [PCT04] and Sprout [MGGM04], that already implement a socially-informed design, but instead of using *single* social edges between users, they could exploit high weight projection graph

edges which represent *multiple* social edges between groups of users, and indicate stronger social bonds and potential trust between users and their peers. Therefore, these peer paths can be assumed more trusted and lead to more secure discovery of new peers for data hosting, within reduced network hops.

Peers can employ methods for bundling data shared with other peers over these paths, thus reducing network communication. Moreover, such projection graph edges could represent social incentives between multiple users for data sharing among neighboring communities and their peers. Consequently, potential increase of the communication between these peers when serving application workload, or for system maintenance due to peer churn, can be tackled with data replication to neighboring peers in the projection graph, for better data availability and load distribution.

Central peers' data can be mirrored on other peers to avoid overloading and unavailability due to peer churn.

The performance of a P2P application is inherently depended on its tolerance to churn. In the presence of high peer churn, users could change their storage peers for better data availability. Since social networks are less dynamic than P2P networks, the user social connections are typically stable. Thus, peers can estimate accurately their importance (via centrality) in the topology, based only on locally stored information, i.e., the centrality of their users, which, given the slow dynamic of social networks, can be computed infrequently.

Central users are connected to each other more directly than average [SBAG08], which means they are more likely to be part of the same community. Consequently, they could be mapped on the same peer, as a direct consequence of the socially-aware distribution of the social graph on peers. This, in effect, increases significantly the peer's centrality leading to the formation of "hot-spot" peers.

The system could infer where these hot-spot peers will appear just based on the centrality metrics of the users mapped on those peers, with no need to analyze the dynamic P2P overlay or the projection graph. Identifying such hot-spot peers can be used for monitoring much of the socially-routed peer-to-peer traffic; for placing caches or replicas; and for avoiding bottlenecks by remapping high betweenness users onto better provisioned peers.

Central peers can be used to tackle system's vulnerability to attacks.

These high centrality peers control much of the information flowing through the P2P topology and could, on one hand, be more vulnerable, and on the other hand, more effective if they participate in a malicious attack. Such peers can be targeted for quarantine in the early stages of a malware outburst or used to disseminate faster and more efficiently security software patches and emergency announcements.

Predicting social edge formation can improve data access latency via proactive caching.

Using intuition from sociological studies, systems such as Prometheus could predict the creation of social edges between users, by monitoring the triadic closures between them and identifying which ones violate the *forbidden triad rule* [Gra73]. This rule refers to the situation where two individuals, not socially connected with each other but with a strong social connection with another mutual individual, will likely form a social connection with each other in the future. This observation could enable the system to anticipate access of the particular users' social data, and thus perform proactive caching on central or neighboring peers.

Chapter 11: Conclusions and Future Work

This dissertation presented Prometheus, a P2P service that provides decentralized, user-controlled, social data management. Its directed, weighted, labeled and multi-edged social graph offers a fine-grained representation of the users' social state. It enables novel socially-aware applications to mine this rich social graph via non-trivial social inference functions, while enforcing user-defined access control policies.

We built and evaluated Prometheus using a large-scale distributed testbed and realistic workloads. Prometheus is designed as an application-oriented platform. Thus, we tested its end-to-end performance and showed that real-time deadlines set by application requests can be met without significant reduction in the quality of results. Additionally, we implemented a proof-of-concept mobile social application that utilizes Prometheus functionalities under real-time deadlines.

We investigated the vulnerability of Prometheus to attacks and established that the distributed, directed, labeled and weighted social multi-graph maintained by this service, in most cases, can successfully mitigate attacks. Also, Prometheus' socially-aware design reduces the opportunity of malicious peers to influence the inference request execution.

We have also proposed the *projection graph*, a model for studying the network properties of a P2P system such as Prometheus, that hosts the social graph of its users in a distributed fashion. We represented analytically the degree, node and edge betweenness centrality for the social and projection graph and discussed the relation between the two types of graphs in terms of these centrality measures. Because the analytical expressions

are heavily dependent on the topology of these graphs and how they are decentralized in the P2P system, we studied experimentally the correlation between their centrality metrics over a wide range of configurations.

Our experiments showed that within a range of 50–150 users mapped on a peer, there is an optimal organization of the projection graph, since peers score the highest average degree and node betweenness centrality. Also, up to this point, there is a high correlation between the properties of users and their peers, which degrades rapidly when the number of users per peer passes this optimal organization threshold. This correlation allows us to estimate with high accuracy the centrality of peers based on the centrality scores of users and their edges.

In addition, we investigated experimentally how peer centrality metrics can be used for the social graph traversal of search applications. We have shown that targeting top ranked degree or betweenness peers in the projection graph or top ranked degree on betweenness users in the social graph can achieve equal improvements in the performance of a social search. This is true when the search is executed either at the application or overlay layer.

The performance of Prometheus can certainly be optimized using our theoretical work on projection graphs, and applying several of the lessons described above on designing socially-aware distributed infrastructures.

First, benefiting from the slow changes that occur in social graphs, we plan to cache and pre-compute results of inferences and perform replication to nearby or central peers as proposed above to deal with peer churn or overloaded peers. Since trusted peers periodically check for new records in the social data files of their users, they can update their local copies and rerun the inferences to ensure consistency of these results.

Another way to improve application response time is to perform the following greedy optimization with respect to the parallel inference execution process. A peer that needs to

send secondary requests to other peers for the next hop of a multi-hop inference request could calculate the minimum set of secondary peers needed to cover all users for the next hop. This optimization could decrease request delays by reducing the number of peers to be contacted at each hop and thus the variability of request execution. This min-set can be further optimized using the network-wise closest peers of the transmitting peer. This greedy algorithm would not be the optimal one which takes into account global knowledge of how users' data are distributed on the subsequent hops (not just the next hop). Therefore, further theoretical and experimental work is needed to calculate how close to the optimal this greedy algorithm can perform, as well as the performance improvements when applied to a real distributed system such as Prometheus.

Furthermore, we can consider that in reality, the social graph is decentralized on peers not only in a socially-aware manner but also in a geographic manner. Therefore, we plan to experimentally examine the improvements of application performance for inference requests under this more realistic distribution of user data on geographically-close peers. We also plan to expand the set of social inferences to utilize the location and collocation of users.

Additionally, we consider Prometheus providing activity ontologies to social sensors and personal aggregators to support label consistency across multiple sensors and allow different sensors to provide input for the same labels. Finally, while our default ACPs prevent any single user from gaining knowledge of the entire graph, we intent to ascertain what level of collusion between users could expose the entire graph.

List of References

[Aca12] Academia. `http://www.academia.edu/`, 2012.

[AG07] Panayotis Antoniadis and Bénédicte Le Grand. Incentives for resource sharing in self-organized communities: From economics to social psychology. In *International Conference on Digital Information Management (ICDIM)*, pages 756–761, Lyon, France, December 11–13 2007. IEEE.

[AKFI10] Paul Anderson, Nicolas Kourtellis, Joshua Finnis, and Adriana Iamnitchi. On managing social data for enabling socially-aware applications and services. In *3rd ACM Workshop on Social Network Systems*, Paris, France, April 2010.

[ALPH01] Lada A. Adamic, Rajan M. Lukose, Amit R. Puniyani, and Bernando A. Huberman. Search in power-law networks. *Physical Review E*, 64(4):046135, September 2001.

[AMRS08] Luca Maria Aiello, Marco Milanesio, Giancarlo Ruffo, and Rossano Schifanella. Tempering kademlia with a robust identity based system. In *8th IEEE International Conference on Peer-to-Peer Computing (P2P)*, pages 30–39, September 2008.

[And10] Paul Anderson. GeoS: A service for the management of geo-social information in a distributed system. Master's thesis, Department of Computer Science, University of South Florida, Tampa, FL, USA, 2010.

[Ant71] Jac Anthonisse. The rush in a directed graph. Technical Report BN9/71, Stichting Mathematisch Centrum, Amsterdam, Netherlands, 1971.

[AR12] Luca Maria Aiello and Giancarlo Ruffo. LotusNet: Tunable privacy for distributed online social network services. *Computer Communications*, 35(1):75–88, December 2012.

[ATK+11] Tharaka Alahakoon, Rahul Tripathi, Nicolas Kourtellis, Ramanuja Simha, and Adriana Iamnitchi. K-path centrality: A new centrality measure in social networks. In *4th ACM Workshop on Social Network Systems*, pages 1–6, Salzburg, Austria, April 2011.

[Bat12] Battlefield. `http://www.battlefield.com/`, 2012.

[BBS+09] Randy Baden, Adam Bender, Neil Spring, Bobby Bhattacharjee, and Daniel Starin. Persona: An online social network with user-defined privacy. *ACM SIGCOMM Computer Communication Review*, 39(4):135–146, 2009.

[BCF09] Jonell Baltazar, Joey Costoya, and Ryan Flores. The real face of KOOBFACE: The largest web 2.0 botnet explained. Technical report, Trend Micro Research, July 2009.

[BD09] Sonja Buchegger and Anwitaman Datta. A case for p2p infrastructure for social networks: Opportunities and challenges. In *6th International Conference on Wireless On-Demand Network Systems and Services (WONS)*, pages 149–156, Snowbird, Utah, USA, 2009.

[BDMN06] Adam Barth, Anupam Datta, John C. Mitchell, and Helen Nissenbaum. Privacy and contextual integrity: Framework and applications. In *IEEE Symposium on Security and Privacy*, pages 184–198, 2006.

[BGLL08] Vincent D Blondel, Jean-Loup Guillaume, Renaud Lambiotte, and Etienne Lefebvre. Fast unfolding of communities in large networks. *Journal of Statistical Mechanics: Theory and Experiment*, 10, 2008.

[BGX+10] Aaron Beach, Mike Gartrell, Xinyu Xing, Richard Han, Qin Lv, Shivakant Mishra, and Karim Seada. Fusing mobile, sensor, and social data to fully enable context-aware computing. In *11th Workshop on Mobile Computing Systems and Applications*, pages 60–65, Annapolis, Maryland, 2010.

[BKI11] Jeremy Blackburn, Nicolas Kourtellis, and Adriana Iamnitchi. Vulnerability in socially-informed peer-to-peer systems. In *4th ACM Workshop on Social Network Systems*, pages 1–6, Salzburg, Austria, 2011.

[BMBR11] Yazan Boshmaf, Ildar Muslukhov, Konstantin Beznosov, and Matei Ripeanu. The socialbot network: When bots socialize for fame and money. In *27th Annual Computer Security Applications Conference (ACSAC)*, December 2011.

[BSK+12] Jeremy Blackburn, Ramanuja Simha, Nicolas Kourtellis, Xiang Zuo, Matei Ripeanu, John Skvoretz, and Adriana Iamnitchi. Branded with a scarlet "C": Cheaters in a gaming social network. In *21st International Conference on World Wide Web*, Lyon, France, April 2012.

[BSL+11a] Jeremy Blackburn, Ramanuja Simha, Clayton Long, Xiang Zuo, Nicolas Kourtellis, John Skvoretz, and Adriana Iamnitchi. Cheaters in a gaming metanetwork. In *International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS), Poster Session*, San Jose, CA, June 2011.

[BSL+11b] Jeremy Blackburn, Ramanuja Simha, Clayton Long, Xiang Zuo, Nicolas Kourtellis, John Skvoretz, and Adriana Iamnitchi. Cheaters in a gaming social network. *SIGMETRICS Performance Evaluation Review*, 39(3):101–103, 2011.

[BSVD09] Sonja Buchegger, Doris Schiöberg, Le-Hung Vu, and Anwitaman Datta. PeerSoN: P2P social networking: early experiences and insights. In *2nd ACM Workshop on Social Network Systems*, pages 46–52, Nuremberg, Germany, 2009.

[CDKR02] Miguel Castro, Peter Druschel, Anne-Marie Kermarrec, and Antony Rowstron. Scribe: A large-scale and decentralized application-level multicast infrastructure. *IEEE Journal on Selected Areas in Communications*, 20(8):1489–1499, 2002.

[CM08] Zack Coburn and Greg Marra. Realboy: believable twitter bots. `http://ca.olin.edu/2008/realboy/index.html`, 2008.

[CMS09] Leucio A. Cutillo, Refik Molva, and Thorsten Strufe. Safebook: a privacy preserving online social network leveraging on real-life trust. *IEEE Communications Magazine*, December 2009.

[CMS10] Leucio Cutillo, Refik Molva, and Thorsten Strufe. On the security and feasibility of Safebook: A distributed privacy-preserving online social network. *Privacy and Identity Management for Life*, 320:86–101, 2010.

[CNM04] Aaron Clauset, M. E. J. Newman, and Cristopher Moore. Finding community structure in very large networks. *Physical Review E*, 70(066111), 2004.

[Coh03] Bram Cohen. Incentives build robustness in bittorrent. In *Workshop on Economics of Peer-to-Peer Systems*, Berkeley, CA, June 2003.

[Con12] Alex Connock. Google+, Twitter, Facebook Connect ... Is your business cashing in on social media marketing in 2012? `http://www.grant-thornton.co.uk/thinking/elevate/index.php/article/google_twitter_facebook_connect_are_you_cashing_in_on_social_media_marketin/`, January 12th 2012.

[Dan09] Danchev Dancho. 56th variant of the koobface worm detected. `http://www.zdnet.com/blog/security/56th-variant-of-the-koobface-worm-detected/3414`, May 15th 2009.

[DBV+10] Anwitaman Datta, Sonja Buchegger, Le-Hung Vu, Thorsten Strufe, and Krzysztof Rzadca. Decentralized online social networks. *Handbook of Social Network Technologies and Applications*, (72):349–378, 2010.

[Dia12] Diaspora. `https://joindiaspora.com/`, 2012.

[DLLKA05] George Danezis, Chris Lesniewski-Laas, M. Frans Kaashoek, and Ross Anderson. Sybil-resistant DHT routing. In *In ESORICS*, pages 305–318. Springer, 2005.

[Dou02] John R. Douceur. The sybil attack. In *First International Workshop Peer-to-Peer Systems (IPTPS)*, pages 251–260, 2002.

[Dun98] Robin Dunbar. *Grooming, gossip and the evolution of language*. Harvard University Press, 1998.

[EB99] Martin G. Everett and Stephen P. Borgatti. The centrality of groups and classes. *Journal on Mathematical Sociology*, 23(3):181–201, April 1999.

[Ent12] Enthinnai. `http://www.enthinnai.com/`, 2012.

[EP06] Nathan Eagle and Alex (Sandy) Pentland. Reality mining: sensing complex social systems. *Personal and Ubiquitous Computing*, 10(4):255–268, 2006.

[Fac12a] Facebook. Graph API - facebook developers. `http://developers.facebook.com/docs/api`, 2012.

[Fac12b] Facebook. Recover your account through friends. `https://www.facebook.com/help/?page=228169557197326`, 2012.

[FLGC02] Gary William Flake, Steve Lawrence, C. Lee Giles, and Frans M. Coetzee. Self-organization and identification of web communities. *Computer*, 35(3):66–70, 2002.

[Fou12] Foursquare. `http://www.foursquare.com/`, 2012.

[Fre77] Linton C. Freeman. A set of measures of centrality based on betweenness. *Sociometry*, 40(1):35–41, 1977.

[Fre79] Linton C. Freeman. Centrality in social networks: Conceptual clarification. *Social Networks*, 1:215–239, 1979.

[Fre12a] FreedomBox. `http://freedomboxfoundation.org/`, 2012.

[Fre12b] FreePastry. Freepastry Java Library. `http://www.freepastry.org/`, 2012.

[Fri83] Noah E. Friedkin. Horizons of observability and limits of informal control in organizations. *Social Forces*, 62(1):57–77, 1983.

[Fri12] Friendster. http://www.friendster.com/, 2012.

[Gal11] Sean Gallagher. Researcher shows how to "friend" anyone on facebook within 24 hours. http://arstechnica.com/tech-policy/news/2011/11/researcher-shows-how-to-friend-anyone-on-facebook-within-24-hours.ars, 2011.

[GCX+05] Lei Guo, Songqing Chen, Zhen Xiao, Enhua Tan, Xiaoning Ding, and Xiaodong Zhang. Measurements, analysis, and modeling of BitTorrent-like systems. In *5th Conference on Internet Measurement*, pages 35–48, Berkeley, CA, USA, 2005.

[GGS+11] Kalman Graffi, Christian Gross, Dominik Stingl, Daniel Hartung, Aleksandra Kovacevic, and Ralf Steinmetz. LifeSocial.KOM: A secure and P2P-based solution for online social networks. In *IEEE Consumer Communications and Networking Conference*, Las Vegas, NV, USA, 2011.

[GJS+08] Ido Guy, Michal Jacovi, Elad Shahar, Noga Meshulam, Vladimir Soroka, and Stephen Farrell. Harvesting with SONAR: the value of aggregating social network information. In *26th Conference on Human Factors in Computing Systems*, pages 1017–1026, Florence, Italy, 2008.

[GKBB09] Ankur Gupta, Achir Kalra, Daniel Boston, and Cristian Borcea. MobiSoC: a middleware for mobile social computing applications. *Mobile Networks and Applications*, 14(1):35–52, 2009.

[GKF+06] Scott Garriss, Michael Kaminsky, Michael J. Freedman, Brad Karp, David Mazières, and Haifeng Yu. Re: Reliable Email. In *3rd Conference on Networked Systems Design and Implementation*, pages 297–310, San Jose, CA, USA, 2006.

[GMD06] Krishna P. Gummadi, Alan Mislove, and Peter Druschel. Exploiting social networks for internet search. In *5th Workshop on Hot Topics in Networks*, pages 79–84, Irvine, CA, USA, 2006.

[GN02] Michelle Girvan and M. E. J. Newman. Community structure in social and biological networks. *National Academy of Sciences of USA*, 99(12):7821–7826, June 2002.

[Gol07] Jennifer Golbeck. The dynamics of web-based social networks: Membership, relationships, and change. *First Monday*, 12(11), November 2007.

[Goo12a] Google. Latitude. `http://www.google.com/latitude/`, 2012.

[Goo12b] Google. Opensocial API. `http://code.google.com/apis/opensocial/`, 2012.

[Gra73] Mark S. Granovetter. The strength of weak ties. *The American Journal of Sociology*, 78:1360–1380, May 1973.

[Gun11] Susan Gunelius. Cash in on content and social media marketing in 2012. `http://www.forbes.com/sites/work-in-progress/2011/12/29/cash-in-on-content-and-social-media-marketing-in-2012/`, December 29th 2011.

[GWH07] Scott Golder, Dennis Wilkinson, and Bernardo A. Huberman. Rhythms of social interaction: messaging within a massive online network. In *3rd International Conference on Communities and Technologies*, East Lansing, MI, USA, 2007.

[Hi512] Hi5. `http://www.h5.com/`, 2012.

[IBK12] Adriana Iamnitchi, Jeremy Blackburn, and Nicolas Kourtellis. The Social Hourglass: an infrastructure for socially-aware applications and services. *IEEE Internet Computing, Special Issue on Social Networking Infrastructures*, 16(3):13–23, May/June 2012.

[IRSNF11] Adriana Iamnitchi, Matei Ripeanu, Elizeu Santos-Neto, and Ian Foster. The small world of file sharing. *IEEE Transactions on Parallel and Distributed Systems*, 22:1120–1134, July 2011.

[JSO12] JSON. `http://www.json.org/`, 2012.

[KAS+12] Nicolas Kourtellis, Tharaka Alahakoon, Ramanuja Simha, Adriana Iamnitchi, and Rahul Tripathi. Identifying high betweenness centrality nodes in large social networks. *Social Network Analysis and Mining*, 2012.

[KBI09] Zach King, Jeremy Blackburn, and Adriana Iamnitchi. BatTorrent: A Battery-Aware BitTorrent for Mobile Devices. In *11th International Conference on Ubiquitous Computing, Poster Session*, Orlando, FL, USA, 2009.

[KCB09] Eric D. Kolaczyk, David B. Chua, and Marc Barthélemy. Group betweenness and co-betweenness: Inter-related notions of coalition centrality. *Social Networks*, 31(3):190–203, 2009.

[Key12] Keyczar. Keyczar cryptographic toolkit. `http://www.keyczar.org/`, 2012.

[KFA+10a] Nicolas Kourtellis, Joshua Finnis, Paul Anderson, Jeremy Blackburn, Cristian Borcea, and Adriana Iamnitchi. Prometheus: User-controlled P2P social data management for socially-aware applications. In *11th ACM/IFIP/USENIX International Conference on Middleware*, pages 212–231, Bangalore, India, November 2010.

[KFA+10b] Nicolas Kourtellis, Joshua Finnis, Paul Anderson, Jeremy Blackburn, and Adriana Iamnitchi. Prometheus: Distributed management of geo-social data. In *7th USENIX Symposium on Networked Systems Design and Implementation (NSDI), Poster-Demo Session*, San Jose, California, 2010.

[KFIB09] Nicolas Kourtellis, Joshua Finnis, Adriana Iamnitchi, and Cristian Borcea. Mobidew: Socially-aware data management for mobile users. In *10th Workshop on Mobile Computing Systems and Applications (HotMobile), Poster Session*, Santa Cruz, CA, 2009.

[KGA08] Balachander Krishnamurthy, Phillipa Gill, and Martin Arlitt. A Few Chirps About Twitter. In *1st Workshop on Online Social Networks*, pages 19–24, Seattle, WA, USA, 2008.

[KI11] Nicolas Kourtellis and Adriana Iamnitchi. Inferring peer centrality in socially-informed peer-to-peer systems. In *11th IEEE International Conference on Peer-to-Peer Computing*, pages 318–327, Kyoto, Japan, August 2011.

[KN09] Indika Kahanda and Jennifer Neville. Using transactional information to predict link strength in online social networks. In *3rd AAAI International Conference on Weblogs and Social Media*, San Jose, CA, USA, 2009.

[KRS+06] Joseph S. Kong, Behnam A. Rezaei, Nima Sarshar, Vwani P. Roychowdhury, and P. Oscar Boykin. Collaborative spam filtering using e-mail networks. *Computer*, 39(8):67–73, 2006.

[LBBP+11] Hong Lu, A. Bernheim Brush, Bodhi Priyantha, Amy Karlson, and Jie Liu. SpeakerSense: Energy Efficient Unobtrusive Speaker Identification on Mobile Phones. *Pervasive Computing*, 6696:188–205, 2011.

[LCTC07] Ching-Ju Lin, Yi-Ting Chang, Shuo-Chan Tsai, and Cheng-Fu Chou. Distributed social-based overlay adaptation for unstructured P2P networks. In *IEEE Global Internet Symposium*, Anchorage, AK, May 2007.

[LD06] Jinyang Li and Frank Dabek. F2F: reliable storage in open networks. In *5th International Workshop on Peer-to-Peer Systems*, Santa Barbara, CA, USA, 2006.

[Les12] Jure Leskovec. Stanford large network dataset collection. `http://snap.stanford.edu/data/`, 2012.

[Lin12] LinkedIn. `http://www.linkedin.com`, 2012.

[LKG+08] Kevin Lewis, Jason Kaufman, Marco Gonzalez, Andreas Wimmer, and Nicholas Christakis. Tastes, ties, and time: A new social network dataset using Facebook.com. *Social Networks*, 30(4):330–342, 2008.

[LLDM08] Jure Leskovec, Kevin J. Lang, Anirban Dasgupta, and Michael W. Mahoney. Statistical properties of community structure in large social and information networks. In *17th International Conference on World Wide Web*, Beijing, China, April 2008.

[LLK10] Chris Lesniewski-Laas and M. Frans Kaashoek. Whanau: a sybil-proof distributed hash table. In *Proceedings of the 7th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, pages 111–126, San Jose, California, 2010. USENIX Association.

[Loo12] Loopt. `http://www.loopt.com/`, 2012.

[MGGM04] Sergio Marti, Prasanna Ganesan, and Hector Garcia-Molina. Sprout: P2P routing with social networks. In *Current Trends in Database Technology, EDBT 2004 Workshops*, volume 3268, pages 425–435, 2004.

[Min12] Minecraft. `http://www.minecraft.net/`, 2012.

[Mis11] Dan Misener. Rise of the socialbots: They could be influencing you online. `http://www.cbc.ca/news/technology/story/2011/03/29/f-vp-misener-socialbot-armies-election.html`, March 29th 2011.

[MLF+08] Emiliano Miluzzo, Nicholas D. Lane, Kristóf Fodor, Ronald Peterson, Hong Lu, Mirco Musolesi, Shane B. Eisenman, Xiao Zheng, and Andrew T. Campbell. Sensing meets mobile social networks: the design, implementation and evaluation of the CenceMe application. In *6th ACM Conference on Embedded Network Sensor Systems*, pages 337–350, Raleigh, NC, USA, 2008.

[MMC09] Sonia Ben Mokhtar, Liam McNamara, and Licia Capra. A Middleware Service for Pervasive Social Networking. In *1st International Workshop on Middleware for Pervasive Mobile & Embedded Computing*, Urbana Champaign, IL, USA, 2009.

[MMG+07] Alan Mislove, Massimiliano Marcon, Krishna P. Gummadi, Peter Druschel, and Bobby Bhattacharjee. Measurement and analysis of online social networks. In *7th SIGCOMM Conference on Internet Measurement (IMC)*, pages 29–42, San Diego, CA, USA, October 24–26 2007.

[MMH+06] Yutaka Matsuo, Junichiro Mori, Masahiro Hamasaki, Keisuke Ishida, Takuichi Nishimura, Hideaki Takeda, Koiti Hasida, and Mitsuru Ishizuka. POLYPHONET: an advanced social network extraction system from the web. In *15th International Conference on World Wide Web*, pages 397–406, 2006.

[Mor09] Evgeny Morozov. Swine flu: Twitter's power to misinform. `http://neteffect.foreignpolicy.com/posts/2009/04/25/swine_flu_twitters_power_to_misinform`, April 25th 2009.

[MPDG08] Alan Mislove, Ansley Post, Peter Druschel, and Krishna P. Gummadi. Ostra: leveraging trust to thwart unwanted communication. In *5th Conference on Networked Systems Design and Implementation*, pages 15–30, San Francisco, CA, USA, 2008.

[MRG+05] Petros Maniatis, Mema Roussopoulos, TJ Giuli, David S. H. Rosenthal, and Mary Baker. The LOCKSS peer-to-peer digital preservation system. *ACM Transactions on Computer Systems*, 23(1):2–50, 2005.

[MyS12] MySpace. `http://www.myspace.com/`, 2012.

[Net12] NetworkX. Networkx graph library. `http://networkx.lanl.gov/`, 2012.

[NG04] M. E. J. Newman and Michelle Girvan. Finding and evaluating community structure in networks. *Physical Review E*, 69:026113, February 2004.

[Nie74] Juhani Nieminen. On the centrality in a graph. *Scandinavian Journal of Psychology*, 15(1):332–336, 1974.

[Nis04] Helen F. Nissenbaum. Privacy as contextual integrity. *Washington Law Review*, 79, 2004.

[NM09] Rupert Neate and Rowena Mason. Networking site cashes in on friends. `http://www.telegraph.co.uk/finance/newsbysector/mediatechnologyandtelecoms/4413483/Networking-site-cashes-in-on-friends.html`, January 31st 2009.

[NXT10] Nam P. Nguyen, Ying Xuan, and My T. Thai. A novel method for worm containment on dynamic social networks. In *Military Communications Conference*, pages 2180–2185, San Jose, CA, November 2010.

[Ork12] Orkut. `http://www.orkut.com/`, 2012.

[OSH+07] J.-P. Onnela, J. Saramäki, J. Hyvönen, G. Szabó, D. Lazer, K. Kaski, J. Kertész, and A.-L. Barabási. Structure and tie strengths in mobile communication networks. *Proceedings of the National Academy of Sciences*, 104(18):7332–7336, 2007.

[PBB11] Susan Juan Pan, Daniel J. Boston, and Cristian Borcea. Analysis of fusing online and co-presence social networks. In *2nd Workshop on Pervasive Collaboration and Social Networking*, Seattle, WA, USA, 2011.

[PCT04] Bogdan Popescu, Bruno Crispo, and Andrew Tanenbaum. Safe and private data sharing with Turtle: Friends team-up and beat the system. In *12th Workshop on Security Protocols*, pages 213–220, 2004.

[PES+10] Josep M. Pujol, Vijay Erramilli, Georgos Siganos, Xiaoyuan Yang, Nikos Laoutaris, Parminder Chhabra, and Pablo Rodriguez. The little engine(s) that could: scaling online social networks. In *ACM SIGCOMM Conference*, pages 375–386, New Delhi, India, 2010.

[PGW+08] J. Pouwelse, P. Garbacki, J. Wang, A. Bakker, J. Yang, A. Iosup, D. H. J. Epema, M. Reinders, M. van Steen, and H. Sips. Tribler: A social-based peer-to-peer system. *Concurrency and Computation: Practice and Experience*, 20:127–138, 2008.

[Pla12] PlanetLab. `http://www.planet-lab.org/`, 2012.

[POL+09] Anna-Kaisa Pietiläinen, Earl Oliver, Jason LeBrun, George Varghese, and Christophe Diot. MobiClique: Middleware for Mobile Social Networking. In *2nd Workshop on Online Social Networks*, pages 49–54, Barcelona, Spain, 2009.

[PP12] PlanetLab (KyoungSoo Park and Vivek Pai). CoMon: A monitoring infrastructure for PlanetLab. `http://comon.cs.princeton.edu/`, 2012.

[RCM+11] Jacob Ratkiewicz, Michael Conover, Mark Meiss, Bruno Gonçalves, Snehal Patil, Alessandro Flammini, and Filippo Menczer. Truthy: Mapping the spread of astroturf in microblog streams. In *20th International Conference on World Wide Web*, Hyberabad, India, March 2011.

[RD01a] Antony Rowstron and Peter Druschel. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In *1st International Conference on Distributed Systems Platforms*, pages 329–350, Heidelberg, Germany, 2001.

[RD01b] Antony Rowstron and Peter Druschel. Storage management and caching in PAST, a large-scale, persistent peer-to-peer storage utility. In *18th Symp. on Operating Systems Principles*, pages 188–201, Banff, Alberta, Canada, 2001.

[RD10] Sam G B Roberts and Robin I M Dunbar. The costs of family and friends: an 18-month longitudinal study of relationship maintenance and decay. *Evolution and Human Behavior*, 32(3):186–197, 2010.

[RF08] Anirudh V. Ramachandran and Nick Feamster. Authenticated out-of-band communication over social links. In *1st Workshop on Online Social Networks*, pages 61–66, Seattle, WA, USA, 2008.

[RIF02] Matei Ripeanu, Adriana Iamnitchi, and Ian Foster. Mapping the Gnutella network. *Internet Computing, IEEE*, 6(1):50–57, January/February 2002.

[ROPT05] Mika Raento, Antti Oulasvirta, Renaud Petit, and Hannu Toivonen. ContextPhone: A prototyping platform for context-aware mobile applications. *Pervasive Computing, IEEE*, 4:51–59, 2005.

[RSR06] Amir H. Rasti, Daniel Stutzbach, and Reza Rejaie. On the long-term evolution of the two-tier gnutella overlay. In *25th IEEE International Conference on Computer Communications (INFOCOM)*, Barcelona, Spain, 2006.

[SBAG08] Xiaolin Shi, Matthew Bonner, Lada Adamic, and Anna C. Gilbert. The very small world of the well-connected. In *19th Conference on Hypertext and Hypermedia*, Pittsburgh, PA, USA, June 2008.

[SCM11] Tao Stein, Erdong Chen, and Karan Mangla. Facebook immune system. In *4th ACM Workshop on Social Network Systems*, Salzburg, Austria, April 2011.

[SCW+10] Alessandra Sala, Lili Cao, Christo Wilson, Robert Zablit, Haitao Zheng, and Ben Y. Zhao. Measurement-calibrated graph models for social network experiments. In *19th International Conference on the World Wide Web*, pages 861–870, Raleigh, NC, USA, 2010.

[SLC+11] Amre Shakimov, Harold Lim, Ramón Cáceres, Landon P. Cox, Kevin Li, Dongtao Liu, and Alexander Varshavsky. Vis-à-vis: Privacy preserving online social networking via virtual individual servers. In *3rd International Conference on Communication Systems and Networks*, Bangalore, India, 2011.

[SLCC08] Amre Shakimov, Harold Lim, Landon P. Cox, and Ramón Cáceres. Vis-à-vis: Online social networking via virtual individual servers. Technical report, Duke University, 2008.

[SM09] Hendrik Schulze and Klaus Mochalski. Internet study 2008/2009. `http://www.ipoque.com/sites/default/files/mediafiles/documents/internet-study-2008-2009.pdf`, 2009.

[SNLM11] Salvatore Scellato, Anastasios Noulas, Renaud Lambiotte, and Cecilia Mascolo. Socio-spatial properties of online location-based social networks. In *Proceedings of the 5th International AAAI Conference on Weblogs and Social Media (ICWSM)*, Barcelona, Spain, jul 2011.

[Spo12] Spokeo. `http://www.spokeo.com/`, 2012.

[SRA10] Emre Sarigol, Oriana Riva, and Gustavo Alonso. A tuple space for social networking on mobile phones. In *26th International Conference on Data Engineering*, pages 988–991, Long Beach, CA, USA, 2010.

[ST10] Michael Szell and Stefan Thurner. Measuring social dynamics in a massive multiplayer online game. *Social Networks*, 32(4):313–329, October 2010.

[Ste12] Steam. Steam Community API. `http://steamcommunity.com/dev`, 2012.

[SVCC09] Amre Shakimov, Alexander Varshavsky, Landon P. Cox, and Ramón Cáceres. Privacy, cost, and availability tradeoffs in decentralized osns. In *2nd Workshop on Online Social Networks*, pages 13–18, Barcelona, Spain, 2009.

[Sym11] Symantec. Social network attacks surge. `http://www.symantec.com/connect/blogs/social-network-attacks-surge`, 2011.

[TCL08] Dinh Nguyen Tran, Frank Chiang, and Jinyang Li. Friendstore: cooperative online backup using trusted nodes. In *1st International Workshop on Social Network Systems*, pages 37–42, Glasgow, Scotland, 2008.

[TD08] E. Rescorla T. Dierks. The Transport Layer Security (TLS) Protocol, version 1.2. `http://tools.ietf.org/html/rfc5246`, 2008.

[TLSC11] Nguyen Tran, Jinyang Li, Lakshminarayanan Subramanian, and Sherman S.M. Chow. Optimal sybil-resilient node admission control. In *30th IEEE International Conference on Computer Communications (INFOCOM)*, pages 3218–3226, Shanghai, P.R. China, 2011.

[TPI11] Alessandra Toninelli, Animesh Pathak, and Valerie Issarny. Yarta: A middleware for managing mobile social ecosystems. In *6th International Conference on Advances in Grid and Pervasive Computing*, pages 209–220, Oulu, Finland, 2011.

[TSGW09] Amin Tootoonchian, Stefan Saroiu, Yashar Ganjali, and Alec Wolman. Lockr: better privacy for social networks. In *5th International Conference on Emerging Networking Experiments and Technologies (CoNEXT)*, pages 169–180, Rome, Italy, 2009.

[Twi12a] Twitter. `www.twitter.com`, 2012.

[Twi12b] Twitter. Twitter Rest API. `https://dev.twitter.com/docs/api`, 2012.

[Twi12c] Twitter. Twitter Streaming API. `https://dev.twitter.com/docs/streaming-api`, 2012.

[UPvS11] Guido Urdaneta, Guillaume Pierre, and Maarten van Steen. A survey of DHT security techniques. *ACM Computing Surveys*, 43(2), January 2011.

[Val12] Valve. The Steam Community. `https://steamcommunity.com/`, 2012.

[Vam08] Vamosi Robert. Koobface virus hits facebook. `http://news.cnet.com/koobface-virus-hits-facebook/`, December 4th 2008.

[Váz03] Alexei Vázquez. Growing network with local rules: Preferential attachment, clustering hierarchy, and degree correlations. *Physical Review E*, 67(5):056104, 2003.

[WBS+09] Christo Wilson, Bryce Boe, Alessandra Sala, Krishna P. N. Puttaswamy, and Ben Y. Zhao. User interactions in social networks and their implications. In *4th European Conference on Computer Systems*, pages 205–218, Nuremberg, Germany, 2009.

[Wel88] Barry Wellman. Structural analysis: From method and metaphor to theory and substance. *Social structures: A network approach.*, pages 19–61, 1988.

[WF94] Stanley Wasserman and Katherine Faust. *Social Network Analysis: Methods and Applications*. Cambridge University Press, 1994.

[WS98] Duncan J. Watts and Steven H. Strogatz. Collective dynamics of "small-world" networks. *Nature*, 393(6684):440–442, June 1998.

[WS08] Fang Wang and Yaoru Sun. Self-organizing peer-to-peer social networks. *Computational Intelligence*, 24(3), 2008.

[XNR10] Rongjing Xiang, Jennifer Neville, and Monica Rogati. Modeling relationship strength in online social networks. In *19th International Conference on World Wide Web*, pages 981–990, Raleigh, NC, USA, 2010.

[Yah12] Yahoo! `http://www.yahoo.com/`, 2012.

[Yel12] Yelp. `http://www.yelp.com/`, 2012.

[YGKX08] Haifeng Yu, Phillip B. Gibbons, Michael Kaminsky, and Feng Xiao. Sybil-limit: A near-optimal social network defense against sybil attacks. In *IEEE Symposium on Security and Privacy*, pages 3–17, 2008.

[YKGF06] Haifeng Yu, Michael Kaminsky, Phillip B. Gibbons, and Abraham Flaxman. Sybilguard: defending against sybil attacks via social networks. In *SIGCOMM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, pages 267–278, Pisa, Italy, 2006.

[YZLT09] Stephen J.H. Yang, Jia Zhang, Leon Lin, and Jeffrey J.P. Tsai. Improving peer-to-peer search performance through intelligent social search. *Expert Systems with Applications*, 36(7):10312–10324, 2009.

[ZCZ+09] Zhichao Zhu, Guohong Cao, Sencun Zhu, Supranamaya Ranjan, and Antonio Nucci. A social network based patching scheme for worm containment in cellular networks. In *28th IEEE International Conference on Computer Communications (INFOCOM)*, pages 1476–1484, April 2009.

Appendices

Appendix A: List of Acronyms

P2P	Peer-to-Peer
PL	PlanetLab
SG	Social Graph
PG	Projection Graph
OSN	Online Social Network
SKS	Social Knowledge Service
TF2	Team Fortress 2
DHT	Distributed Hash Table
TPL	Trusted Peer List
RTT	Round Trip Time
SNA	Social Network Analysis
DC	Degree Centrality
NBC	Node Betweenness Centrality
EBC	Edge Betweenness Centrality
CDCU	Degree Centrality of Users
CNBCU	Cumulative Node Betweenness Centrality of Users
CEBCU	Cumulative Edge Betweenness Centrality of Users
DCP	Degree Centrality of Peers
NBCP	Node Betweenness Centrality of Peers
EBCP	Edge Betweenness Centrality of Peers
IEEE	Institute of Electrical and Electronic Engineers
ACM	Association of Computing Machinery

Appendix B: Pseudocode for Generation of Synthetic Networks

Here we present the pseudocode of the algorithm used for the generation of synthetic social networks (Chapter 3 and Chapter 4) based on the model introduced in [Váz03] and refined in [SCW$^+$10].

Input : The probability u (= 0.8 [SCW$^+$10]) of attachment with an existent node, the number of pairs κ (= 1) of existing nodes to be connected, and the desired size N of the Synthetic Social Network.

Output : A graph $G = (V, E)$ which is a Synthetic Network with Social Network Power-Law properties.

begin
- Let $G = (V, E)$ be a graph with a single node $x \in V$;
- **while** $|V| \leq N$ **do**
  - $u' \leftarrow$ a real chosen uniformly at random from $[0, 1]$;
  - **if** $u' \leq 1 - u$ **then**
    - $z \leftarrow$ a node chosen uniformly at random from V;
    - Let y be a new node;
    - $V \leftarrow V \cup \{y\}$;
    - $E \leftarrow E \cup \{(y, z)\}$;
    - **for** $i = 1$ *to* κ **do**
      - Let E' be the set of potential edges, i.e., the set of all pairs (a, b) such that $(a, b) \notin E$;
      - $(a, b) \leftarrow$ an edge chosen uniformly at random from E';
      - $E \leftarrow E \cup \{(a, b)\}$;
    - **end**
  - **end**
  - **else**
    - Let E' be the set of potential edges, i.e., the set of all pairs (a, b) such that $(a, b) \notin E$;
    - $(a, b) \leftarrow$ an edge chosen uniformly at random from E';
    - $E \leftarrow E \cup \{(a, b)\}$;
  - **end**
- **end**

end

Algorithm 1: Pseudocode for the generation of synthetic social networks following the algorithm from [SCW$^+$10].

Appendix C: Reuse of Material from Copyrighted Sources

Figure C.1 shows the permission to reuse material from the paper [KFA+10a] in Chapter 2 and Chapter 3 of this dissertation.

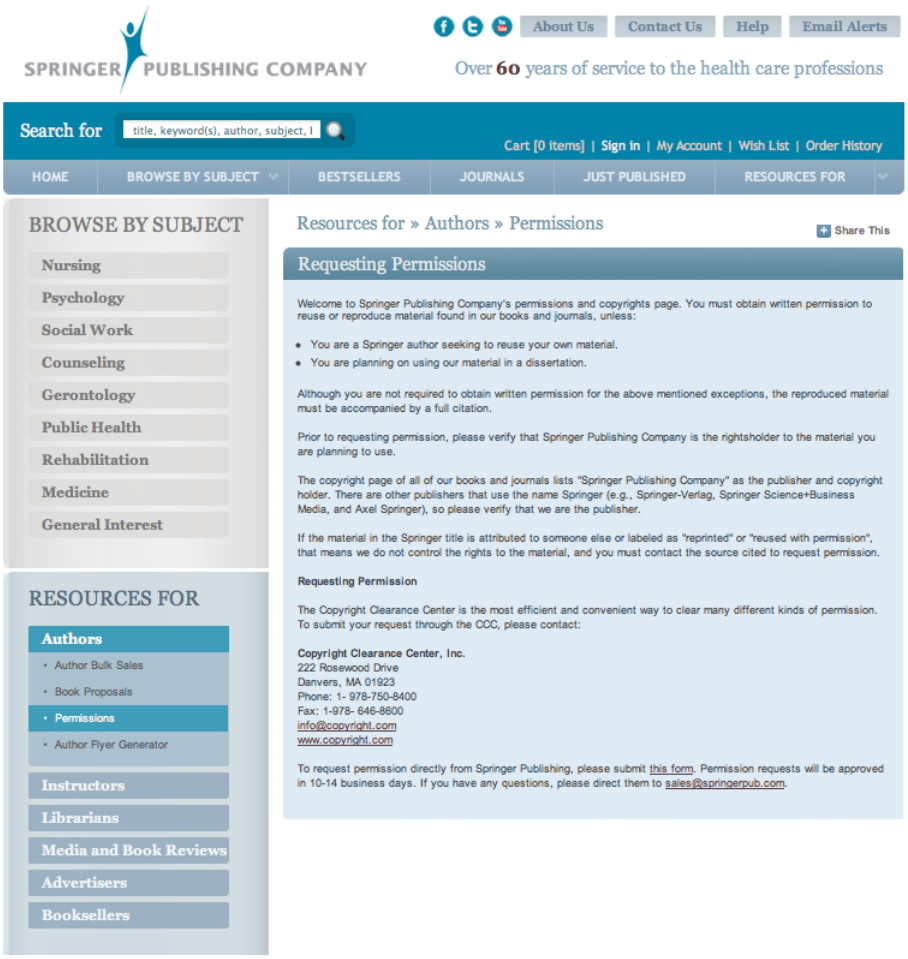

Figure C.1: Permission for reusing material from 11th ACM/IFIP/USENIX International Conference on Middleware, November-December 2010.

Appendix C: (continued)

Figure C.2 shows the approval to reuse material from the paper [IBK12] in Chapter 2 of this dissertation.

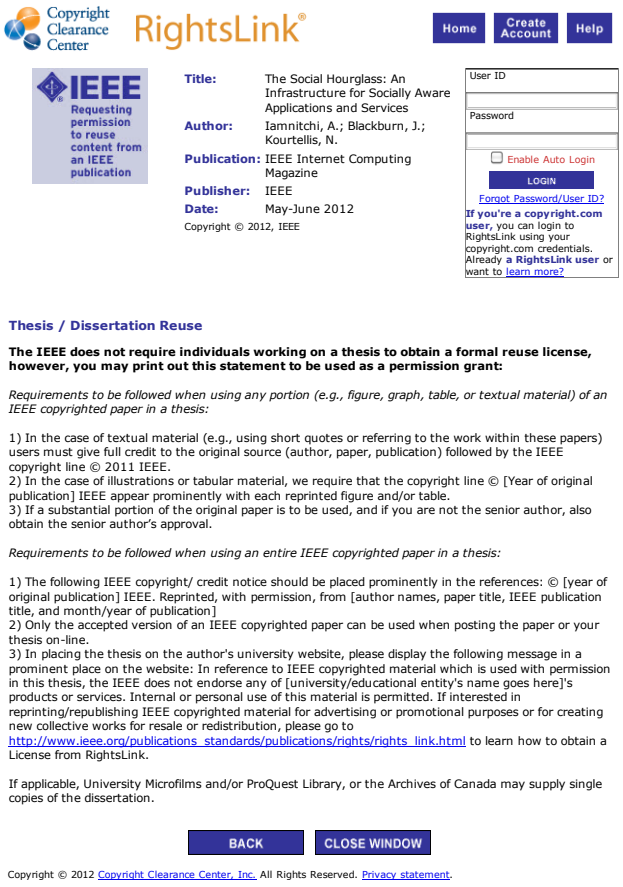

Figure C.2: Approval for reusing material from IEEE Internet Computing Magazine, May-June 2012.

Appendix C: (continued)

Figure C.3 shows the approval to reuse material from the paper [KI11] in Chapter 5, Chapter 6 and Chapter 7 of this dissertation.

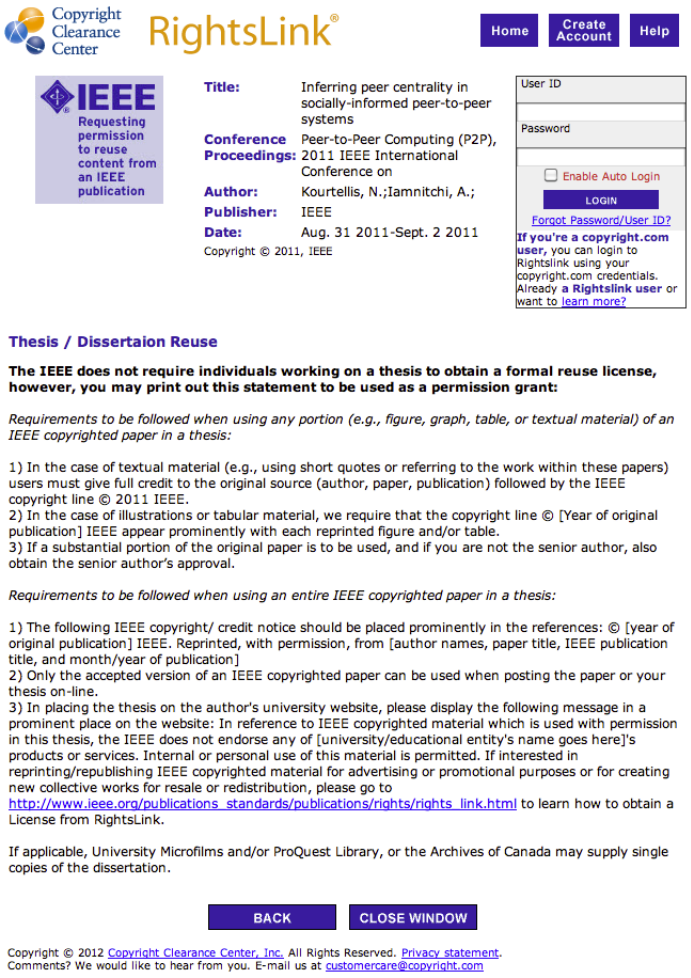

Figure C.3: Approval for reusing material from 11th IEEE International Conference on Peer-to-Peer Computing, August-September 2011.

About the Author

Nicolas Kourtellis is a PhD Candidate in the Department of Computer Science and Engineering at the University of South Florida. His research interests include the design of large-scale, socially-aware distributed systems, social networks analysis, design of socially-aware applications and services, and mobile computing. He received a MSc in Computer Science from the University of South Florida in 2008 and a Diploma (MEng) in Electrical and Computer Engineering from the National Technical University of Athens, Greece, in 2006. He is a full member of the Tau Beta Pi (ΤΒΠ) Engineering honor society, the Eta Kappa Nu (HKN) Electrical and Computer Engineering honor society of the Institute of Electrical and Electronics Engineers (IEEE), the Phi Kappa Phi (ΦΚΦ) all-discipline honor society, and a student member of the IEEE.