GeoS: A Service for the Management of Geo-Social Information in a Distributed System

by

Paul Anderson

A thesis submitted in partial fulfillment
of the requirements for the degree of
Master of Science in Computer Science
Department of Computer Science and Engineering
College of Engineering
University of South Florida

Major Professor: Adriana Iamnitchi, Ph.D.
Cristian Borcea, Ph.D.
Jay Ligatti, Ph.D.

Date of Approval:
May 18, 2010

Keywords: Data Management, Peer-to-Peer Systems, Social Graph, Socially-Aware Applications, Privacy Protection

Copyright © 2010, Paul Anderson

Dedication

To my parents. Thank you for teaching me the importance of an education.

Acknowledgments

This thesis would not have been possible without the assistance of many people. My colleagues, Nicolas Kourtellis and Joshua Finnis, designed the Prometheus architecture, helped in experimentation, and were always there for me to discuss ideas with. Jeremy Blackburn brought up many great ideas and questions throughout the design of GeoS. Daniel Boston and Susan Juan Pan from NJIT provided me with social data that was very useful for experimentation. Mayur Palankar's presentation on social networking is where the first spark of the idea for GeoS started. Sachae Soso created the first applications that made use of GeoS. I would like to thank my major professor, Dr. Adriana Iamnitchi, most of all, for all the support she gave, knowledge she shared, and for allowing me to do my research with her.

This research was supported by the National Science Foundation under Grant No. CNS-0831785. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the sponsors.

Table of Contents

List of Tables

List of Figures

GeoS: A Service for the Management of Geo-Social Information in a Distributed System

Paul Anderson

Abstract

Applications and services that take advantage of social data usually infer social relationships using information produced only within their own context, using a greatly simplified representation of users' social data. We propose to combine social information from multiple sources into a directed and weighted social multigraph in order to enable novel socially-aware applications and services. We present GeoS, a geo-social data management service which implements a representative set of social inferences and can run on a decentralized system. We demonstrate GeoS' potential for social applications on a collection of social data that combines collocation information and Facebook friendship declarations from 100 students. We demonstrate its performance by testing it both on PlanetLab and a LAN with a realistic workload for a 1000 node graph.

Chapter 1:

Introduction

The popularity of user-generated content tools exposes an unprecedented amount of information about the interaction between Internet users. In particular, two recent classes of Internet applications are revealing much social information: popular online social networks (such as Facebook or LinkedIn); and widely adopted collaborative tools (such as CiteULike or Delicious) that provide a rich information fabric through tags, annotations, and organization of text. In addition, the ubiquitous GPS and Bluetooth-capable mobile devices can generate yet another layer of social data through location and collocation information.

This unprecedented amount of social information can be leveraged in support of social computing applications that could really harness the power of social ties between users. However, the state of the art in recording and maintaining social information places three significant obstacles in the way of developing innovative social applications. First, many sources of social information rely on declared relationships that all have the same weight, which obscures relevant information in terms of trust, shared interest, social solidarity, etc. Some social networks have tried to correct this particular problem. For example, Facebook has attempted to use the volume of interaction between users to rank their relationships, but this rank is not available to third-party applications [Won10].

Second, the manual declaration of relationships can leave out relevant social ties or pressurize users into declaring unsubstantiated relationships. The implicit social incentives in such systems - the more friends a user has, the better - motivate users to declare a large number of friends. For example, one study of Facebook [WBS+09] found that more than a third of Facebook users have 100 or more friendships declared. The same study also found that for most users, 70% of their interactions on Facebook happen with only 20% of their friends. These two facts show that users declare many friends, but then only interact with a small subset of them. Another study found that a user who has many friends on an online social network is perceived as more confident and popular than if that same user had less friends [KRZBS07].

Third, the business model of current online social networks leads to centralized storage and management of user social data, and gives users little control over their social data. For example, all existing systems are centralized: Online social networks profit from using and/or selling details on their users' social networks for

marketing purposes [NM09]. This business model can also restrict application developers by limiting what social data they are allowed to access and how they can use this data [Facb].

This thesis proposes GeoS, a geo-social data management service that addresses these issues and allows for the rapid development of *socially-aware applications* that make use of the vast amounts of social data available. GeoS automatically infers social relationships by receiving details of social interactions from third-party geo-social applications that we refer to as *social sensors*. The social relationships reported by GeoS can be much more complex than the simple boolean friendship declarations used in many online social networks, as they record common activities and estimate how strongly those activities relate two people. Examples of activities include collocation at a concert, discussing a football game on Facebook, and sending an SMS to a friend to ask about the upcoming hiking trip. GeoS also provides a set of social inference functions that can be used by third party applications to find indirect social connections, the users interacted with most, the strength of a social relationship, and socially connected users who are physically nearby, for example. Finally, GeoS is built as a part of the Prometheus system that has a fully decentralized architecture on a peer-to-peer network. Prometheus provides users complete control over where their social data is stored and who has the right to access it. Prometheus also provides the support for GeoS' distributed management of social data.

1.1 Motivation

There are three main motivations behind the development of GeoS. The first is to speed up the development of socially-aware applications by allowing them to use a third-party service that collects and manages social data and provides social inferences based on the data it has collected. Most online social networks already give some access to social information, but they limit the amount of information accessible: in particular, they can only gather information on how users interact with their service and more importantly, some networks limit what data can be accessed and how it can be used [Facb].

The second motivation for GeoS is to provide more detailed social information than current services do. Most research so far has only considered one or two sources of social information [GM04, AYP+05, MPDG08] and used a simplified representation of the social network [AHK+07, LKG+08, ZC10]. Because each source of social information is limited to its own context, a single source can only provide a piece of a user's complete social information. If social information is being represented by a simple graph data structure, important information is likely to be lost because very little is known about relations beyond the simple fact that certain ones exist. A complete view of users' social information allows for analyses that

are more detailed. Note that the simple graph most sociological studies use to represent social networks is likely chosen due to the difficulties inherit to surveys [BKKS84, MS63] (one of the main providers of social information in sociology).

The third motivation is to give users complete control over the storage of their social information by decentralizing the data managed by GeoS. This is in stark contrast to current online social networks where, for example, Facebook keeps users' social data even after they delete their account [Kha09]. When an organization gathers data about many users in one centralized spot, there is the possibility of an entity using this data to monitor those users, creating what is often called a "Big Brother" scenario. In addition, if an organization has centralized control over a set of social data, users may use it less due to possible privacy concerns or the organization may limit access to the data, causing the progression of advancements in social computing applications and related social network analysis to be stifled. Finally, if an application was built to use a single centralized service and that service went offline, the application would no longer function until it was rewritten.

1.2 Solution, Thesis Contribution, and Roadmap

GeoS is a distributed service that collects social data, manages the vast amount of available social data using a novel social graph data structure, and solves social inference requests to support a variety of social computing applications. GeoS receives social information from different sources, including cell phone call logs, friendship declaration on social networking sites (such as Facebook and MySpace), and emails, with no limit on the possible sources. The received social information is recorded in a social graph that has a tag and weight associated with each edge that connects two users. The tag and weight system records the strength of each activity or social group that relates two people, and since the edges are directed, relationship strength can be asymmetric. The particular graph data structure used in GeoS enables finding nuanced social relations in terms of the type and strength, typically specified by an application. As a part of the Prometheus architecture, GeoS runs on a peer-to-peer network that allows users to have full control of where they want their social information stored. We also present in this thesis experimental results that show the performance and capabilities of GeoS.

In Chapter 2:, we present applications and scenarios that have motivated the design of GeoS. Chapter 3: contains the objectives that were in mind while designing GeoS, the architecture of Prometheus, and the API and functionality of the social inference functions within GeoS. Chapter 4: lists the implementation details of GeoS. We detail how we prepared for experimentation in Chapter 5:. That is followed by the experimental

results in Chapter 6:. Related work is presented in Chapter 7:. We propose future work in Chapter 8:. In Chapter 9:, we summarize the work presented in this thesis.

Chapter 2:

Target Applications and Motivating Scenarios

Many classes of applications can benefit from the services offered by GeoS. Some of those applications would be impossible to do with current services, while others require less work to implement or can gain more features by using GeoS.

One example application allows users to invite friends who went separately to a sporting event to join them afterwards. Using current technology, this could be done by manually using Google Latitude to see which friends are located near the user, and then manually sending a message out to each friend the user wants to invite. With GeoS, an application could simply query GeoS to infer which friends of the user are also at the game and invite them. The application that uses GeoS could also provide more features, such as filtering the invited friends based on relationship strength.

A similar type of application would be one that simplifies notifying select friends about an upcoming event. One of the easiest ways to do this right now would be to send an invitation to select friends on a social networking site like Facebook. The problem with this is that most people have Facebook friends in the hundreds [WBS+09], and so it would be time consuming to select whom to invite from such a large set. GeoS allows users to filter their friends easily. For example, a user can limit friendships returned to only friends with a strong relationship and/or only friends with which they do certain activities. In addition, GeoS allows the user to invite friends of friends. Thus, a user could invite to an outdoor event all his hiking friends, along with their friends. In contrast to the last example application, the present location of the users is not checked and the application tests the amount that hiking relates the users instead of the general relationship strength between them. Also, in this example the friends of friends were returned instead of only direct connections.

Another application silences phone calls when the user is in a particular social context and the incoming phone call is outside that context, regardless of location. The social context of the user is inferred by looking at the social connections the user has with those the user is currently collocated with. Because it does not rely on location, this application could silence the user's phone automatically when the user is in a meeting, as well as when they are at lunch with coworkers. The application could infer the user's social context even when no direct social connection exists between them and someone at their location. For example, the phone could

silence personal calls when in an interview with the regional manager whom the user has never met before (but an indirect connection from the user to their boss to the regional manager exists.) Several applications meant to solve the common problem of cell phone interruption have already been developed, but they do not provide as many features as the proposed one that uses GeoS. SenSay [SSF+03] tries to discover if the phone should be silenced through two methods: if the user has scheduled an interruptible event in their calendar or if the user is speaking. The first method requires work from the user while the second will not function if the phone does not pick up any loud sounds (from the user talking softly or not talking at all.) A commercial application, Locale [Loc], silences the phone when the user is at certain locations defined by the user, but this requires the user to set up these locations ahead of time. Neither of these solutions differentiates between personal and business callers.

King et al. [KBI09] have proposed BatTorrent, a derivative of BitTorrent meant for mobile phones that uses social knowledge to decide whether to trust a "low battery" message from another peer. When the application receives a low battery message from a trusted source, the receiver no longer requires the other peer to upload, allowing it to conserve battery power. BatTorrent needs GeoS to check whether the sender is a trusted person: an untrusted user could report that their battery is low in order to game the system and avoid having to upload. GeoS infers trust for BatTorrent by determining the strength of the social connection from the seeder to the peer with the low battery. The intuition is that the number and intensity of existing social ties (even ones that do not directly connect the two users) can in this case be transformed into social incentives for good behavior and an estimate of social trust. Without GeoS, the user would have to decide manually whether users were trusted or not.

A few applications already use social knowledge to great effect, but we believe many of them would have results that are more precise by using the more complex social data from multiple sources provided by GeoS. `RE:` [GKF+06] uses white lists to help prevent the spreading of spam. What makes RE: special is that it allows "friend-of-friend" white listing: if user A white lists user B and user B white lists user C, emails from user C will be allowed to reach user A. The current system requires users to declare which senders should have their emails allowed, but GeoS would allow the white list to be inferred automatically.

F2F is a peer-to-peer network that uses social information to provide users with incentives for sharing [LD06]. The authors only test the example application (BlockParty) with social graphs from Orkut and a dating site, but as stated in the paper, users may not be willing to share their data or resources with those in the tested graphs. If a similar application wanted to automatically decide whom to share with, GeoS provides it with a weight rather than a simple boolean value found in Orkut. In addition, interesting possibilities could

open up if the type of relation that connects people is also used. For example, a user could possibly only share their concert photos with their music friends.

Peerspective uses social knowledge to make searching the web more powerful [GMD06]. The weights and edge types in GeoS could be used to decide how strongly to bias each user's search results. For example, if a user searches for "team schedule", the system could check the user's sports friends to see what type of teams they usually searched for. The higher the sports weight connecting the user to his or her friend, the more the two must interact about sports, and thus, the more the friend's results should matter on the subject.

One final possible use of GeoS, and perhaps the most important, is in the area of sociological studies. GeoS allows for sociology researchers to record social information from multiple sources in a complex social graph with little programming required. GeoS together with social sensors can simplify sociological studies while giving more detailed results.

Chapter 3:

GeoS

In the following sections, we shall describe the objectives of GeoS along with how it meets those objectives (Section 3.1). This will be followed by the architecture of GeoS (Section 3.2). This section will include an overview of Prometheus, the system of which GeoS is a part, with a focus on the role of GeoS within Prometheus and how GeoS interacts with the other parts. The design of GeoS and how it interacts with users' applications is also included in the architecture description. The specifications of the algorithms GeoS uses to complete social inferences will follow this (Section 3.3).

3.1 Design Objectives and Performance Requirements

The purpose of GeoS is to both manage geo-social information submitted by diverse social sensors while reducing privacy-loss risks and answer a variety of social inference requests in a reasonable time. To this end, the design objectives are:

- store social information in a decentralized manner
- accept input from a variety of social sensors
- manage the geo-social information it receives

The performance requirements are:

- reasonably fast processing of social inference requests
- light communication load

3.1.1 Social Sensors

A main objective of GeoS is to allow any source of geo-social information, or what we call "social sensors", to send input to GeoS. We expect the following social sensors to be developed:

- mobile devices that find users who are at the same location as the owner and then determine at which location the interaction is occurring

- applications attached to social networking sites, email applications, instant messaging applications, and other providers of text-based social interaction that establish the topic discussed by looking for keywords
- devices that report information about the user's phone call and SMS details
- services that record the sharing of files between users
- extensions to video games that observe users interacting together

However, GeoS should not have to be modified whenever a new social sensor is developed.

Thus, the social input message format must be open, but the system also has to keep network communication at a low level and privacy-loss risks at a minimum. To meet these objectives, we keep the input message format small, simple, and generic. The small message size creates little network traffic, which is important when we expect mobile devices to be one of the main providers of input. The simple nature of the input has enough information for GeoS to build a social graph over time, yet reduces the possible privacy-loss to network security attacks, such as, man-in-the-middle attacks. Finally, its generic format allows any social sensor to begin sending social information without any changes to GeoS.

In our design we made the following assumptions about social sensors:

First, social sensors are sophisticated enough to aggregate the information they collect. This will allow them to both recognize trends and limit the number of messages they send to GeoS. A type of trend we expect social sensors to detect, using activity recognition through ubiquitous computing [LHP+07, PFF+03], is routines. For example, if a user goes to the gym each day, we expect their device not to report a social interaction with the other daily gym goers around them. We assume most social sensors will report in at most once a day, but that the sensor will send several messages consecutively at that time. Aggregating input is also good for privacy because it protects against a malicious user knowing exactly what another user is currently doing. For example, if input was not aggregated and a malicious user saw that users A and B just had a movie interaction, the malicious user would know A and B are likely at a movie theater currently. The more time between sending aggregated input, the more difficult it would be to build up a user's schedule, but then there comes the problems of the social sensor potentially losing its information (e.g. from becoming lost or damaged) and the information not being in the social graph for other applications to make use of it.

Second, while we expect social sensors will likely aggregate information, we still expect the value of an individual input to be small because an accurate relationship weight will be built slowly over time. Since users are likely to have applications use relationship strength as a factor when allowing or denying privileges,

it is better in terms of privacy for a relationship weight to begin low and slowly grow to an accurate weight than start out high and be reduced. A benefit of individual inputs being small is that the loss of a single input will not strongly affect a user's social information. To combat the slow building of a relationship weight, we expect applications will "bootstrap" the user's social information by looking at the user's past social interactions through such avenues as social networking websites and emails.

Third, social sensors can also identify the strength an individual interaction affects the connection between two users by looking at many variables, such as, the length of the interaction, how socially active the users are, and how interested the users are in the activity they did together. We also assume that the social sensors will be able to ascertain what activity it was that the two users did together. From the above expected social sensor types, co-location input could take the GPS coordinates of the location, use reverse geocoding to find the address of the location, and finally use some type of database to find what activity is related with that location. The social sensors that handle text-based social interactions could mine the text for keywords that show what activity the users were discussing by text. The phone call device could use speech recognition to again, look for keywords to discover what topic the users discussed.

3.1.2 Decentralization of Social Data

When managing the geo-social data, a main objective is to make the users feel safe by giving them control over the storage of their social data and again, keep potential privacy-loss to a minimum. To this end, we decided that GeoS would operate on a decentralized platform. The decentralized platform provides many benefits to users, including:

- users are given command over their social data
- "Big Brother" scenarios are prevented by not giving an entity access to the social data of all users
- the effects of a security attack are lessened versus a centralized platform because if a single system is compromised, a reduced number of users are affected

3.1.3 Management of Social Data

In terms of managing geo-social data, GeoS is responsible for the following:

- organizing users' social information in a graph data structure
- applying and tracking input

- propagating social information across the decentralized system
- submitting social inference requests to other users when social information on those users is needed

3.1.4 Social Inference Requests

GeoS needs to provide a set of social inference functions as API for applications. These inference functions need to be reasonably efficient since users are likely to be interacting directly with them. Given the likely scale of the graph (for example, Facebook has 400 million active users [Faca]), the fact that the graph will be distributed across many peers, and potential privacy issues, some social metrics, such as betweenness centrality, are too complex. The ones implemented in GeoS, however, are simple and can still have a high impact on socially-aware applications.

3.2 Architecture

We built the GeoS architecture so that the service could manage social data in a distributed system. GeoS is a part of Prometheus, a peer-to-peer system that provides users with control over where their social data is stored and who accesses it.

3.2.1 Overall Architecture of Prometheus and GeoS as a Component

We designed GeoS to run on a peer-to-peer distributed system and cooperate with other services that are specific to handling the management of a distributed system [KFA+10]. We named the architecture comprised of GeoS together with the services that handle the distributed system *Prometheus*. The components that make up Prometheus are:

- *Ermis* - manages the decentralized peer-to-peer network and groups peers based on which ones a user declares as trusted
- *Registrar* - manages which peers users trust
- *Gateway* - enforces privacy through encryption and access control lists (ACLs)
- *GeoS* - manages the social data of users

An overview of the Prometheus architecture can be found in Figure 1.

Each peer that is a part of Prometheus runs a copy of the four services. Users of Prometheus declare peers as trusted and have their social data stored on those peers. Since most people are sensitive about the privacy

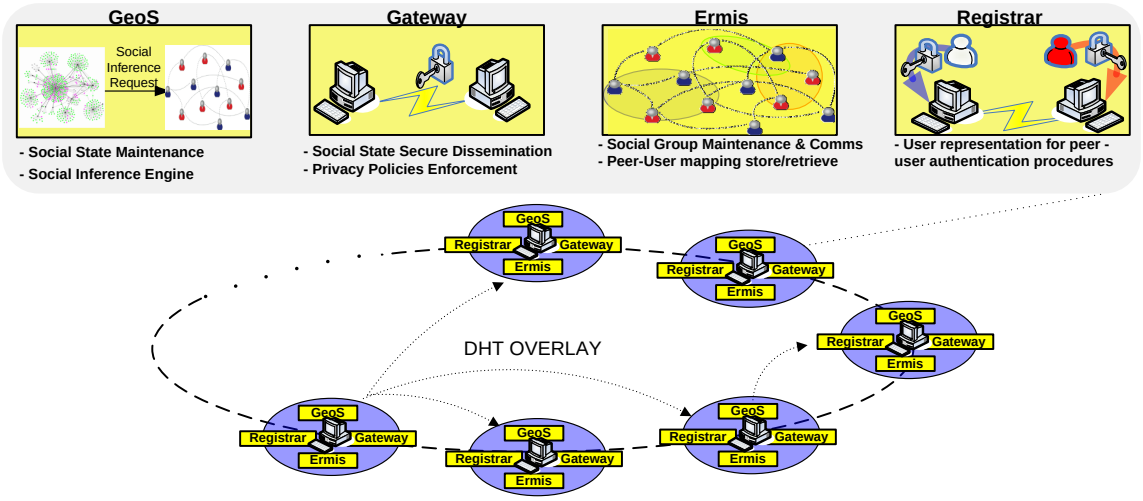


Figure 1. An Overview of the Prometheus Architecture.

of their social data, we expect users to choose peers owned by users that they have social relationships with as their trusted peers. Thus, GeoS can aid users in choosing their trusted peers by using its social inferences. Ermis maintains a list of which peers are trusted by a user in that user's trusted peer list (TPL). These TPLs contain the IP addresses of the peers that are trusted for the user and are updated whenever a trusted peer joins the peer-to-peer network. As will be described in detail later, all messages in the Prometheus system first go through the Gateway so that it can enforce privacy and so that only one component is in charge of the routing of messages to the correct peers. When the Gateway needs to know where to route a message, it requests the trusted peer list for the destination user from Ermis.

The user defines an ACL, which in turn the Gateway uses to decide who has access to the user's data. The ACLs are a powerful form of privacy and let the user use many parameters when deciding who can and cannot access their data. Such parameters include the distance in the social graph that the request for data originated, how the users are socially connected, and who the requesting user is. The Gateway uses public-key encryption when transferring messages between peers in order to protect the privacy of the user from possible data interceptions.

Ermis acts as an interface for SCRIBE [RKCD01] which is itself built upon Pastry [RD01]. As an implementation of distributed hash tables [PRR97], Pastry is a peer-to-peer network overlay that allows for the quick routing of messages and is able to adapt to peers leaving and joining the network frequently. SCRIBE organizes peers into groups, which, in the case of Prometheus, allows it to contact all the trusted peers of a user quickly through anycast or multicast.

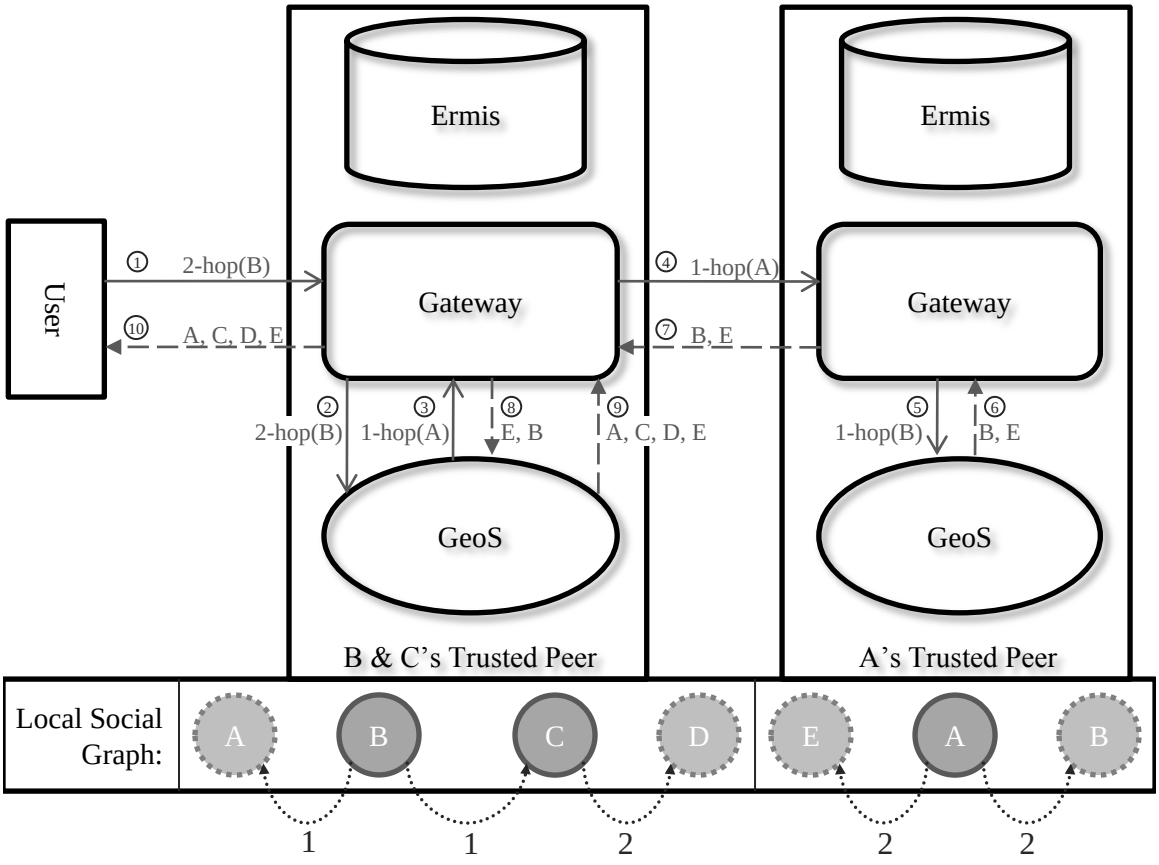
3.2.2 Communication Between GeoS and Other Components

The components run as separate services and communicate with each other using sockets to allow for complete modularity. All communication with GeoS first goes through the Gateway; all input, requests, results, and even communication between GeoS peers must go through the Gateway. In this way, the Gateway can check its ACLs, decrypt incoming messages, and encrypt outgoing messages. This also allows GeoS to almost "blindly" send messages to other peers since all it needs to do is tell the Gateway which user the message is to be sent to. If the Gateway on peer A receives a message for a user who has not set peer A as trusted, the Gateway will forward the request to a correct peer. Due to all communication to GeoS first going through the Gateway, the format of messages to the Gateway shall be defined.

Messages to the Gateway use the following the generic format: "*component* <-r> | <*destination_IP*> | *destination_user* | *requesting_user* | *message* <:::*reply_IP*:::*reply_port* :::*ID*>". Note that optional fields are placed in angle brackets and variables are italicized. Based on the previous generic format, to make a social inference request, an application sends a message to the Gateway using the following format: "geos-r || *destination_user* | *requesting_user* | *inference(destination_user)*:::*reply_IP*:::*reply_port*:::*ID*". Social input uses the following format: "geos || *destination_user* | *destination_user* | *input(destination_user)*".

Fields are delimted by the pipe character. The *component* field defines which component of Prometheus the message is destined for. The optional *destination_IP* field is only used when a message is to be delivered to a specific peer. If that field is empty, a peer trusted by *destination_user* is chosen. The ACL of *destination_user* is checked to see if *requesting_user* is allowed to access their social data. The *message* field is simply the message to be delivered. The optional -r in the component field and :::*reply_IP*:::*reply_port*:::*ID* after the message field are used when the message expects a reply. In order to save resources, the Gateway and GeoS close all socket connections as soon as they receive the entire message. However, in order for the Gateway to send a reply, it must know at what IP and port the sender will be listening. The ID is for the sender to identify the reply.

The only time GeoS sends a message destined for Ermis is when it needs a message delivered to all the trusted peers of a user. Ermis does this for GeoS when GeoS needs an update sent out or the most up to date coordinate data. As with all communication leaving from or coming to GeoS, GeoS uses the Gateway to communicate with Ermis. There is also indirect communication between GeoS and Ermis when GeoS makes a request of the Gateway and the Gateway needs information on the peer-to-peer network.



① The user sends a request. The Gateway checks the ACL for the user in the request.
② The request is allowed according to the user's ACL. GeoS receives the request and begins searching its graph.
③ GeoS finds nodes A, C, and D in the graph. A does not trust the peer and is only one hop away from B, so B sends a request for A to its Gateway. The Gateway will take care of routing the message to the correct peer.
④ The Gateway for A's trusted peer checks its ACL. The request is allowed according to the user's ACL.
⑤ GeoS for A's trusted peer finds E and B.
⑥ GeoS for A's trusted peer replies with the results to its Gateway to be returned to the requestor.
⑦ The Gateway for A's trusted peer returns the result to the Gateway for B's trusted peer.
⑧ The Gateway for B's trusted peer returns the result to its instance of GeoS.
⑨ GeoS combines the results and removes duplicates and itself. It returns these results to its Gateway.
⑩ The Gateway for A's trusted peer returns the results to the user.

Figure 2. GeoS and the Gateway Communicating.

3.2.3 GeoS Design

GeoS uses a directed multigraph where each edge has a weight and a label for holding the social data of its users (detailed in Section 3.2.3.1). To reduce the amount of network traffic throughout the distributed system and the energy costs on mobile devices that are running social sensors, social sensors may send input to only one trusted peer. To keep the social data consistent amongst the trusted peers of a user, peers periodically send updates, as described in Section 3.2.3.2. The system decides when to send updates by using an adaptive threshold function (presented in Section 3.2.3.3) that is designed to be fair with respect to the amount of network traffic created for each user.

3.2.3.1 Data Structure

GeoS uses a weighted, directed multigraph to represent the social data for the users. Each user can have multiple parallel edges connecting them to another user, with each edge having a different tag associated with it. A tag usually represents a social activity shared between the two users. For example, "hiking", "school", "work", and "movies" are possible tags. The social sensors will provide activity tags. There are some tags that will most likely be explicitly declared, such as "family". Social sensors to tags is a many to many relationship: Each social sensor can send input of any tag and each tag can be used to label an input sent by any social sensor. A social sensor is not restricted to using a single tag. However, because of this lack of a restriction, when social sensors send input for a particular edge, they must take into account the weight currently on the edge in case another social sensor has changed it.

Each edge has a weight associated with it that represents how strongly that activity relates the two users. The weight is a real number between zero and one, inclusively, and varies depending on how frequently the two users share the activity.

We chose to represent the social graph as a directed graph because in sociology it is generally accepted that "ties are usually asymmetrically reciprocal, differing in content and intensity" [Wel88]. In addition, making the graph directed enhances privacy without requiring extra work by ensuring that if user A attempts to artificially raise its relationship strength with user B by sending a massive number of interactions, it will not affect the strength of the relationship user B has with user A. If the graph were undirected, the system would have to detect that a bulk of unidirectional interactions, or "spam", were received and eliminate them to keep users from artificially increasing the mutual relationship strength between them and other users. Detecting spam with an undirected graph would require GeoS to record the amount of interaction reported by each user for every user they reportedly interacted with and if the amount of interaction between two users

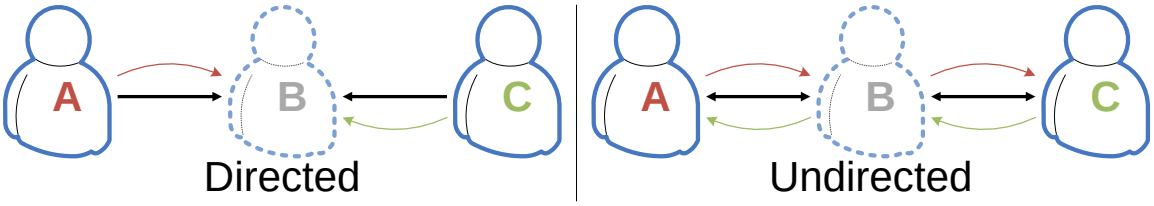


Figure 3. Directed Graph Versus Undirected Graph for an Inference Request of at Least Path Length Two.

is decidedly one-way, the sender of the spam can be removed. While this certainly does work, recording the one-way interactions uses extra resources and is similar to making the graph directed. In addition, GeoS would remove spam after the system discovered it instead of spam never existing in the first place. Removing duplicate or inconsistent input in an undirected graph requires a bit more thought.

Using a directed graph also simplifies data management in a distributed environment by removing the chance of an input being applied twice: in an undirected graph, if both parties that are part of an interaction send a social input, the input will be applied twice if the two users share a trusted peer. In addition, if the two applications or devices that sent the input sent different values, an inconsistency would occur. These two facts would require GeoS or the social sensors to do extra work. For example, the social sensors of the interacting users might have to communicate with each other to determine which sensor is sending the input for the two users and to verify that an agreed upon weight is being sent.

The biggest cost to using the directed graph is that, as seen in Figure 3, some requests that require searching 2 hops or greater away will require a remote request when they would not have to in an undirected graph. In the example, A wants information about C through B and C wants information about A through B, but B has not declared the trusted peer of A and C as its trusted peer. In the undirected version of the example, A and C could get the required information locally.

Other social data such as when the user last interacted with each other user and the current coordinate location on file for the user is also stored with the user's social graph. We refer to the entirety of a user's social data as their *social state data file*, or SSDF. Data on the amount of traffic the user receives from input (whose use shall be described in Section 3.2.3.3) is also transferred with the SSDF, even though it may not be considered social data. It is easier to include this data with the SSDF rather than send it separately. In a simple graph with 1000 nodes and one edge type that was randomly generated using the Nearest Neighbor algorithm described in Section 5.1, the mean size of a single user's SSDF was 1062 bytes. The user's current coordinates are included in a transferred SSDF only in case of peer failure. Because we expect location

to change frequently, a peer will query all trusted peers of a user for the latest coordinate sent by the user anytime it needs location data.

Table 1. Stored Social Data.

Data	Description	Entry in Transferred SSDF
Social Graph		
Graph	Directed multigraph data structure that can be used to find edges and nodes in the graph	Built using second
Weights	After an edge has been found in the graph, its weight can be looked up in this data structure	Second
Timestamps	Contains the time since two given users last interacted	Third
Start Time	Used with timestamps to calculate server time	N/A
Update Preparation		
Graph Delta	Amount weights have changed since last update sent	N/A
Average Input	Average number and weight of input to calculate adaptive threshold	Fourth
Coordinates		
Coordinates	Latitude, longitude, and altitude	Fifth
Coordinate Timestamp	Timestamp set by input device that is used to make sure most up-to-date coordinate is used	

3.2.3.2 Updates

Each GeoS peer tracks the input it received for each user. If the total weight increase for a user passes a certain threshold, GeoS will send an update to Ermis via the Gateway for Ermis to distribute an update to all other trusted peers of the user. The update is in the format of an SSDF and includes all outbound edges from the user, and the weight for each of those edges. By including all edges and weights for the user, nodes who might have missed previous updates are updated. The local Gateway receives this update, encrypts it with the user's key, and hands it over to Ermis for delivery to all of the user's trusted peers. Upon receiving it, the Gateway of each trusted peer decrypts the update and delivers it to the local GeoS.

Each trusted peer of a user has its own local copy of the social graph for that user. When a trusted peer of a user receives an input that another trusted peer of that user does not, their local copies are inconsistent. A goal of GeoS is to keep each local copy of a user's social graph in a near consistent state across the distributed system while keeping network traffic low. Sending input for a user to every trusted peer of that user would keep the graphs consistent as long as no peer missed an input, but it would also cause a lot of traffic across the network each time a social sensor sent input. Using the periodic update system allows each graph to be in a near consistent state while only occasionally creating extra traffic to each trusted peer.

When a user causes an update to be sent out, each copy of their social graph (on all of their trusted peers) has its "base state" brought to the same point. Input that is received, but not yet released in an update is kept separate from the data in the base state and can be imagined as being on top of it. As seen in Figure 4, when an update is received by a peer, the current social graph for the user is replaced by the new base state in the update, and then any unreleased input is applied on top of it. Two graphs are equivalent if they are at the same base state and there is no unreleased input. When a peer sends an update, the Gateway receives a local copy of the SSDF, which it keeps encrypted. When a new GeoS node comes online, it will be sent one of these encrypted local copies from a Gateway to "bootstrap" it. The SSDF brings the new node to the same base state as all the other peers.

3.2.3.3 Adaptive Threshold Function

Studies have found that the social interaction frequency of people generally takes the form of a power-law distribution. In other words, some users are very active socially while others are much less active [GWH07]. Because there are generally such large variations in how socially active people are, we expect that different users will have a large divergence in the amount of input their social sensors send to GeoS. A static threshold for deciding when GeoS should send updates could leave users in an out-of-date state for a long time if their social sensors report little social interaction. Consequently, we chose an adaptive threshold for each user.

This adaptive threshold function works by recording the average weight of input messages the peer receives for the user along with the average number of inputs that peer receives per hour for the user. It then uses this data to try to cause the user to send out on average X updates an hour.

The threshold is set to: (<average input weight> $*$ <average number of input an hour>) / <desired average updates an hour>

The desired average updates an hour can be scaled depending on how much traffic is wished for in the network. For our experiments, it was set to three updates per hour. As mentioned in Section 3.2.3.2, the average weight and input count is distributed via the SSDF so that the threshold function on new peers will know how often to send updates. These remote weight and input counts also influence the adaptive threshold of already existing peers, but the local counts influence the function more. The thought behind this is that the accumulated knowledge of how socially active a user is should influence a peer's adaptive threshold function, but not as strongly as its own personal experience. Since a local peer will receive its own counts back in a remote peer's SSDF update, it subtracts its counts from the incoming remote counts before using them. The procedure by which a count is processed when received in an update is given in Figure 5.

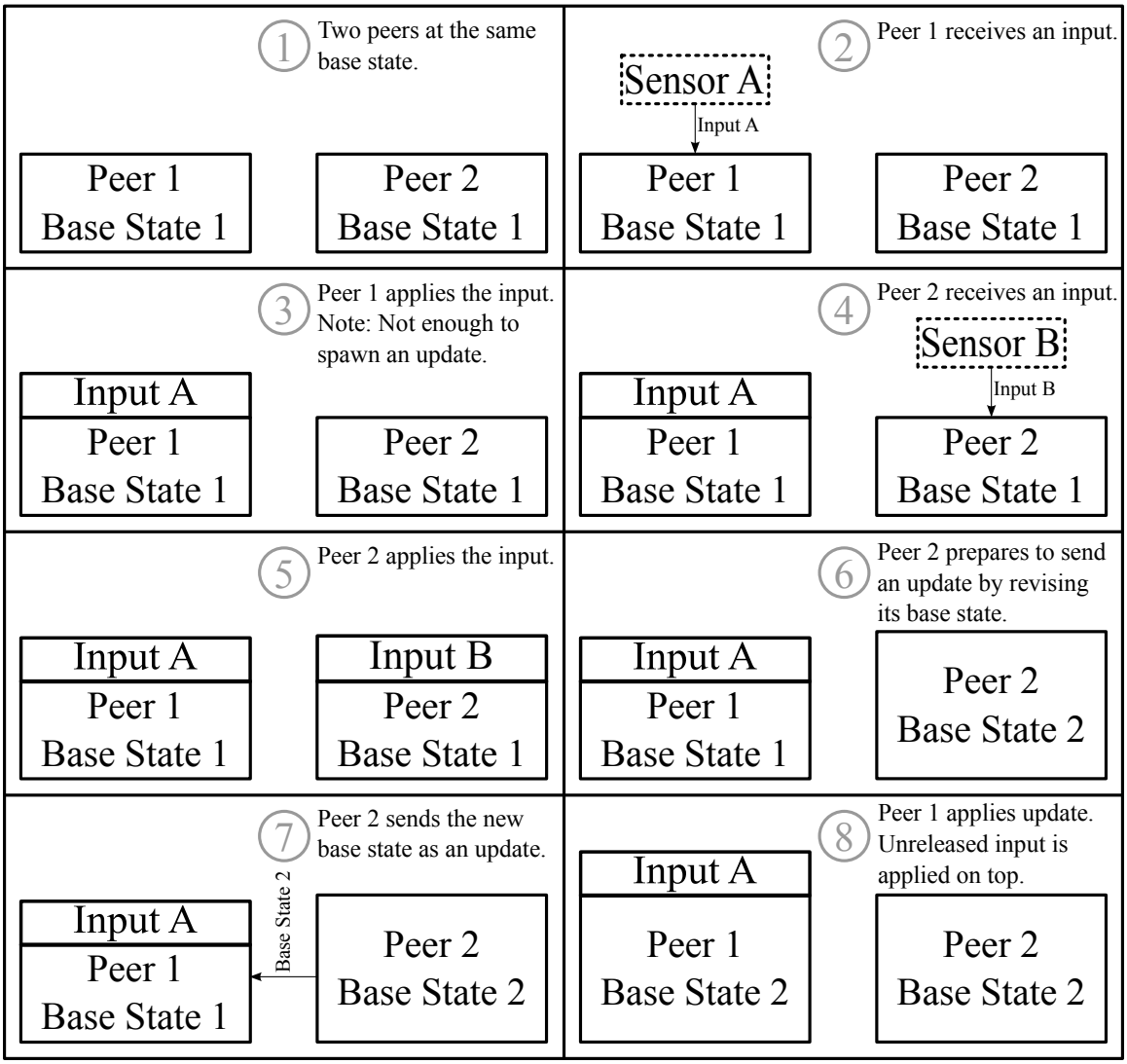


Figure 4. Inputs Being Received and Then Distributed Through an Update.

number of remote inputs $\leftarrow$ received number of remote inputs $-$ number of local inputs;
remote servers total run time $\leftarrow$ received run time $-$ time since local peer was initialized;
remote inputs per second $\leftarrow$ number of remote inputs/remote servers total run time;
local inputs per second $\leftarrow$ number of local inputs/time since local peer was initialized;
inputs per second $\leftarrow$ (remote inputs per second $+ (2 *$ local inputs per second$))/3$;
total input $\leftarrow$ inputs per second $*$ time since local peer was initialized;

Figure 5. Applying Remote Input Information to the Adaptive Threshold Function.

We chose to base the calculations on the average number of updates because it would allow us to easily shape the network traffic that comes from updates being sent. It also allowed for easy calculations since GeoS only needed to keep track of the average volume of input.

3.2.3.4 Example Social Input

Now that all parts of the update process have been explained, the following is an example of the steps input goes through to be applied in GeoS. In this example, two users (A and B) are collocated at a university and their social sensors detect their interaction. The interaction is noticed by an application on their mobile phone that uses Bluetooth to search for other Bluetooth devices. When it finds a new Bluetooth device present, it queries a service that translates devices to GeoS user names. When it determines the other Bluetooth device is a social sensor of another user, it records how long the users are together to verify they were not just passing by. The users stay together for several minutes and so it decides that it must have been a real interaction. The social sensor now uses GPS to find its current location, then uses a reverse geocoding service to determine the place, and finally uses a service to find the tag associated with this place ("school"). With the tag found, it now calculates a weight. It takes into account many factors, such as how long the users were together, how often the owner interacts with others at school, and if the two users are related in some other way.

The social sensor for user A collects data for several hours and then decides it is prepared to send a social input. The input is encrypted on the phone and then sent to the Gateway of a trusted peer of user A. The Gateway decrypts the input and forwards it to GeoS. GeoS applies the input to its social graph. The "school" edge from user A to user B is increased by the calculated weight. The weight of the input is greater than the threshold returned by the adaptive threshold function, and so an update is released by the peer.

GeoS compiles together the user's SSDF and gives it to the Gateway, telling it to have Ermis multicast the update to all trusted peers of the user. The Gateway encrypts the SSDF and gives it to Ermis. Ermis multicasts the update to the Gateways of all the trusted peers of the user. These Gateways verify the update is allowed by their ACLs, decrypt the update, and then pass it to GeoS. Each of the GeoS instances replaces its SSDF with the one in the update and then applies on top of that any input it has received, but not yet released in an update.

3.2.3.5 Aging

When people do not interact for an extended period of time, the social bonds between them begin to slowly degrade. Consequently, unless GeoS is used for a short-term study, it will need some type of "aging" function to mimic this slow degradation of social ties due to lack of interaction.

Our current approach is to reduce the weight between users by 10% each week they do not interact. This results in a very slow degradation and relationships never fully end (but eventually become very weak.) This seems like a passable solution since it gives users plenty of time to reinitiate contact. However, we have no way of knowing whether this, or a variation of it, is the correct solution from a sociological view. In the following, we propose a few possible solutions that could solve the aging problem.

Since GeoS has support for input to have a negative value, one alternative is for an outside application to handle the specifics of aging. Leaving the aging to an outside application would be a good solution because it is a modular approach. This way, anyone can work on an aging function. Note that although GeoS supports input that has a negative value, this is currently the only case where we expect it.

Another approach to the aging problem is to have users manually set how much time they want to go by before aging occurs and how strongly the aging function should affect the relationship when it does. In a similar fashion, instead of setting an amount of time before aging should occur, a user could manually go into an application and degrade the connection whenever they choose. For example, if they notice someone they no longer interact with is showing up in the results for inference requests, they could enter the degradation application and reduce the strength of the connection they have with the other user. If users must degrade relation strengths themselves, the system gives them full control over their social graph and the responsibility is taken away from GeoS or an application. If an application handled aging, a mistake or miscalculation in the application could ruin a great deal of social data.

However, requiring the user to do extra work is one of the worst problems of this approach since we designed GeoS to require very little user intervention. Also, users may have a difficult time determining what values to use and end up disrupting their social graph.

3.2.3.6 Message Formats

The format of calls to GeoS is similar in appearance to typical programming function calls. The format of these calls can be found in Table 2.

3.3 Social Inference Algorithms

GeoS includes several social inference functions to allow users' applications to get social information easily. By providing an API that includes inference functions, developers are able to easily query GeoS for social information. The social inference functions provided with GeoS are relatively simple, and we expect more complicated social inferences to be developed in the future using the default ones as a base.

Table 2. Format of Messages in GeoS.

Communication With Users' Applications

User input		
soc_input(ego, alter, tag, weight)	Returns:	Void
geo_input(ego, alter, longitude, timestamp)	Returns:	Void
Requests		
SocS(ego, alter, request_num)e	Returns:	Real number
neighborhood(ego, tag, min_weight, hops, request_num)	Returns:	List of users who met specifications
top_friends(ego, tag, num_top, request_num)	Returns:	List of up to num_top users
friend_test(ego, alter, tag, min_weight, request_num)	Returns:	Boolean
get_coordinate(ego, request_num)	Returns:	Latitude and longitude
proximity_friends(ego, tag, min_weight, hops, distance, request_num)	Returns:	List of users who met specifications

Communication Within Prometheus

SSDF Distribution		
initialize(SSDF)	Returns:	Void
mass_update(update)	Returns:	Void
GeoS-to-GeoS Communication		
get_coordinate(ego, replyUser)	Returns:	Latitude, longitude
get_coordinate_secondary(ego, replyIP, replyUser, request_num)	Returns:	Latitude, longitude
get_coordinate_reply(latitude, longitude, timestamp, request_num)	Returns:	Void
neighborhood_secondary(ego, tag, min_weight, hops, replyIP, replyUser, request_num, hopstravelled, user_chain)	Note:	Results in neighborhood_reply() call (with encryption)
neighborhood_reply(list_of_users, request_num)	Returns:	Void
SocS_secondary(ego, alter, replyIP, replyUser, request_num)	Note:	Results in SocS_reply() call
SocS_reply(social_strength, request_num)	Returns:	Void
Outbound Messages to Other Components		
ermis|127.0.0.1|user|user|scribe update_ssdf ego 0 enc->SSDF	Note:	Results in mass_update() call
gateway|127.0.0.1|||log_ssdf ego SSDF	Note:	Creates local, encrypted copy of SSDF

If application developers build future social inference functions from the composition of the basic provided ones, no changes have to be made to GeoS. However, while there is a gain of modularity by doing it this way, there will be a large amount of back and forth communication compared to if the new function was implemented within GeoS. One idea to reduce this unneeded communication is to create "user services" that run on the same peer as GeoS, receive complex requests from users, and then automatically take care of submitting a composition of functions. These inference functions were first presented in [AKFI10].

3.3.1 Friend Test

The function *friend_test(ego, userB, α, x)* is a boolean function that checks whether ego is directly connected to $userB$ in the social graph by an edge with label α and with a minimum weight of x. A mobile

phone application can use this function to determine whether an incoming call from a coworker should be let through or silenced on weekends.

3.3.2 Top Friends

The function *top_friends(ego, α, n)* returns a size n set of users in the social graph that are directly connected to ego by an edge with label α, ordered by decreasing weights. An application can use this function, for example, to invite users highly connected with ego to share content related to activity α.

3.3.3 Neighborhood

The function *neighborhood(ego, α, x, radius)* returns the set of users in ego's neighborhood of $radius$ who are connected through social ties of a label α and weight larger than x. The function performs a breadth-first search on the social graph. An application that silences a user's cell phone near coworkers uses this function to infer ego's work neighborhood in the social graph even if not directly connected to the user (i.e., the phone will still be silenced if on a business lunch with a regional manager that the user has not met before).

If the radius is greater than one, GeoS may need to use the Gateway to request other GeoSs (on other peers) to infer social information to complete the inference. This need for a remote GeoS will only happen if the local GeoS is not the trusted peer for a user that is $(radius - 1)$ hops away from ego. The local GeoS will send a *neighborhood_secondary()* request to a remote GeoS of this user for which it is not a trusted peer. The secondary request contains within it the usual details of tag and weight, but also contains how many more hops to search and where to reply to. The secondary request could potentially result in this user also sending off a secondary request of its own. Each peer that gets a secondary request replies back to the sender with the result (even if it was empty). When the original peer finally gets all the replies it is waiting on, it compiles the results (removing duplicates if any), and returns the results. The waiting peer will eventually time out if it does not receive a reply.

The searching of the graph uses a breadth-first search algorithm. The algorithm starts out with a queue that is empty except for ego. The algorithm continues to remove the user in the front of the queue and make it the currently searched user until the queue is empty. Each neighbor of the current user who has an edge that meets the specifications and who has not been visited yet is added to the final results. A user added to the final results is either added to the queue or processed on a remote peer. If the user trusts the peer, the algorithm places the user in the back of the queue; otherwise, it sends off a remote request. GeoS sends off

remote requests right away with a unique request number attached to them. After traversing the graph as far as possible locally, the algorithm waits to receive replies from all requests. It then combines the results from the remote requests, removes duplicates, and gives the results to the Gateway.

```
input  : ego, α, x, radius
output: The set of users in ego's neighborhood of radius who are connected through social ties of a label α and
        weight larger than x
Enqueue(ego);
while queue not empty do
    ego ← Dequeue();
    foreach alter in Neighbors(ego) do
        if alter not visited then
            if α in Edges(ego, alter) then
                if x ≥ Weight(ego, alter, α) then
                    Visit(alter);
                    distance[alter] = distance[ego] + 1;
                    results ⟵ alter;
                    if distance[alter] < radius then
                        Enqueue(alter);
                    end
                end
            end
        end
    end
end
```

Figure 6. Algorithm for Neighborhood(ego, α, x, radius).

The request *proximity_friends(ego, α, x, radius, distance)* is a simple extension of *neighborhood()*. After the results are gathered from *neighborhood()*, it removes any users who are not within a given physical distance. This inference could be used to invite nearby friends to a bar after a football game. To get the location of a user, it simply calls *get_coordinate()*. Since locations are likely to change often, social sensors send location input to only one peer and location information is not included in updates. Instead, whenever *get_coordinate()* is called, it contacts every trusted peer of the user to get the current coordinate the peer has on file for the user along with the timestamp given with that location. Whichever coordinate received has the latest timestamp associated with it is the one used in the distance calculation. The distance between two coordinates is calculated using the spherical law of cosines.

3.3.4 Social Strength

The function *social_strength(ego, alter)* returns a real number between 0 and 1 that quantifies the social strength between ego and $alter$ from ego's perspective. This value is normalized to ego's social ties to ensure that the social strength is less sensitive to the social activity of the users. As shown in Figure 7, the normalized

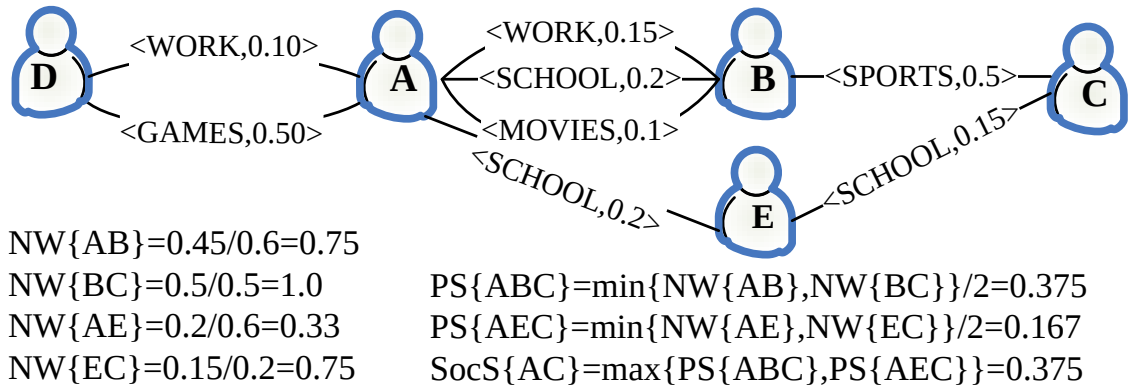


Figure 7. A Social Strength Calculation Example.

weight from A to its adjacent neighbor B (NW{AB}) is the sum of all the weights of the edges from A to B (aggregating over all types of interactions between A and B) divided by the largest of all the sums of weights going from user A to one of its neighbors (D). The path strength from A to C through B (PS{ABC}) is the lowest of all the NW on that path, divided by the length of the path. Finally, the social strength from user A to user C (SocS{AC}) is the largest path strength from A to C.

The social strength request provides users with a quantitative measure on social solidarity [BF01] based on the frequency and intensity of their social interactions. Such a function could be used, for example, to estimate social incentives for resource sharing, such as storage or hosting a service for another user or a community, or the battery-aware BitTorrent example presented above. Social strength can be applied to dyads that may not be directly connected. Moreover, even if a direct tie exists, an indirect path (through common friends) may be stronger [Bia97]. We limit the length of the indirect path that connects two users to 2, using a well-accepted result in sociology known as the "horizon of observability": Friendkin [Fri83] shows that people know about persons up to two social hops away, even though they do not interact directly with them. We interpret this as an indication that it is unlikely for users who are connected by a shortest path of length 3 or longer to have a non-zero social strength between them.

Since the social strength requires searching two hops away, one GeoS peer may not locally have all the required information. This means it will have to use the Gateway to contact GeoSs on other peers as seen in Section 3.2.2. The remote peer will return the normalized weight from the remote user to $userB$, which can be combined with local data to calculate the path strength. The local GeoS will make sure it is possible for a path going through the remote user to result in a larger path strength than the largest path strength found

locally before sending out the remote request. A social strength request from a user A to any other user will result in a maximum number of remote calls equal to the number of neighbors user A has in the social graph.

input : ego, alter
output: A real number between 0 and 1 that quantifies the social strength between ego and alter from ego's perspective

```
Sum(userA, userB):
begin
    foreach tag in Edges(userA, userB) do
        sum ⟵ Weight(userA, userB, tag);
    end
end

MaxSum(userA):
begin
    max ← 0;
    foreach userB in Neighbors(userA) do
        if Sum(userA, userB) > max then
            max ← Sum(userA, userB);
        end
    end
end

NormalizedWeight(userA, userB):
begin
    Sum(userA, userB )/MaxSum(userA );
end

PathStrength(userA, userB, userC):
begin
    if NormalizedWeight(userA, userB ) > NormalizedWeight(userB, userC ) then
        NormalizedWeight(userA, userB )/pathLength;
    else
        NormalizedWeight(userB, userC )/pathLength;
    end
end

Social_Strength(ego, alter):
begin
    max ← 0;
    foreach userB in Neighbors(ego) do
        if userB is alter then
            if NormalizedWeight(ego, userB ) > max then
                max ← NormalizedWeight(ego, userB );
            end
        else
            foreach userC in Neighbors(userB) do
                if userC is alter then
                    if PathStrength(ego, userB, userC ) > max then
                        max ← PathStrength(ego, userB, userC );
                    end
                end
            end
        end
    end
end
```

Figure 8. Algorithm for Social_Strength (Continued).

Chapter 4:

Implementation

GeoS is written in Python and uses some preexisting Python modules. An important such module is NetworkX, which is a graphing module that GeoS uses for handling the directed, multiple parallel edge, graph data structure. Specifically, we use the MultiDiGraph class. The class provides simple to use functions to deal with graph tasks that would be difficult to implement manually using built-in data structures. Adding edges, finding neighbors of a user, and determining the number and type of edges connecting two users are all easily accomplished using this class. The class also provides graph algorithms, such as calculating betweenness centrality, which GeoS does not use currently, but may prove useful for future GeoS algorithms. Some possible uses of these graph algorithms are presented in Chapter 8:. NetworkX was chosen over other graphing libraries, like iGraph, because of how well it is documented and because it is written in Python, which allows for easier testing.

Python dictionaries, also known as associative arrays, are used for holding the weight for each edge along with other edge data like the sum of all edges between two users. GeoS also uses dictionaries for storing the GPS location of each user, and the last time each pair of users interacted. The dictionary data structure is a built-in part of Python and simplifies programming in GeoS by allowing it to use user names as keys.

For remote requests between GeoS nodes, we built a request-handling system. Remote requests are sent any time a peer does not have edge data that it needs to correctly fulfill a request due to it not being a user's trusted peer. Each time a peer sends a remote request, a request number is incrementally generated that will be unique for the local GeoS until twice the maximum integer size requests have been sent. The request number only needs to be unique for the local GeoS because the peer only uses it to identify the result of the remote request when it receives it. After the remote GeoS fulfills the request, it replies with the result and the request number. The local GeoS collects all these replies and then gives them to the function that made the remote requests so that it can finish processing its results. The local GeoS continues waiting for replies until either a timeout value is passed or the request is marked as fulfilled. If a request requires social data more than two hops, it is possible for a peer to be waiting on a remote peer who is waiting on other remote peers.

GeoS is multi-threaded. Connections are listened for with a non-blocking socket because non-blocking sockets are faster than blocking sockets [McM], and the Unix *select()* system call is used for checking when there are incoming connections. Whenever it receives a new connection, the peer spawns a new thread, receives the incoming message, and calls the function that corresponds to the request. We chose threading over using a single thread with asynchronous connections because when a function makes remote requests, it must wait until it receives the results before it can finish processing. It would be much more complicated to call the request function, send out the remote requests, go back into the main receiving loop, have the receiving loop keep checking whether all the remote requests have been fulfilled or timed out while it is still taking new requests, and then go back to finishing the request instead of just running the request in a thread, sending the remote requests, and then putting the request's thread to sleep until all the results are received or the remote requests time out.

GeoS also uses threading to run certain calls on a schedule. One scheduled thread updates the number of requests received each second for logging purposes, another cleans up any remote requests that never received a reply, and finally, a last one runs every fifteen minutes to check if users who have not sent out an update for a long period of time now exceed the threshold needed to send an update. For the latter, the threshold will be lower than last checked due to time passing.

Chapter 5:

Experimental Setup

To test the performance and capabilities of GeoS, we executed several experiments. All testing data were either based on social data or were actual social data to make the tests as close to the real world as possible. Several ideas for creating the social graphs to be used in the tests were considered. For performance tests, the frequency of remote input and request messages were based on probability distribution functions (PDF) from data released by previous research.

5.1 Social Graph Generation

When running experiments on the service, a graph that was based on real social data was desired so that the results of the experiments would be similar to what would be seen in the real world. The graph needed to have many users and multiple parallel edges linking users. Data from multiple sources is very useful for testing the features of GeoS because the more known about the users, the more detailed the graph will be and the closer it will be to what a graph made in GeoS will look like in the real world.

5.1.1 Synthetic Graphs

The next idea on how to create a useful social graph was to generate random graphs with properties similar to a social graph. This would allow us to generate a graph that had exactly the desired number of nodes. A modified version of the Nearest Neighbor algorithm was chosen for generating the graphs after seeing that Sala et al. found the algorithm consistently produced graphs with properties similar to real social graphs [SCW+10]. The Nearest Neighbor algorithm was originally presented in [Váz03] and Sala et al. modified the original algorithm in order to bring the properties of graphs it generates closer to those seen in real social graphs. The only feature graphs generated with this algorithm lack is multiple parallel edges between users, but they are still useful for testing. For our tests, we generated a graph with 1000 nodes using the algorithm. Its properties can be seen in Table 3.

We have considered that the algorithm can build a social multigraph while keeping the social graph properties. We propose that each node in the graph can be randomly given a set of real numbers between

Table 3. Properties of Graph Generated Using Nearest Neighbor Algorithm.

Nodes:	1000
Edges:	5784
Average Clustering Coefficient:	0.32
Average Clustering Coefficient of a random graph:	0.12
Average Eccentricity:	7.64
Power-law Distribution:	1.39
Average Shortest Path Length:	3.58
Average Shortest Path of a Random Graph:	3.94
Average Degree:	11.57

0 and 1, where each value represents how interested they are in a subject (represented by an edge tag). Whenever the algorithm connects a pair of users by an edge, two directed edges are created between them for each tag. The weights of the edges are randomly generated between 0 and their interest in the tag's subject. In this way, if the graph was reduced to just one edge type, it would still be a social graph, but some edges leaving users will have such a low weight that they are almost nonexistent.

5.1.2 NJIT Social Graph

To verify GeoS also works correctly on real social data, a data set that included social information from two sources was tested. The social data consist of one-month of Bluetooth collocation data and Facebook friend lists for a set of 104 students at the New Jersey Institute of Technology (NJIT). The data collection took place on this medium-sized urban campus, and the subjects were representative of the various majors offered on campus, with 75% undergraduates and 28% women.

Similar to the Reality Mining traces [EP06], smart phones were distributed to students and an application quietly recorded the Bluetooth addresses of nearby devices and periodically transmitted them to a server. Given that the sample size was small compared to the university population (104 randomly selected out of 9000 students) and that many students are commuters, the trace data is sparse. For example, about half the subjects reported less than 24 hours of data for the entire month, and only 17% of the scans detected other Bluetooth devices in proximity. The typical user provided a few hours of data per day, especially during the weekdays. The same set of subjects installed a Facebook application to participate in a survey, and gave permission for their friend lists to be collected when they installed this application.

The recorded social data were separated into three categories: pairs of users who were collocated for a total of at least 45 minutes (at least 15 of those minutes were consecutive) during the study ($CL.45$), pairs of users who were collocated for a total of at least 90 minutes (at least 15 of those minutes were consecutive)

Table 4. Properties of NJIT Graph.

Nodes:	100
Edges:	469
Average Clustering Coefficient:	0.37
Average Clustering Coefficient of a random graph:	0.09
Average Eccentricity:	4.47
Power-law Distribution:	1.37
Average Shortest Path Length:	2.50
Average Shortest Path of a Random Graph:	2.98
Average Degree:	9.38

during the study ($CL.90$), and pairs of users who declared each other friends on Facebook (FB). Each pair in the $CL.90$ data was also in the $CL.45$ data. There were 87 users in the FB data, 94 in the $CL.45$ data, and 81 in the $CL.90$ data. There were a total of 100 unique users in the three data sets with 81 appearing in both the Facebook and collocation data.

We created a multigraph based on the NJIT data where edges were labeled "facebook" with weight 0.1 for the FB data, "collocation" with weight 0.2 for the $CL.45$ data, and "collocation" with weight 0.3 for the $CL.90$ data. A search of the graph starting from a node in the periphery can be seen in Figure 9 and the graph's properties can be found in Table 4. Black edges are "collocation" edges with weight 0.3, gray edges are "collocation" edges with weight 0.2, and dotted edges are "facebook" edges with weight 0.1.

5.2 Generation of Workloads

For experimentation, realistic workloads were created for testing the real-world performance of the service. There have been several studies into how often people communicate through electronic sources like Twitter and Facebook [WBS$^+$09, KGA08]). We combined this information with details on the inward and outward connections of the nodes in the graph to create the various workloads. Our assumption (based on findings from [WBS$^+$09]) is that the amount and ratio of connections a user has is representative of a user's social activity level.

5.2.1 Input Based on Facebook

We emulate a Facebook social sensor based on a Facebook trace analysis [WBS$^+$09]. We extract workload characteristics such as the probability distribution function and the rate of user activity.

The sensor ranks users into seven groups based on their social degree (in and out degree). Using the cumulative distribution function from the Facebook study (Figure 8 in [WBS$^+$09]), we map each group into

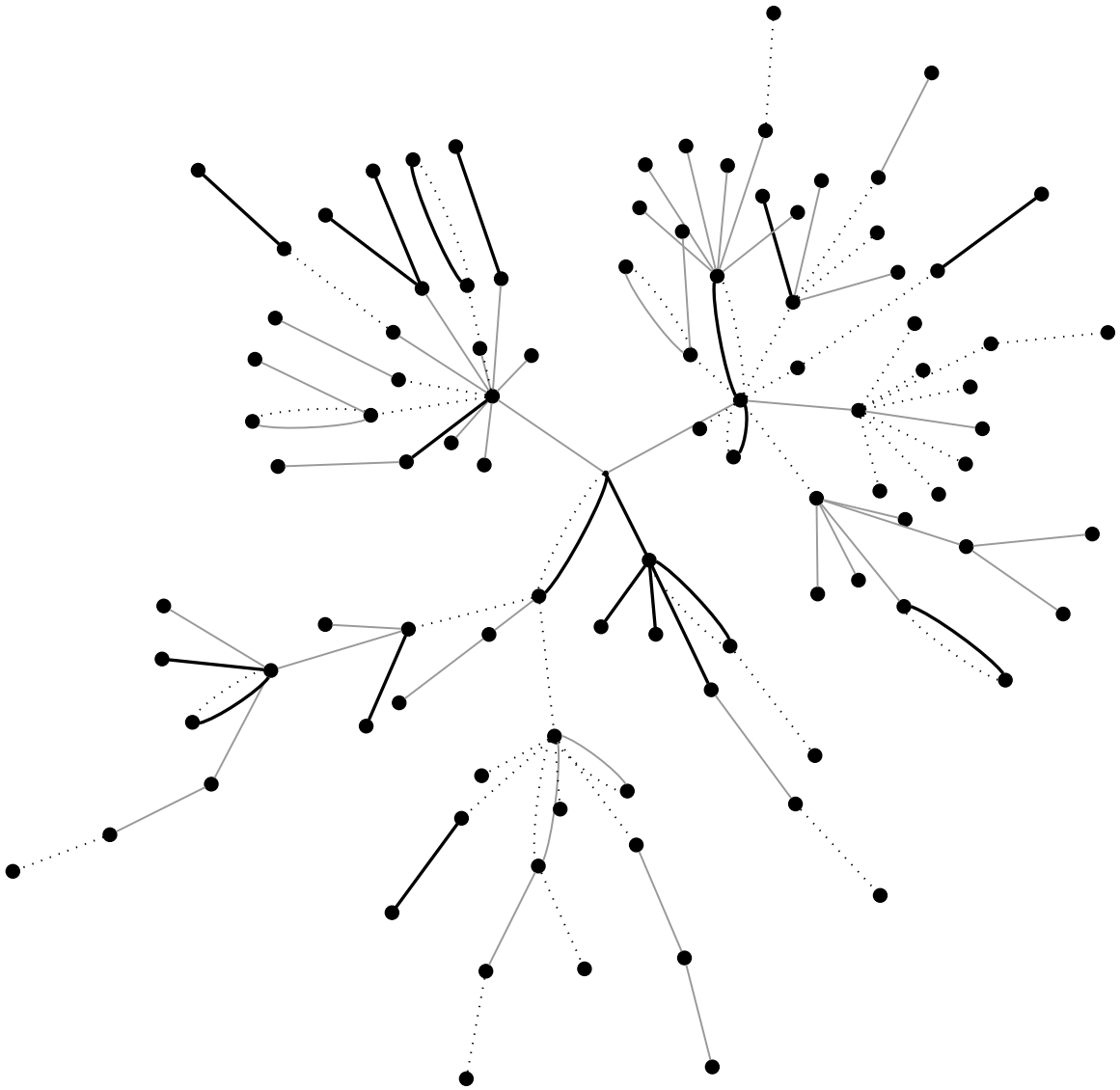


Figure 9. Traversal of NJIT Graph.

a particular probability class, as shown in Table 5. Once the group is selected, a user who has not been selected yet is picked to be the source of the input, S. For the destination of the input, another user is picked, from the available social connections of S. The weight x of the input is kept constant to a small value, across all inputs for all users. Since the users are picked based on their social degree, users with higher social degree probabilistically will produce more input. We also use the same studys observation that users produce an average of about one Facebook interaction per user per day, which translates into one Facebook interaction every 90 seconds for a user base of 1000 users.

Table 5. Probability Distribution Function for Input Using Facebook Data.

% Social Degree Rank (desc. order)	% of 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

5.2.2 Requests Based on Twitter

In order to emulate the workload produced by applications that need neighborhood inferences, we used an analysis on Twitter traces [KGA08]. Intuitively, we associate a tweet in Twitter with a neighborhood request (centered at the leader of the tweet) in GeoS. From previous Twitter trace analysis, we extract the probability distribution function and the rate of submitted tweets.

We ranked users into three groups based on their social degree ratio of incoming to outgoing edges. Using the CDF from the Twitter study (Figure 4 in [KGA08]), each group is mapped onto a particular probability to be selected and submit a neighborhood request as show in Table 6. Once the group is selected, a user who has not been selected yet is picked to be the source of the request. The number of hops for the request can be randomly picked from 1, 2 or 3 hops (the average path length in a social network is typically low, so we assume that an inference of 3 social hops leads to a large enough graph coverage for such a request). From the same study, we assume about 1 neighborhood inference request every 12 seconds interval for a trace-set of 1000 users.

Table 6. Probability Distribution Function for the Neighborhood Request Using Twitter Data.

Followers/Following (In/Out Degree)	Average Number of Tweets per day	Group assigned to user	Probability to choose user (PDF)
100-1000	86	1	0.53
10-100	50	2	0.31
0-10	25	3	0.16

5.2.3 Requests Based on BitTorrent

The social strength between two users may be needed as a measure to infer trust between two users. For example, in a battery-aware BitTorrent application [KBI09] on mobile devices, users may rely on social incentives to be allowed to temporarily "free ride" the system when low on battery. Their social contacts will check their social strength with the leecher who is low on battery to see if they want to allow it access to data.

We emulate requests for social strength inference based on an analysis of BitTorrent traces [GCX+05] from which we again extract the probability distribution function and rate of submitted requests. Two users are randomly selected as the source and destination of the social strength inference request. The source is also associated with a number of requests to submit throughout the experiment, using the probability distribution function shown in Table 7. We assume about one social strength inference request every 12 seconds interval for a traceset of 1000 users.

Table 7. Probability Distribution Function for the Social Strength Request Using BitTorrent Data.

Number of requests to submit	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

5.3 Testbed

We executed all tests on some form of Linux-based computer. The experiments meant to test the performance of GeoS were done using ten quad-core computers with 4GB of RAM running on a LAN. This was to reduce the possibility of network delays. The experiments that tested the Prometheus architecture as a whole were carried out on ten PlanetLab machines. The specifics of these machines are unknown, but we tried to spread the machines to different locations around the world. Three were in North America, three

were in Europe, three were in Asia and one was in South America. A copy of the application that sent and received requests was on each peer. The source and destination of each request was randomly chosen. Performance tests continually sent neighborhood and social strength requests along with social input (which caused updates to be sent.)

We mapped users onto peers using either a uniform or a non-uniform distribution. With either distribution, we kept the number of users on each peer at a constant. In the uniform distribution, the only concern was giving each peer around the same number of total edges. This resulted in an almost random distribution from the viewpoint of the social graph. The non-uniform distribution instead uses the social graph to guide the placement of users. It works by first picking the ten users with the highest degree and assigning each one to a peer. Whenever a user is assigned to a peer, all of its neighbors are sorted by their degree and added to the end of the queue for the peer. Each peer takes turns removing users from its queue until an unassigned user is removed. This last removed user is then assigned to the peer. Any user who is removed, but not assigned is put in a secondary queue. If the main assignment queue empties, the first user in the secondary queue is removed and its neighbors are added to the assignment queue. When assigning users to peers, assigning users close to the first user is the highest priority. The next highest priority is to assign users with the highest number of edges. The overall goal was for peers to have a similar number of total edges while keeping socially connected users on the same peer. In our test using a randomly generated graph, the largest number of edges on one peer was 437, the least number on one peer was 338, and the mean number of edges on a peer was 394.4.

Chapter 6:

Experimental Evaluation

We performed three classes of experimentation. First, we tested the capabilities of GeoS using some scenarios the service may be used for in the future. Second, we tested the performance of the service. Finally, we analyzed the performance of the entire Prometheus architecture for any observations that can be made about GeoS as a part of it.

6.1 Performance Metrics

There are two main metrics on which the performance of GeoS is judged. In the capabilities test, the metric is simply a qualitative one. For example, do the results follow what would be expected for the graph? For the performance tests, the time to fulfill a request is evaluated.

6.2 Experimental Results

We present the following experimental results. The experiments were conducted using either a graph with 1000 nodes randomly generated using a modified version of the Nearest Neighbor algorithm [SCW+10] or a real social graph built using collocation and Facebook traces from NJIT. The tests were administered on either ten PCs on a LAN cluster or ten PlanetLab machines spread throughout the world.

6.2.1 GeoS Capabilities

We first demonstrate how a social application could exploit the GeoS functionalities and the use of the weighted multigraph by presenting the following toy scenario that uses the NJIT data set.

Students who attend a particular university together are recorded in GeoS as connected by a collocation edge with different weights (0.2 for 45 minutes of collocation and 0.3 for 90 minutes of collocation) reported by their Bluetooth social sensor. They may also be Facebook friends using information reported by a Facebook social sensor.

Table 8. Mean Number of Contacts Returned by a Neighborhood Request Submitted by Ego (for Various Combinations of Edges and Weights) With Mean Social Strength Between Ego and the Contacts Within Two Hops in Parentheses.

Social Hops	1	2	3	4	5	6
FB	3(0.60)	11(0.39)	23	32	36	37
CL.45	8(0.71)	38(0.35)	72	84	87	87
CL.90	3(0.95)	8(0.62)	17	29	40	49
CL.45 or FB	9(0.64)	43(0.33)	78	90	92	92
CL.45 and FB	1(0.92)	5(0.53)	17	27	31	32
CL.90 or FB	5(0.73)	17(0.44)	34	49	60	67
CL.90 and FB	1(1.0)	2(0.77)	6	12	17	19

Alice has acquired several tickets to the sold out football game and does not want any of the tickets to go to waste, even if she does not personally know everyone who uses one of her tickets. Since anyone who uses her tickets needs to attend with her, she wants to pick those with which she has the closest relationship. To aid her in deciding who to give the tickets to, she uses GeoS to assess the social strength of her connection with students whom she is connected with directly or indirectly. First, she submits a neighborhood request for a given number of social hops using particular types of edges and combinations ("collocation" with a weight of 0.2 or 0.3 and "facebook" with a weight of 0.1), to retrieve the students connected directly or indirectly with her (Table 8). When she decides to be strict and find only the students who were collocated with her the longest and are Facebook friends with her, much fewer students are returned. Depending on where Alice is in the social graph, a mean of one (or two) students within one (or two) hops are returned, thus making the search focused and useful. Next, she requests from GeoS the social strength between her and each of the students returned by the previous neighborhood request. All she has to do is contact the students with the highest social strength (1.0), even 2 social hops away, to invite them to the game.

GeoS was able to simplify Alice's task of finding the students with which she has the strongest relationship. It is intuitive to see that the users she had the strongest weight with from the neighborhood request were the same ones with which she had the strongest social strength. One exception is that following two hop Facebook edges (of weight 0.1) results in a higher social strength than following two hop collocation edges of minimum weight 0.2. This is seen because there are less Facebook edges than collocation edges, which results in there being a higher probability of a Facebook and collocation edge appearing between two users if there is already a Facebook edge between them.

As the restrictions on edges are lowered, the number of users returned by the neighborhood request increases and the mean social strength of those users decreases. Restricting the edges that can be traversed makes many nodes unreachable, which is especially interesting since the two types of edges used different

social sensors. If this test had been done using only Facebook or collocation data, many social connections would be lost. This observation fits with the claim that only considering one source of social information can result in a loss of important details.

6.2.2 GeoS Performance

The following data are from an experiment executed on the LAN cluster and show the performance of GeoS. Because the LAN has very few network delays and because these values were recorded from within GeoS, the data focuses almost solely on the performance of GeoS. During the test, a total of 1,855,737 messages were received by the ten peers running GeoS. There were an average of 8.1 messages received per second and the test continued for a little more than six and a half hours. In the following figures, the Y axis represents how long it took to fulfill a request and the X axis represents the order of incoming messages. The time measured is from when GeoS receives the request until GeoS sends the result to the Gateway.

The results from the neighborhood request will be shown first. The performance results will start with requests that were processed completely locally, followed by one network hop requests (a remote peer had to be contacted for information), and go up to requests that required two network hops (a remote peer had to be contacted for information and in turn, the remote peer had to contact a second remote peer).

Figure 10 shows that the requests which were handled locally were completed very quickly. 99% of local requests are completed in less than a millisecond. The mean run time was 0.12 milliseconds, the median run time was 0.11 seconds, and the maximum time a request took was 3.86 milliseconds.

Next, Figure 11 shows some interesting information on requests that needed to contact a remote peer. The first thing to notice is that there is a spike at the beginning that is an order of magnitude larger than the rest of the results. This is caused by Ermis first finding which peers are online and which ones are the trusted peers of the requested user. After this initial spike, the running times are low due to Ermis caching which peers are online. Even though there are many new delays added, such as network transfer and the Gateway encrypting results from the remote peer, the mean run time of a request is 0.18 seconds and the median is 0.13 seconds. Over 90% of the requests are completed in less than a quarter of a second.

Figure 12 shows neighborhood requests where a remote peer is contacted indirectly. The initial spike is seen again, but this time it is even taller. The few lines that reach all the way to 30 seconds means that a GeoS peer sent out a remote request to a remote peer, but never heard anything back. For testing purposes, GeoS has a very large timeout value of 15 seconds times the maximum number of network hops this remote request could end up taking. These 30 second delays are only seen at the very beginning and only a couple of

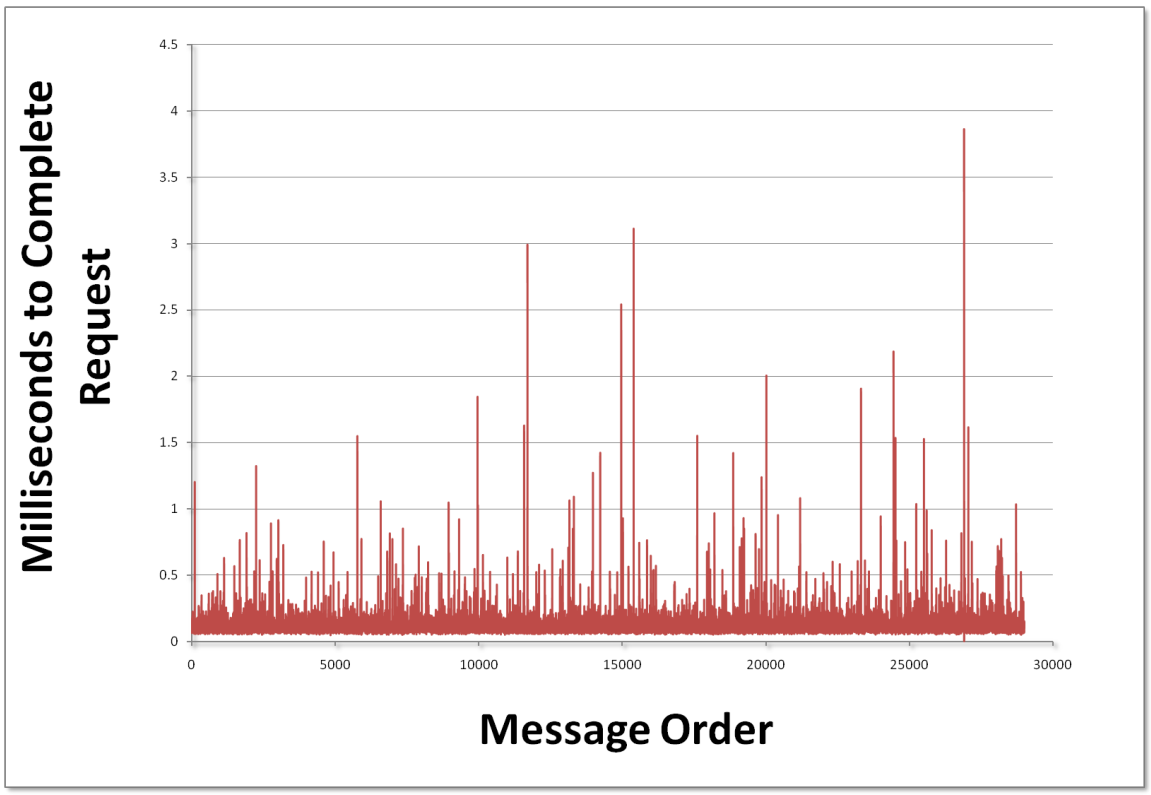


Figure 10. Performance Results of Neighborhood Requests that Were Computed Locally.

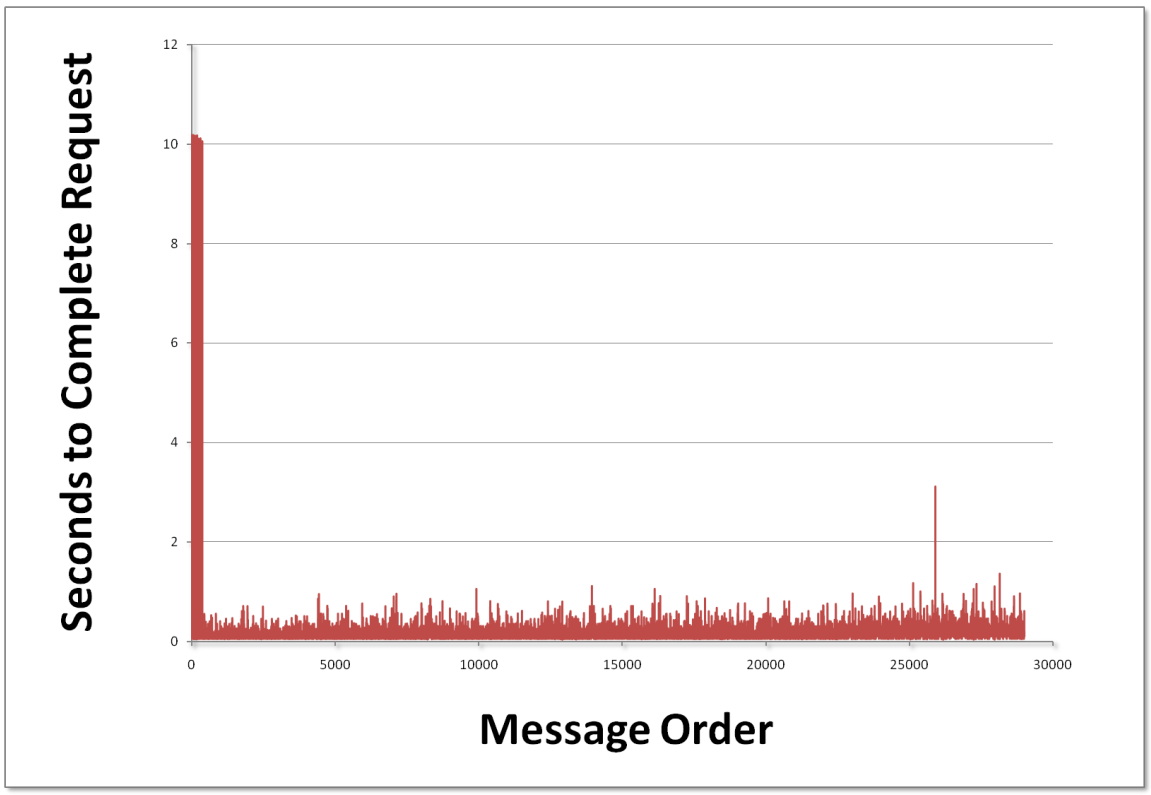


Figure 11. Performance Results of Neighborhood Requests that Were Computed Using Direct Remote Peers.

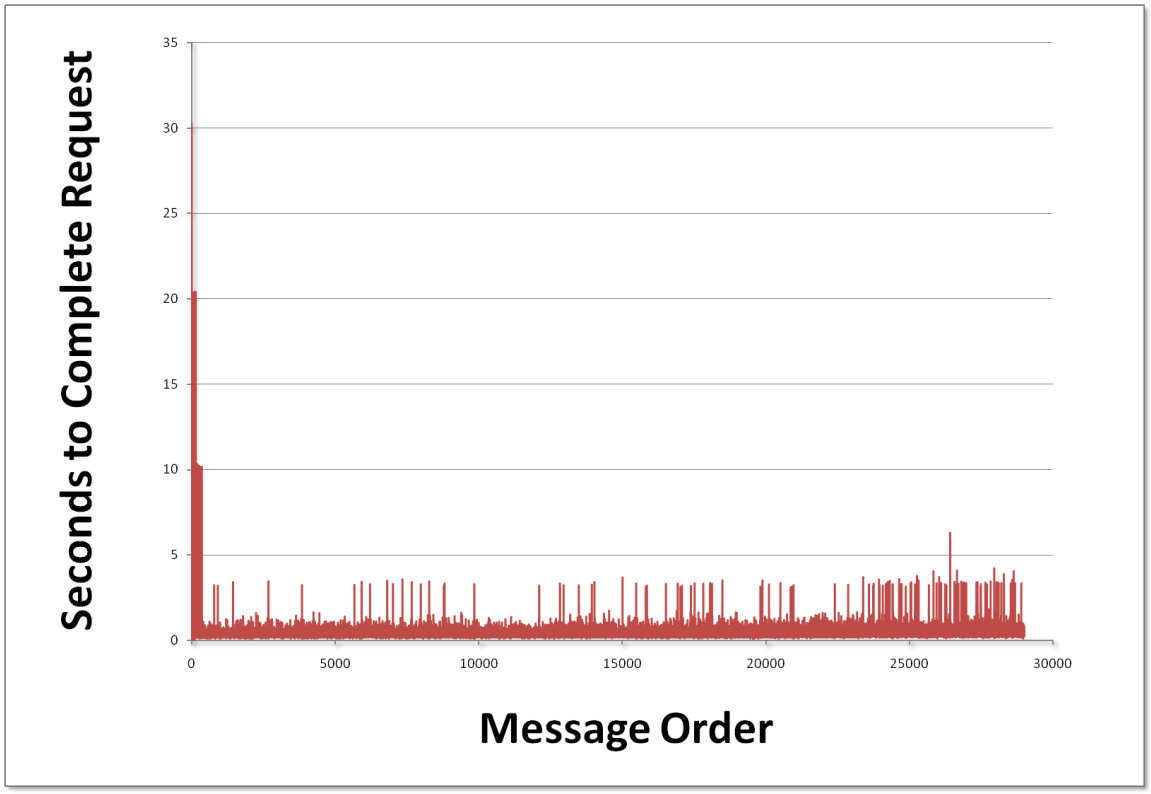


Figure 12. Performance Results of Neighborhood Requests that Were Computed Using Indirect Remote Peers.

times; this means that a few requests failed at the beginning of the test when a peer or service was not ready to fulfill them. The 20 second spikes show that for a three hop request, Ermis waited for ten seconds at the first network hop and then waited ten seconds again at the second network hop. In Figure 13, we cut off the vertical part of Figure 12 that is above seven seconds in order to zoom in on the results after the initial spike. While the run times are not quite as fast as some seen earlier, 97% are still finished in less than a second. The mean run time is 0.45 seconds and the median is 0.31 seconds. The median is a little more than double that seen in the two hop request, which indicates that there is no major delay added even though many more remote requests are being spawned.

Finally, Figure 14 shows the performance results of social strength requests while Figure 15 cuts off the vertical part of the graph above one second to zoom in on the results below a second. The social strength performance results are very similar to those seen in the second neighborhood results, which makes sense since it may contact remote peers of connected users as well. It should be noted that social strength requests do not require any remote calls if an indirect connection could not be as strong as one already found in the

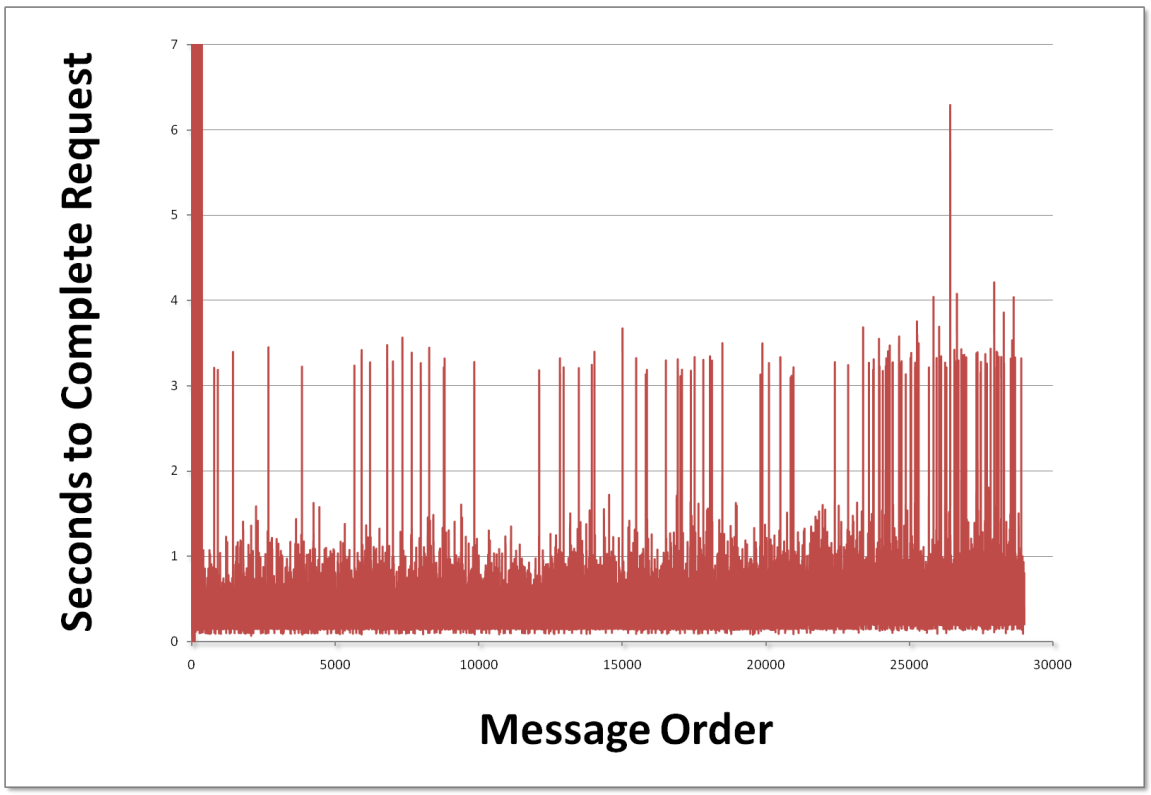


Figure 13. Neighborhood Results Zoomed-In on Seven Seconds and Below.

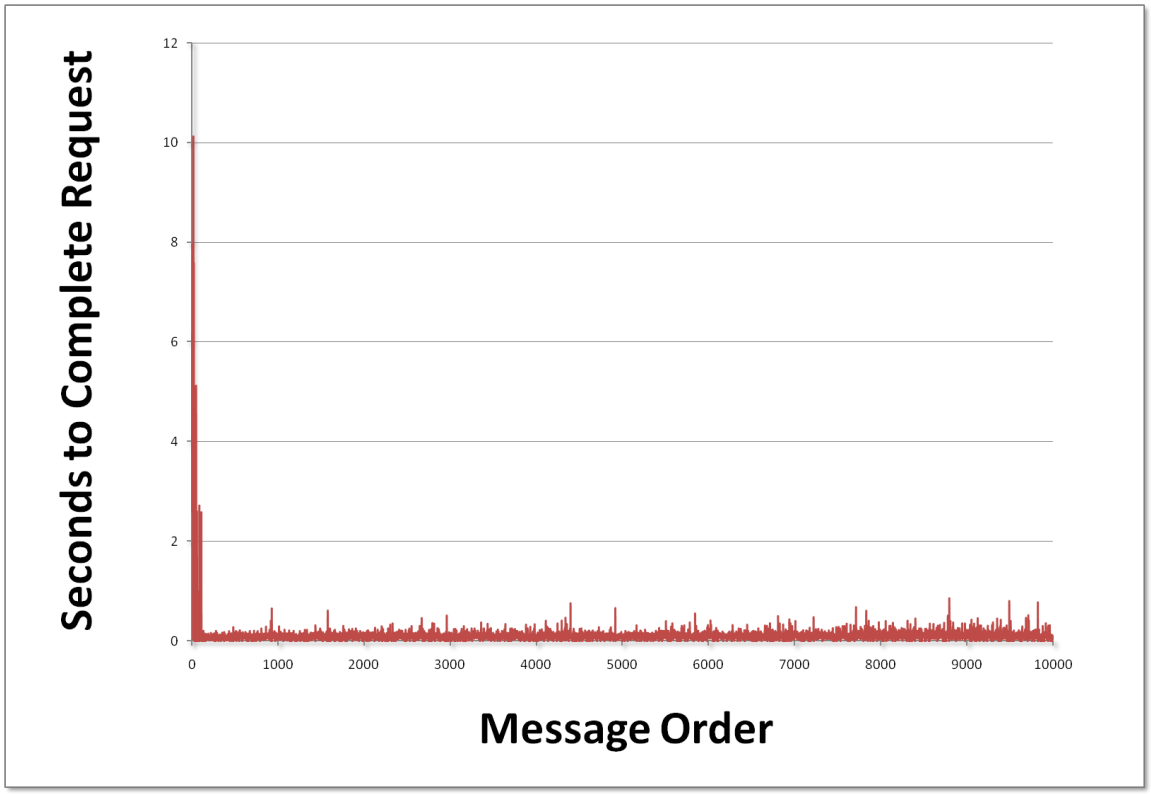


Figure 14. Performance Results of Social Strength Requests that Were Computed Using Direct Remote Peers or Completely Locally.

local graph. A non-uniform distribution was used for this test, which, as will be shown in Section 6.2.3, does increase the probability of an inference request being completed locally. 97.5% of the requests were done in less than a quarter of a second. The mean run time was 0.12 seconds and the median was 0.10 seconds.

6.2.3 Prometheus Performance

The neighborhood request results from the experiments on the LAN cluster and PlanetLab are shown in Figure 16. The experiment measures from when the request leaves the requester until the result is received. As expected, the requests submitted in PlanetLab take longer time to finish than in the cluster (approximately 10 times more). What is interesting is the difference in the response time between the uniform and nonuniform distributions of users onto peers: only about 32% of the requests finish in less than one second in the uniform distribution, compared to 42.7% in the nonuniform.

In Figure 17 we present the results from PlanetLab, grouped using the number of social hops. The results demonstrate that for one social hop, the user-peer distribution does not affect the time needed to fulfill a

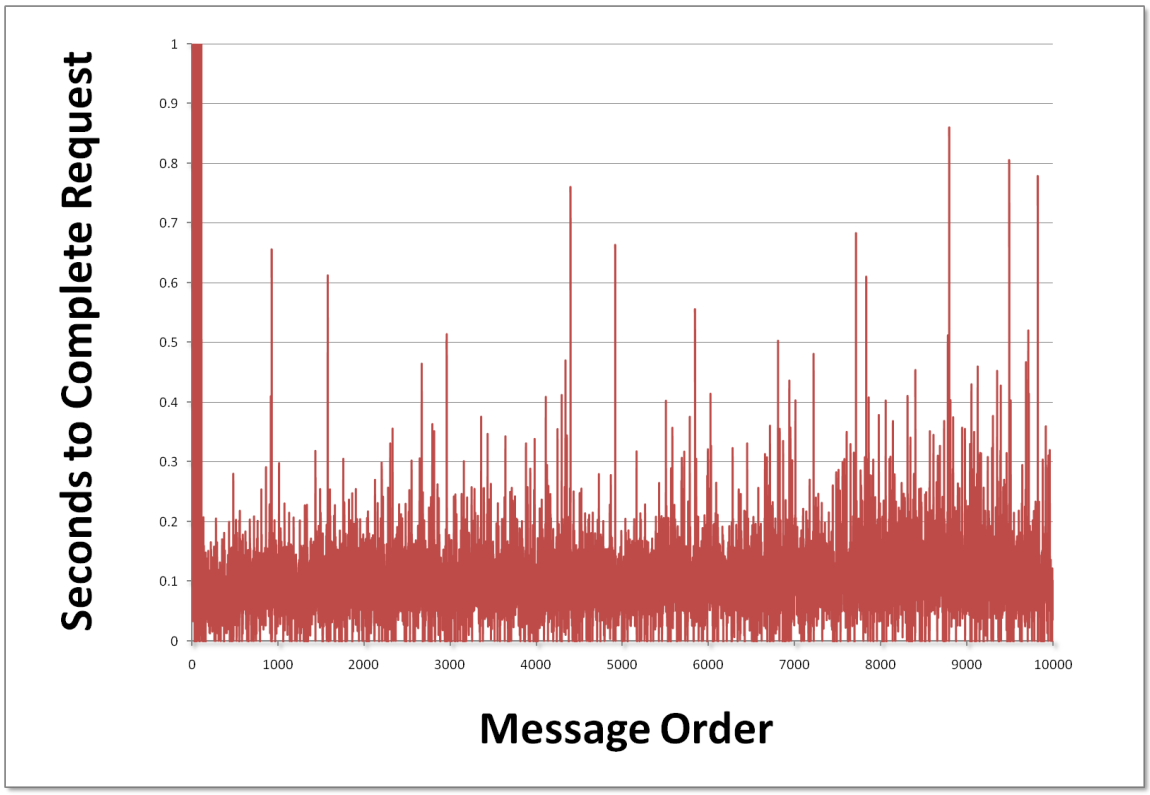


Figure 15. Social Strength Results Zoomed-In on One Second and Below.

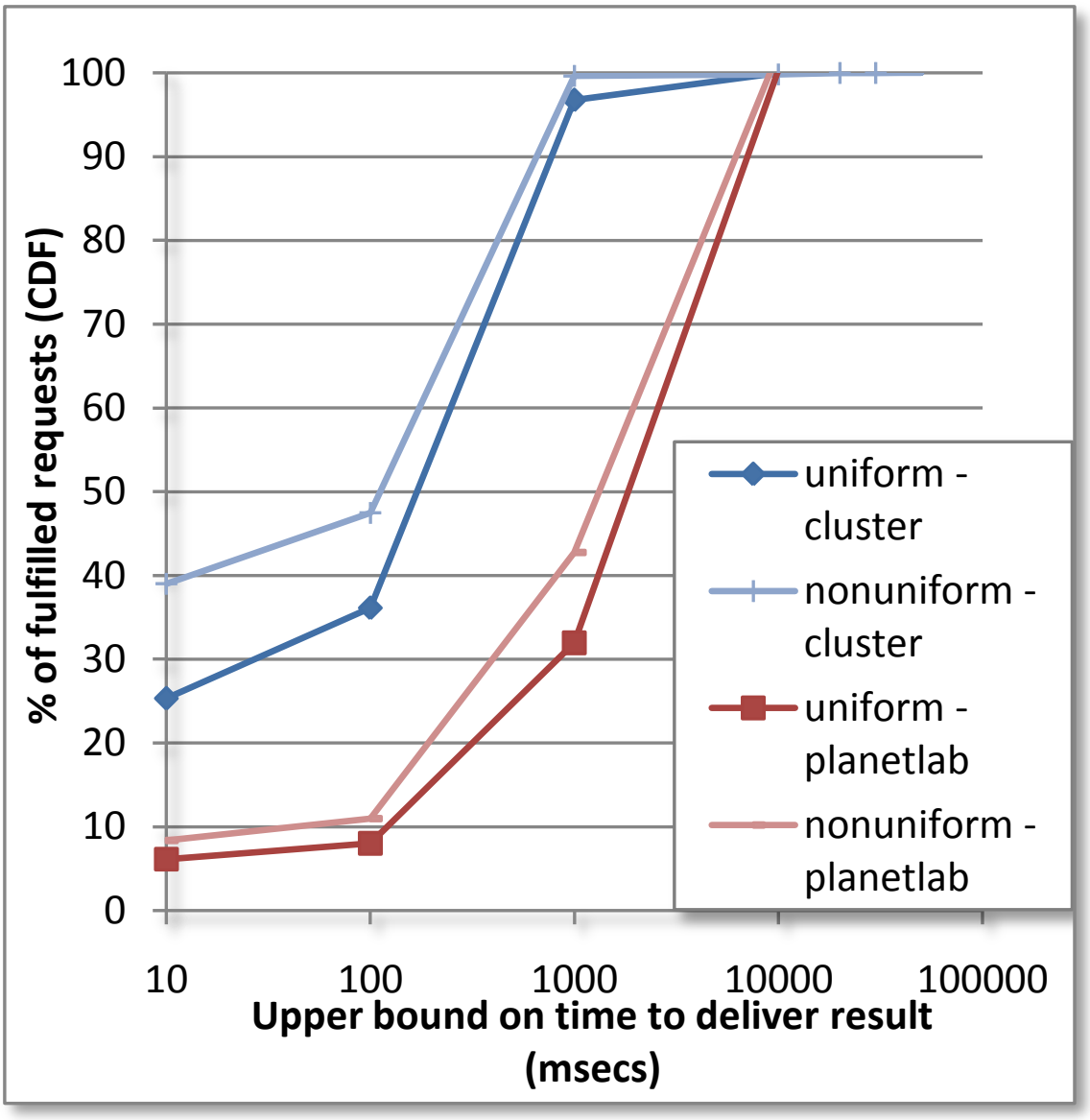


Figure 16. Fulfilled Neighborhood Requests Versus Time to Deliver Result (Over All Social Hops).

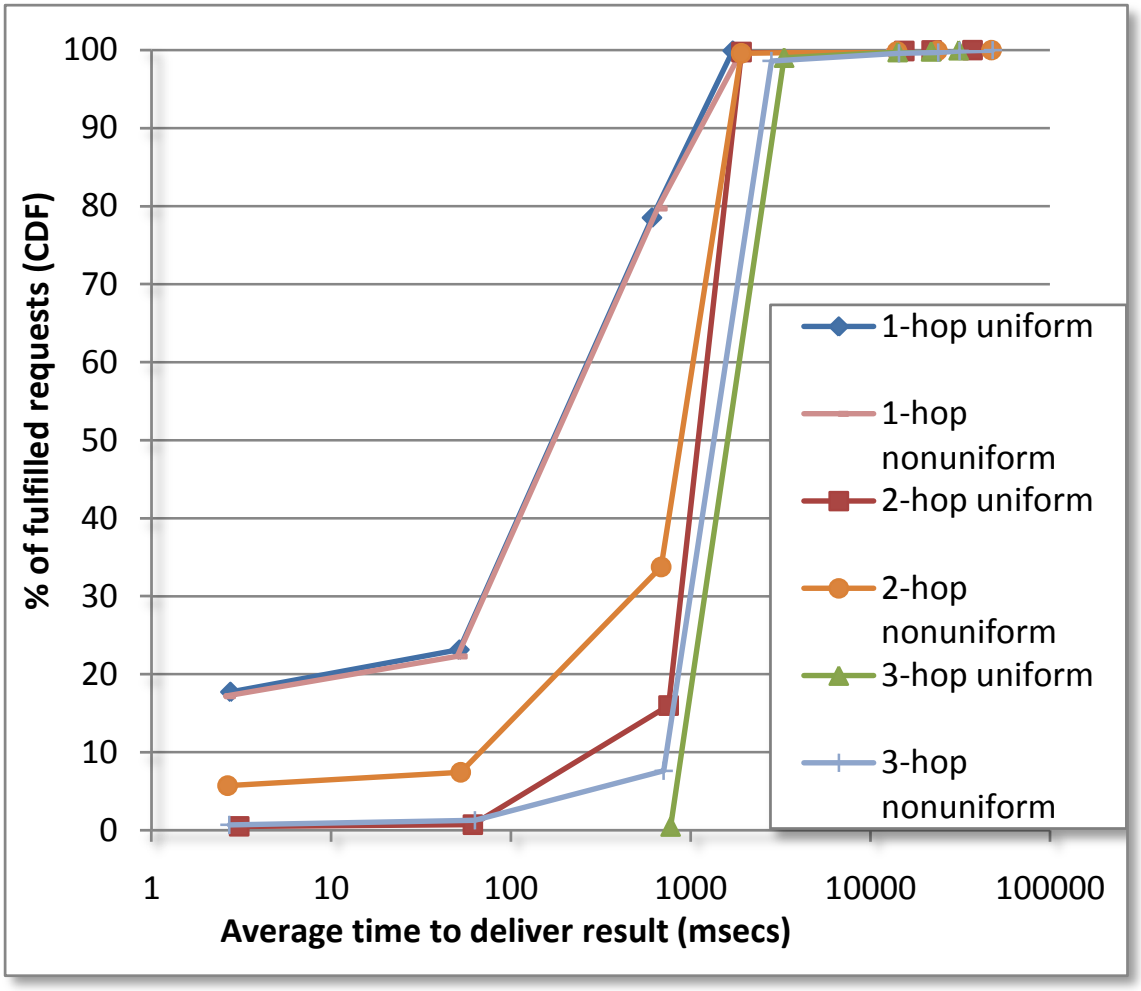


Figure 17. Fulfilled Neighborhood Requests Versus Time to Deliver Result (Grouped by Social Hops).

request. On the other hand, when the request is for two social hops, the difference is significant: only 16% of the requests finish in less than one second in the uniform distribution versus 33.7% in the nonuniform. When the request is for three social hops, the nonuniform distribution delivers the result about 3% faster than the uniform. We expect socially connected users to share peers and this sharing of peers clearly provides a performance improvement when a request could need remote information.

The social strength request results from the experiments on the LAN cluster and PlanetLab are shown in Figure 18. We notice the difference in the performance of the system, between uniform and nonuniform distribution of users onto peers. Only about 16.8% of the requests finish in less than one second in the uniform distribution, whereas 37.3% in the nonuniform. Note the similarity of the results from the social

strength requests (Figure 18) and the neighborhood requests for 2 social hops (Figure 16). This is because the calculation of the social strength request goes as far as two social hops.

Single hop requests in these tests have two distinct request completion points, one at 100 milliseconds and another at 1 second. One observation is that these times are much worse than from the perspective of GeoS. This shows that there is obviously some network delay in sending and receiving the request and its result. The 100 millisecond requests resulted from the request initiator, the peer that receives the request, and the peer that needs to fulfill the request being linked through a fast connection (likely from being the same machine or being close geographically.) Since several of the peers are on different continents, it does not seem unusual that the other requests take 1 second to complete. GeoS was shown completing local requests in less than one millisecond on average earlier. The PlanetLab computers are slower than the computers in the earlier test, but even if GeoS was ten times slower in this case, it would only be causing one percent of the overall processing time for most requests.

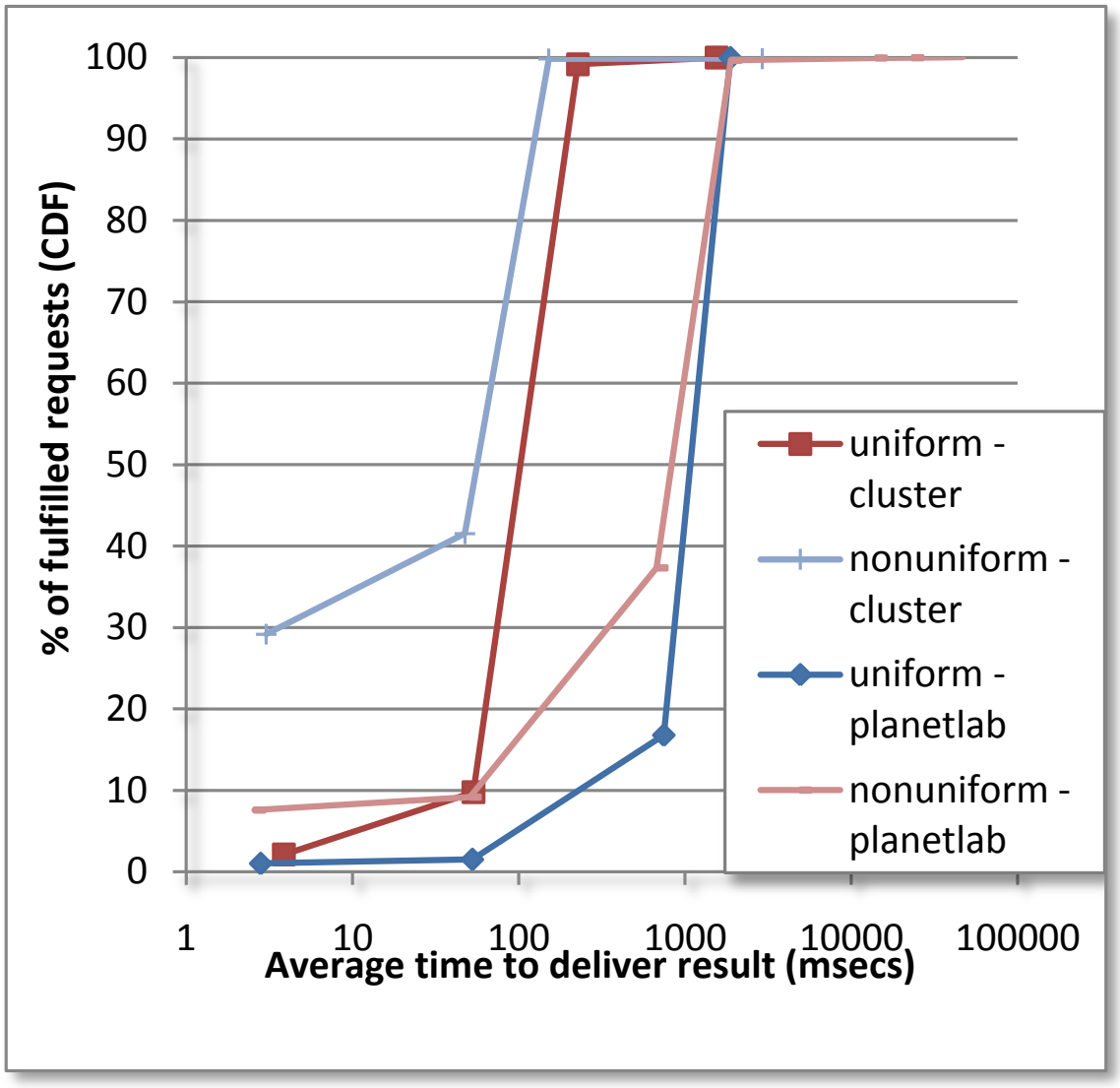


Figure 18. Fulfilled Social Strength Requests Versus Average Time to Deliver Result.

Chapter 7:

Related Work

Digital services and applications that work with social information are a popular subject currently, as is evidenced by the amount of recent research on the subject. Our exploration of related work first covers general research into the area of social networks, which is then followed by an examination of the many pieces of research that either collect social information or analyze social information collected by others, with us finally looking into applications and services that could make use of a social data storage service.

7.1 Social Networks

Our research on social networks focused on social graphs and social network analysis, but also included basic sociology concepts, such as the formation of social ties. The goal in the research was to gain enough knowledge of the sociology to make sure the design of GeoS stayed in line with generally accepted sociology theories.

The core idea that relationship strength should be based on interactions comes from [Hom51], a classic sociology book, in which Homans states "the more frequently persons interact with one another, the stronger their sentiments of friendship for one another are apt to be." As pointed out by Engeström in [Eng05], objects, such as a job or sports team, are what cause people to interact with each other. It was this idea that spawned the design decision of using a graph with multiple parallel edges between users, where each edge type is a class of object. By combining the ideas from those two sources, it was decided that each edge would have its own weight, allowing for the strength by which one class of object (out of possibly many) relates two users to be found.

[Gra73] is another classic sociology work. In it, Granovetter puts social connections in one of three groups: strong, weak, or absent. He finds that if there is an A-B link and an A-C link then there is a good chance of a B-C link. He also finds that weak ties are more likely to link different small groups than stronger ties. These significant findings are not directly used in GeoS, but they were important to remember while developing the system.

Granovetter also defines the strength of relationships as ”a (probably linear) combination of the amount of time, the emotional intensity, the intimacy (mutual confiding), and the reciprocal services which characterize the tie.” While some of these factors are likely difficult for a social sensor to calculate, the authors of social sensors should still keep them in mind. Note however that Granovetter later states in the paper that the amount of time invested into a relationship probably defines the relationship strength for some cases. This is much easier for social sensors to track and fits better with Homans' observations [Hom51].

Friedkin reports in [Fri83] that, “observability is very unlikely among persons who are three or more steps removed.” This is an essential finding for the design of the social strength function. Friedkin also reported in the same paper that observability is more likely with an increased number of paths from the observer to the other user.

In [YHC08], Yoneki et al. look at several data sets from CRAWDAD. The main focus of the paper is on finding different kinds of hub nodes (degree, rank, cross, date, and party.) The paper also looks at how removing hub nodes affects epidemic spread. The data sets are represented with undirected, weighted graphs. Hub nodes are not currently considered in GeoS, but [YHC08] revealed how useful the hub nodes can be for conveying information.

7.2 Data Collection

A large part of GeoS is its job of collecting social data from multiple sources. To aid in the design of this, we needed to do much research into the collection of social data. Our research focused on papers that used social information to infer information about the user, analysis of online social networks, and building a social graph using social information.

Lewis et al.'s paper [LKG+08] discusses using Facebook data to determine social relationships. It also gives ideas on how Facebook data can be used for determining a relationship. For example, it points out that if two users are tagged together in a photograph on Facebook, they must have been physically together. Another analysis of Facebook data occurs in [GWH07] where the Facebook message sending habits of 4.2 million Facebook users are analyzed. Observations of the authors include the following:

- only 15.1% of all pairs of analyzed Facebook friends exchanged messages with each other through Facebook
- users sent an average of 0.97 messages a week

- socializing through Facebook is more likely to happen in a time when the user is less likely to interact with people

All these observations are useful for developing a Facebook social sensor.

Eagle and Pentland use data solely from cell phones to infer friendships and routines in [EP06]. They also describe how cell phones are "wearable sensors." In addition, the paper contains statistics on how often cell phones were used for different functions in their study. The idea that routines could be detected allowed us to make the assumption that a social sensor could determine a particular interaction is just part of a routine. The inference of friendship is also a strong argument that accurate social sensors can be developed.

Matsuo et al. create a social graph based on the appearance of people's names together on webpages in [MMH+06]. A similar system presented in [CBM04] scans through a user's sent and received emails for email addresses and then searches the web for each of these addresses. By doing this, it can find the homepage of the person and from there, the user's contact information can be gathered along with those who are related to them. This allows a social network to be generated. Both papers provide the idea of a social sensor that spiders the web for relationships.

In [SA04], the Enron email data set is analyzed. This data set was used in early experimentation of GeoS because directed edges with realistic weights can be easily generated using the length and number of emails sent between users. Mislove et al. analyze the Flickr, LiveJournal, Orkut, and YouTube social networks along with social graphs based on them in [MMG+07]. This analysis is especially interesting because it shows the graph properties of some real social graphs.

Miluzzo et al. present CenceMe in [MLF+08]. CenceMe is a mobile phone application that uses the many different sensing devices available on a mobile phone (GPS, microphone, accelerometer, Bluetooth, camera) to detect where the owner of the phone is and what the owner is doing. In [PM09], the authors use multiple sensors (GPS, GSM, Wi-Fi, Bluetooth) to determine the places a user visits regularly and what activities the user does there. It can also predict activities and the visiting of places. The fact that both authors were able to infer the activity users were doing helps reinforce the fact that it is possible for a real social sensor to detect what class of activity two users are interacting in regards to.

Wyatt et al. use a microphone on a small device to detect when the wearer is having a conversation in [WBCK08]. It records conversations in a way that preserves privacy, but still had greater than 90% accuracy in detecting conversations in testing. The paper also found that speech patterns change less when someone is talking to a close relation. This may be useful information for developing collocation or phone call social sensors.

In [NK06], Nicolai et al. have a short discussion about the Bluetooth data the authors collected. The authors also present some of their theories on determining social context using Bluetooth. We expect that Bluetooth shall be the best way in which to find collocated users, and so this discussion is important when developing a collocation social sensor.

The authors of [SR08] find that people who chat with each other are more likely to have similar interests. Also, those people are likely to be the same age and from the same location and of opposite gender. In addition, they find that people who have mutual friends are likely to have similar interests. Neto et al. find that users with similar tagging activity likely share interests in [NCA+09]. These findings may be useful when developing social sensors. For example, maybe the intensity of an interaction should be stronger if the two users who are interacting are less likely to interact.

7.3 Socially-Aware Applications

The overall objective of GeoS is to allow socially-aware applications to query it for the social information of users. An important stage in the design of GeoS was determining what type of social data would be needed by these applications. We generated some of our own ideas for socially-aware applications, but also researched what applications had already been developed.

A "context-aware mobile phone" is created in [SSF+03]. Basically, the project will turn off a phone's ringer and do other things to the phone based on what context the user is in. It uses sensor data (accelerometer, microphone, light sensor) together with the user's calendar. The paper was of great use while designing GeoS. First, it silences cell phones, which is an early goal that we wanted a socially-aware application that uses GeoS to be able to do. Second, it combines data from multiple sources (cell phone sensors and the user's calendar.) However, the phone silencer only silences an incoming call if the owner is speaking or if their calendar says they are busy at the time. The former could have problems if the owner is in a situation where the phone should be silence, but the user is not talking (such as a lecture.) Also, it could silence an important phone call while the owner is having a casual conversation. The latter requires some work from the user. For these reasons, we feel it would be better for the device to also use the user's social context.

Ostra, a system to reduce spam, uses a social graph to determine the maximum number of messages a user can send before requiring at least one of their sent messages to be marked as a wanted communication [MPDG08]. SybilGuard uses the fact that in the case of a sybil attack, all of the sybil identities will be close to fully-connected with each other in order to reduce the impact of a sybil attack [YKGF06]. Friendstore allows users to backup their data onto the computer of those they have a social relationship with [TCL08]. All

of these applications use simple boolean edges for relationships and we believe that they would benefit from having a more complex social graph available to them. For example, Friendstore could possibly allow a remote user A to store more data on the system than remote user B if there existed a stronger social relationship between the owner and A than between the owner and B.

Li and Dabek found that using social relationship information in a peer-to-peer network provides benefits due to users having social incentives to stay online [LD06]. Hogg and Adamic recommend using a weighted social graph for a visible reputation level that would be difficult to spoof [HA04]. RE uses two hop relationships to automatically populate email whitelists [GKF+06].

Chapter 8:

Future Work

GeoS provides many novel features, but there are still several ideas we would like to research. The inclusion of more social inference functions will support the development of advanced socially-aware applications. GeoS will eventually manage the entirety of a user's social data and to this end, implementing support for the management of social data other than a user's relationships and geo-location is needed. Finally, some questions are still open which require further research.

8.1 Additional Social Inference Functions

Several new social inference functions have been considered, but require further research. First, a more accurate version of the social strength function has been hypothesized. When calculating the strength of indirect links, the current version only considers the strongest path. In addition to the "Horizon of Observability", Friedkin observed in [Fri83] that users with more paths linking them to an indirect colleague were more likely to know of the colleague's work. Using this observation, we would like to extend the social strength function to take into account the number of paths linking two users. We propose an algorithm for this function in Figure 19. This algorithm utilizes some ideas from [DGS09], where the authors present an algorithm for calculating the probability of trust between two users who are not directly connected. The algorithm we propose is incomplete as it is completely untested and we do not completely agree with its results in the case of a path where all the edge weights are 1.0. In such a case, the result is a social strength of 1.0, which seems too high when the users may not even know each other.

Including algorithms that can estimate the graph measurements of a node without requiring the entirety of the graph would add a very powerful feature to GeoS. For example, the betweenness centrality of a node is useful for determining the importance of a node in connecting others, but it requires the entire graph. Because we expect many users will limit their social information to friends of friends, and to refrain from creating a huge amount of network traffic, a graph algorithm in GeoS should not need more than two hops of social information. Another possibility is for the algorithms to be written in such a way that a user could give needed information while retaining its privacy.

input : ego, alter
output: A real number between 0 and 1 that quantifies the social strength between ego and alter from ego's perspective

```
Sum(userA, userB):
begin
    foreach tag in Edges(userA, userB) do
        sum ⟵ Weight(userA, userB, tag);
    end
end

MaxSum(userA):
begin
    max ← 0;
    foreach userB in Neighbors(userA) do
        if Sum(userA, userB) > max then
            max ← Sum(userA, userB);
        end
    end
end

NW(userA, userB):
begin
    if userA is userB then
        1.0;
    else
        Sum(userA, userB)/MaxSum(userA);
    end
end

PathStrength(userA, userB, userC):
begin
    if NW(userA, userB) < NW(userB, userC) then
        1 − NW(userA, userB)^(2/pathLength);
    else
        1 − (NW(userA, userB) ∗ NW(userB, userC))^(1/pathLength);
    end
end

Social_Strength(ego, alter):
begin
    foreach userB in Neighbors(ego) do
        if alter in Neighbors(userB) then
            product ← product ∗ PathStrength(ego, userB, alter);
        end
    end
    1 − product;
end
```

Figure 19. Proposed Algorithm for New Social_Strength(ego, alter).

A final social inference function that has been contemplated is one that automatically infers a user's interests by analyzing the social graph of the user. The basic idea is that the stronger the weights a user has for a particular tag, the more that user must interact regarding that tag. If a user takes part in a particular activity A more than an activity B, the user must be more interested in activity A. Determining interest levels could be as simple as summing all of the weights of each tag a user is directly connected through and comparing each tag's summed weight to each other tag's summed weight.

8.2 Expanded Social Data

There are several pieces of social information that could be managed by GeoS. These include contact information (for the user and its friends), schedule (declared and inferred), and group membership (declared and inferred). By storing social information in GeoS, the social information is available in a distributed system and can make use of or be used in social inference functions.

Contact information can have a type of access control list that requires that ego be connected to users by a particular path in order for them to gain access to a certain field of contact information. For example, a user may say that the only contact information it wants given to those it has a work relationship with is its office phone or a different user may say a social strength of at least 0.3 is required to access its email address. This would greatly simplify the sharing of contact information for users who have multiple forms of communication open to them.

A schedule is a basic piece of social information, but we propose that since it is possible to infer routines [EP06], a complete schedule for a user could be inferred using past knowledge together with declared events. Inferred events would include a probability that the event will occur and outside users could query this inferred schedule for times when the user might be available. Events could include tags that are parallel to edge tags and describe the event. Similar to the contact information, a type of access control list can be placed on the schedule so that only certain users would be able to know the user may be available at a particular time. Querying for availability is meant to ease the finding of time slots where multiple users can meet while filtering access to availability is designed to reduce the chance a user will be asked if they are available at a time in which they are busy or doing an activity the requestor may not be interested in. For example, if the inferred schedule believes there is a 60% chance a user is doing a hiking activity at a given time, the user may only allow strongly connected hikers to see that the user is likely available at that time.

Research has been done into the inference of group membership [GPJB07]. GeoS could use the knowledge of group membership in several ways. It could be used to validate the social graph within GeoS, the

social graph in GeoS could be used to validate the algorithm's findings, or GeoS could use group membership for social inferences.

8.3 Open Questions

GeoS requires social sensors and requires that social sensors provide it accurate input. Exactly how this input is determined requires further research. When designing a set of social sensors, not only must they be able to take interactions and turn them into accurate weights, but the social sensors must also be able to take into account the weight that other social sensors have already added to the graph. It is very much possible that two users will discuss an activity by phone or email and then meet in person to do that activity. This case would result in the phone or email sensor sending an input for the tag associated with that activity followed by the collocation sensor later sending another input for that same activity tag. One idea is that social sensors will include the current weight of the edge when calculating the weight of a new input. If this idea is possible, it may reduce the history needed for calculating a weight, but would result in more calls to GeoS. Another idea is that each social sensor will create its own tag that is a subset of another tag (for example, "hiking-email" and "hiking-collocation" would be subsets of "hiking") and GeoS will aggregate these tags in an intelligent way to calculate the superset.

The tag to give input is also still under consideration. GeoS currently allows any tag to be sent with input, but it can be argued that this is too open because one social sensor may label an activity "work" while another labels the same activity "job". Thus, it may be better to restrict tags to a published set despite making the system a little more closed off. Another idea for tags (which requires a set group of tags) is that they should be arranged in a hierarchy going from specific to general. For example, "baseball" and "football" may be a children of "sports", which may itself be a child of "recreation". The weight of the sports tag would be based on the weight of its children along with any input sent directly to that tag. The most difficult part of this approach would be determining the weight of tags that have children.

The final open question is how updates should be handled. The current approach is very simple and can result in data being lost due to a race condition or invalid data being propagated through a group of trusted peers. Much research has been done recently into consistency in a peer-to-peer system, but we want to be sure to choose the optimal solution before implementing it in GeoS.

Two solutions for ensuring consistency in a peer-to-peer network have been looked into thus far. These are:

- Balanced Consistency Maintenance (BCoM) Protocol [HFB10]
- WOOT Framework [OUMI06]
- Atlas Peer-to-Peer Architecture [AM06]

With the BCoM Protocol, for each shared object (e.g. a user's SSDF in our case), a dissemination tree structure is formed above the P2P overlay. Updates are then sent to the root of the tree and propagated to the rest of the peers. The WOOT framework is designed for consistency in collaborative editing over a P2P network, but the removal and addition of edges, weights, and other social information seen in GeoS is similar to the removal and addition of characters in text. The framework provides a set of algorithms that can ensure that insertions and deletions are applied in the correct order across the P2P network. The Atlas architecture uses a timestamping service that provides a monotonically increasing timestamps to the peers. This allows peers to detect missed updates and them in the correct order when later received.

Chapter 9:

Conclusion

We have shown that GeoS can combine social data from many separate sources into a weighted, directed, and labeled multigraph data structure. Thanks to the popularity of ubiquitous computing and online social networks, there are vast quantities of social data easily available to be fed to GeoS.

The social information within GeoS is stored on a decentralized system and allows for inferences of user's social information. The inferred social information can be used by socially-aware applications to make them more complex than if they had used an online social network. The decentralization of social data increases privacy and helps avoid a "big brother" scenario. We presented the currently available social inference functions and demonstrated their performance.

We also presented applications that could use the results of the social inference functions. In addition, the functionality of the social inference functions was tested. The other services in the Prometheus architecture, of which GeoS is a part, were briefly discussed.

GeoS fulfills our original design objectives of storing social data in a decentralized system, while recording social input from any type of social sensor, and providing a set of social inference functions that can be used by the applications of users.

References

[AHK+07] Ahn, Yong Yeol, Seungyeop Han, Haewoon Kwak, Sue Moon, and Hawoong Jeong: *Analysis of topological characteristics of huge online social networking services*. In *WWW '07: Proceedings of the 16th international conference on World Wide Web*, pages 835–844, New York, NY, USA, 2007. ACM, ISBN 978-1-59593-654-7.

[AKFI10] Anderson, P., N. Kourtellis, J. Finnis, and A. Iamnitchi: *On Managing Social Data for Enabling Socially-Aware Applications and Services*. In *Social Network Systems 2010*. ACM, 2010.

[AM06] Akbarinia, R. and V. Martins: *Data management in the APPA P2P system*. In *Int. Workshop on High-Performance Data Management in Grid Environments (HPDGrid)*. Citeseer, 2006.

[AYP+05] Anwar, Zahid, William Yurcik, Vivek Pandey, Asim Shankar, Indranil Gupta, and Roy H. Campbell: *Leveraging social-network infrastructure to improve peer-to-peer overlay performance: Results from orkut*. CoRR, abs/cs/0509095, 2005.

[BF01] Bourgeois, Michael and Noah E. Friedkin: *The distant core: social solidarity, social distance and interpersonal ties in core-periphery structures*. Social Networks, 23(4):245–260, October 2001.

[Bia97] Bian, Yanjie: *Bringing strong ties back in: Indirect ties, network bridges, and job searches in china*. American Sociological Review, 62(3):366–385, 1997, ISSN 00031224.

[BKKS84] Bernard, H. Russell, Peter Killworth, David Kronenfeld, and Lee Sailer: *The problem of informant accuracy: The validity of retrospective data*. Annual Review of Anthropology, 13:495–517, 1984, ISSN 00846570.

[CBM04] Culotta, Aron, Ron Bekkerman, and Andrew Mccallum: *Extracting social networks and contact information from email and the web*. In *Collaboration, Electronic messaging, Anti-Abuse and Spam Conference*. Citeseer, 2004.

[DGS09] Dubois, Thomas, Jennifer Golbeck, and Aravind Srinivasan: *Rigorous probabilistic trust-inference with applications to clustering. submitted to the*. In *IEEE / WIC / ACM Conference on Web Intelligence*, 2009.

[Eng05] Engeström, Jyri: *Why some social network services work and others don't or: the case for object-centered sociality*, 2005. http://bit.ly/aVrQRb, visited on 2010-01-23.

[EP06] Eagle, Nathan and Alex (Sandy) Pentland: *Reality mining: sensing complex social systems*. Personal and Ubiquitous Computing, 10(4):255–268, May 2006, ISSN 1617-4909 (PRINT) 1617-4917 (ONLINE).

[Faca] Facebook: *Facebook — statistics*. http://www.facebook.com/press/info.php?statistics, visited on 2010-02-17.

[Facb] Facebook: *Statement of rights and responsibilities*. http://www.facebook.com/terms.php, visited on 2010-02-17.

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

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

[GKF+06] Garriss, S., M. Kaminsky, M.J. Freedman, B. Karp, D. Mazières, and H. Yu: *RE: reliable email*. In *Proceedings of the 3rd conference on Networked Systems Design & Implementation*, pages 22–22, 2006.

[GM04] Goecks, Jeremy and Elizabeth D. Mynatt: *Leveraging social networks for information sharing*. In *CSCW '04: Proceedings of the 2004 ACM conference on Computer supported cooperative work*, pages 328–331, New York, NY, USA, 2004. ACM, ISBN 1-58113-810-5.

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

[GPJB07] Gupta, A., S. Paul, Q. Jones, and C. Borcea: *Automatic identification of informal social groups and places for geo-social recommendations*. International Journal of Mobile Network Design and Innovation, 2(3):159–171, 2007.

[Gra73] Granovetter, Mark S.: *The strength of weak ties*. The American Journal of Sociology, 78(6):1360–1380, May 1973.

[GWH07] Golder, S. A., D. Wilkinson, and B. A. Huberman: *Rhythms of social interaction: Messaging within a massive online network*. In *3rd International Conference on Communities and Technologies*, June 2007.

[HA04] Hogg, Tad and Lada Adamic: *Enhancing reputation mechanisms via online social networks*. In *Proceedings of the 5th ACM conference on Electronic commerce - EC '04*, page 236, 2004.

[HFB10] Hu, Y., M. Feng, and L.N. Bhuyan: *A Balanced Consistency Maintenance Protocol for Structured P2P Systems*. 2010 Proceedings IEEE INFOCOM, pages 1–5, 2010.

[Hom51] Homans, George C.: *The human group / by George C. Homans*. Routledge & Kegan, London, 1951.

[KBI09] King, Zach, Jeremy Blackburn, and Adriana Iamnitchi: *Battorrent: A battery-aware bittorrent for mobile devices*. In *Proceedings of the 11th International Conference on Ubiquitous Computing, Poster Session*, 2009.

[KFA+10] Kourtellis, N., J. Finnis, P. Anderson, J. Blackburn, and A. Iamnitchi: *Prometheus: Distributed management of geo-social data*. In *USENIX NSDI*, 2010.

[KGA08] Krishnamurthy, Balachander, Phillipa Gill, and Martin Arlitt: *A few chirps about twitter*. In *WOSP '08: Proceedings of the first workshop on Online social networks*, pages 19–24, New York, NY, USA, 2008. ACM, ISBN 978-1-60558-182-8.

[Kha09] Khan, Urmee: *Facebook controversy over right to delete personal information*, February 2009. http://bit.ly/2HaBtU, visited on 2010-02-09.

[KRZBS07] Kleck, Christine, Christen Reese, Dawn Ziegerer-Behnken, and S. Shyam Sundar: *The company you keep and the image you project: Putting your best face forward in online social networks*. In *57th Annual Conference of the International Communication Association*, San Francisco, CA, 2007.

[LD06] Li, Jinyang and Frank Dabek: *F2f: reliable storage in open networks*. In *Proceedings of the 5th International Workshop on Peer-to-Peer Systems*, February 2006.

[LHP+07] Logan, B., J. Healey, M. Philipose, E. Tapia, and S. Intille: *A long-term evaluation of sensing modalities for activity recognition*. UbiComp 2007: Ubiquitous Computing, pages 483–500, 2007.

[LKG+08] Lewis, Kevin, 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, ISSN 0378–8733.

[Loc] Locale: *Locale for android*. http://www.twofortyfouram.com/, visited on 2009-11-12.

[McM] McMillan, Gordon: *Socket programming howto*. http://docs.python.org/dev/howto/sockets.html, visited on 2009-05-11.

[MLF+08] Miluzzo, Emiliano, 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 *SenSys '08: Proceedings of the 6th ACM conference on Embedded network sensor systems*, pages 337–350, New York, NY, USA, 2008. ACM, ISBN 978-1-59593-990-6.

[MMG+07] Mislove, Alan, Massimiliano Marcon, Krishna P. Gummadi, Peter Druschel, and Bobby Bhattacharjee: *Measurement and analysis of online social networks*. In *Proceedings of the 5th Internet Measurement Conference*, October 2007.

[MMH+06] Matsuo, Yutaka, 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 *WWW '06: Proceedings of the 15th international conference on World Wide Web*, pages 397–406, New York, NY, USA, 2006. ACM Press, ISBN 1595933239.

[MPDG08] Mislove, Alan, Ansley Post, Peter Druschel, and Krishna P. Gummadi: *Ostra: leveraging trust to thwart unwanted communication*. In *Proceedings of the 5th Symposium on Networked Systems Design and Implementation*, pages 15–30, Berkeley, CA, USA, 2008. USENIX Association, ISBN 111-999-5555-22-1.

[MS63] Morgan, James N. and John A. Sonquist: *Problems in the analysis of survey data, and a proposal*. Journal of the American Statistical Association, 58(302):415–434, 1963, ISSN 01621459.

[NCA+09] Neto, Elizeu S., David Condon, Nazareno Andrade, Adriana Iamnitchi, and Matei Ripeanu: *Individual and social behavior in tagging systems*. In *HT '09: Proceedings of the 20th ACM conference on Hypertext and hypermedia*, pages 183–192, New York, NY, USA, 2009. ACM, ISBN 978-1-60558-486-7.

[NK06] Nicolai, Tom and Holger Kenn: *Towards detecting social situations with bluetooth*. In *Ubicomp*, 2006.

[NM09] Neate, Rupert and Rowena Mason: *Networking site cashes in on friends*, January 2009. http://bit.ly/1aPH, visited on 2010-02-10.

[OUMI06] Oster, G., P. Urso, P. Molli, and A. Imine: *Data consistency for P2P collaborative editing*. In *Proceedings of the 2006 20th anniversary conference on Computer supported cooperative work*, page 268. ACM, 2006.

[PFF+03] Philipose, M., K. Fishkin, D. Fox, H. Kautz, D. Patterson, and M. Perkowitz: *Guide: Towards understanding daily life via auto-identification and statistical analysis*. In *Proc. of the Int. Workshop on Ubiquitous Computing for Pervasive Healthcare Applications (Ubihealth)*. Citeseer, 2003.

[PM09] Papliatseyeu, Andrei and Oscar Mayora: *Mobile habits: Inferring and predicting user activities with a location-aware smartphone*. In *3rd Symposium of Ubiquitous Computing and Ambient Intelligence 2008*, volume 51 of *Advances in Soft Computing*, pages 343–352. Springer Berlin / Heidelberg, 2009, ISBN 978-3-540-85866-9.

[PRR97] Plaxton, C. Greg, Rajmohan Rajaraman, and Andréa W. Richa: *Accessing nearby copies of replicated objects in a distributed environment*. In *SPAA '97: Proceedings of the ninth annual ACM symposium on Parallel algorithms and architectures*, pages 311–320, New York, NY, USA, 1997. ACM, ISBN 0-89791-890-8.

[RD01] Rowstron, Antony and Peter Druschel: *Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems*. In *Middleware '01: Proceedings of the IFIP/ACM International Conference on Distributed Systems Platforms Heidelberg*, pages 329–350. Springer-Verlag, 2001.

[RKCD01] Rowstron, A., A.M. Kermarrec, M. Castro, and P. Druschel: *SCRIBE: The design of a large-scale event notification infrastructure*. Networked Group Communication, pages 30–43, 2001.

[SA04] Shetty, J. and J. Adibi: *The enron email dataset database schema and brief statistical report*. Information Sciences Institute Technical Report, University of Southern California, 2004.

[SCW+10] Sala, Alessandra, Lili Cao, Christo Wilson, Robert Zablit, Haitao Zheng, and Ben Y. Zhao: *Measurement-calibrated graph models for social network experiments*. In *19th International World Wide Web Conference*. ACM, April 2010.

[SR08] Singla, Parag and Matthew Richardson: *Yes, there is a correlation: - from social networks to personal behavior on the web*. In *WWW '08: Proceeding of the 17th international conference on World Wide Web*, pages 655–664, New York, NY, USA, 2008. ACM, ISBN 978-1-60558-085-2.

[SSF+03] Siewiorek, Daniel, Asim Smailagic, Junichi Furukawa, Andreas Krause, Neema Moraveji, Kathryn Reiger, Jeremy Shaffer, and Fei Lung Wong: *Sensay: A context-aware mobile phone*. In *Proceedings of the 7th IEEE International Symposium on Wearable Computers*, page 248, Washington, DC, USA, 2003. IEEE Computer Society, ISBN 0-7695-2034-0.

[TCL08] Tran, Dinh Nguyen, Frank Chiang, and Jinyang Li: *Friendstore: cooperative online backup using trusted nodes*. In *SocialNets '08: Proceedings of the 1st workshop on Social network systems*, pages 37–42, New York, NY, USA, 2008. ACM, ISBN 978-1-60558-124-8.

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

[WBCK08] Wyatt, Danny, Jeff Bilmes, Tanzeem Choudhury, and James A. Kitts: *Towards the automated social analysis of situated speech data*. In *UbiComp '08: Proceedings of the 10th international conference on Ubiquitous computing*, pages 168–171, New York, NY, USA, 2008. ACM, ISBN 978-1-60558-136-1.

[WBS+09] Wilson, Christo, Bryce Boe, Alessandra Sala, Krishna P. N. Puttaswamy, and Ben Y. Zhao: *User interactions in social networks and their implications*. In *Proceedings of the 4th European conference on Computer systems*, pages 205–218, New York, NY, USA, 2009. ACM, ISBN 978-1-60558-482-9.

[Wel88] Wellman, Barry: *Structural analysis: From method and metaphor to theory and substance*, pages 19–61. Cambridge University Press, 1988, ISBN 0-521-24441-2 (HARDCOVER); 0-521-28687-5 (PAPERBACK). ID: 1988-97324-001.

[Won10] Wong, Phil: *Conversations about the internet 5: Anonymous facebook employee*, January 2010. http://bit.ly/7O2Yqf, visited on 2010-02-10.

[YHC08] Yoneki, Eiko, Pan Hui, and Jon Crowcroft: *Distinct types of hubs in human dynamic networks*. In *SocialNets '08: Proceedings of the 1st workshop on Social Network Systems*, pages 7–12, New York, NY, USA, April 2008. ACM, ISBN 978-1-60558-124-8.

[YKGF06] Yu, Haifeng, Michael Kaminsky, Phillip B. Gibbons, and Abraham Flaxman: *Sybilguard: defending against sybil attacks via social networks*. In *SIGCOMM '06: Proceedings of the 2006 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 267–278, New York, NY, USA, 2006. ACM, ISBN 1-59593-308-5.

[ZC10] Zhang, Jian and Chaomei Chen: *Collaboration in an open data escience: A case study of sloan digital sky survey*. CoRR, abs/1001.3663, 2010.