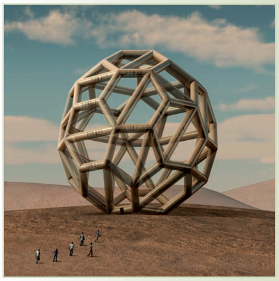

The Social Hourglass

An Infrastructure for Socially Aware Applications and Services

As the Internet's hourglass architecture connects various resources to various applications, an infrastructure that collects information from various social signals can support an ever-evolving set of socially aware applications and services. Among the proposed infrastructure's features are social sensors to capture and interpret social signals from user interactions, a personal social information aggregator, and a set of social-inference functions as its API for social applications.

Adriana Iamnitchi, Jeremy Blackburn, and Nicolas Kourtellis
University of South Florida

Given the abundance of interpersonal data exposed by Web 2.0 applications, we have a tremendous opportunity to exploit social information for myriad applications that can truly enhance our well being. These applications can go well beyond the keeping-in-touch functionalities that made online social networks (OSNs) popular and into domains such as medical and social research, emergency interventions, and personalized service discovery.

We're developing a unifying infrastructure that can capture multiple types of social information in a social knowledge management service to support a variety of socially aware applications and services. This system uses social sensors as the user-controlled collectors of already available or recordable social interactions between people. It then personalizes social information from multiple sources on a user's device and sends it to a social knowledge management service. This service exposes an API used by applications to, for example, infer trust between users or discover users with particular interaction characteristics in the social neighborhood. Here, we describe the proposed architecture and present a proof of concept social sensor implementation based on online gaming that, combined with our previous implementation of a social knowledge management service,[1] shows the architecture's practicality.

The Social Hourglass Infrastructure

Today's Web 2.0 applications have exposed an abundance of social information, enabling service providers to mine information about the relationships between users to provide improved service. This approach has been exploited in various domains, most notably security and spam protection,[2] emergency medical alerts,[3] and recommendation systems such as Yelp. These socially aware applications rely on what we refer to as *social signals*: information that exposes social relationships between people. Such relationships can be declared, as in the case of friendship on Facebook or LinkedIn, or interaction-based, such as those inferred from recorded colocation traces,[4] conversations,[5] sociometric badges,[6] or online gaming (as we describe later). Vast and diverse social signals already exist as byproducts of Internet- or phone-mediated interactions, such as email logs, comments on blogs, like/dislike votes on user-generated content, and phone call history.

Interaction-based social signals provide an unprecedented level of detail compared to the binary declared relationships typical of OSNs. Such signals make it possible to quantify a social relationship's strength based on domain-specific metrics, such as quantity (for example, phone call duration or the number of characters in an IM exchange) and frequency. They also convey more accurate information than declared relationships, which users create casually and rarely remove.

Given the opportunities for exploiting social information, we think it's time to move from the vertically integrated approach — in which one source of information is mined for a specific application — to a service infrastructure that can synthesize information from multiple types of social interactions and thereby offer accurate and personalized support for various social applications. Just as the Internet's hourglass architecture connects various resources to various applications, this infrastructure must be able to absorb information from an unrestricted set of social signals and export it to an ever-evolving collection of socially aware applications and services.

Such an infrastructure would enable a continuum of innovative social applications and services. For example, a couch-surfing application that finds travelers lodging for a night could explore the traveler's social *k*-hop neighborhood to select people with similar social interaction patterns, such as hiking or playing online games (both hobbies in which relationships often persist beyond the activity itself). A personal-cloud service could mine the implicit social incentives of a two-hop neighborhood to recruit cycles for a computationally intensive search-and-rescue mission.[7] A context-aware phone-call filtering application (such as CallCensor[1]) might filter calls based on the social relationship between the user and caller, as well as people colocated with the user so that, for example, personal calls are automatically silenced during professional meetings, but coworkers' calls are let through.

Architecture

An infrastructure that supports a variety of social applications must meet several requirements. First, it must accept input from diverse social signals, differentiating information based on interaction type and intensity. Exposing relationship type and strength lets applications not only use context-relevant relationships but also compare these relationships. For example, the CallCensor application could be configured to accept calls from non-coworkers who have a strong social tie with the call recipient.

Second, the infrastructure must allow for user-controlled fusion of social signals. For some users, a social interaction during weekly online multiuser gaming might translate into trust outside the gaming context, such as for occasional couch surfing while traveling. For others, a trusted social relationship might require a combination of more traditional interactions such as physical colocation, phone conversations, and sharing photographs from family vacations. In any case, the choice and relative importance of the relevant social signals should be personalized and fully under the user's control.

Finally, the infrastructure must include a persistent social knowledge management service scalable with the number of users represented and the number of social signals read. Such a service should support a variety of social requests through a basic API. For example, a user might want to find new climbing partners for a trip by asking the climbing partners of her current partners, who are likely a trusted set of skilled and reliable mountaineers. This query requires access to a neighborhood of distance 2 from the enquiring user, making demands on the distribution of the social information on storage nodes and node availability in a decentralized service.

These three requirements correspond loosely to the architectural components Figure 1 depicts. In this multilayered architecture, the bottom layer encompasses the diversity of existing or future social signals. Without necessarily calling them such, researchers have collected and analyzed many social signals — such as messaging,[8] phone call patterns,[9] or Facebook friendship declarations.[10] Although many social signals are publicly available on the Web or recorded by mobile phone applications, new signals are likely to emerge as a consequence of new applications or be specifically crafted for consumption by an infrastructure like the one we propose.

The next architectural layer consists of social sensors: applications running on a user's behalf on various platforms — such as mobile phones, Web browsers, or third-party servers — that analyze one or more social signals, transforming the domain-specific interactions for the same social activity into a weighted and labeled social edge.

The social edges reported by all sensors deployed on a user's behalf are then directed to the next layer, the personal aggregator. This trusted application, which typically runs on a user-owned device, fuses multiple same-activity social edges and personalizes user's social relations.

Once the aggregator has personalized it, it sends this digital representation of social relations to the *social knowledge service* (SKS) layer. The SKS provides a mechanism for storing, managing, and exposing social data to applications, subject to user-defined data access control.

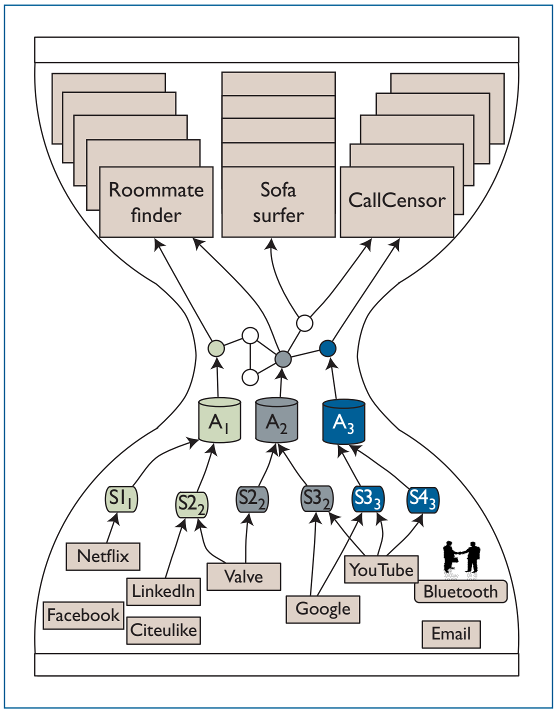

Figure 1. The social hourglass architecture. The architectural components in the center of the figure address the three requirements: collects input from diverse social signals, differentiating information based on interaction type and intensity; allows for user-controlled fusion of social signals; and provides a persistent social knowledge management service scalable with the number of users and social signals.

Usage Scenario

Suppose a user, Bob, installs a new application, SofaSurfer, that lets him tap his social relations and identify who in his social circle to ask for hosting while on a low-budget road trip. At installation, the application checks with Bob's aggregator to see which of the required and optional social sensors he's registered with. In this case, we'll assume Bob has accounts on Facebook, Skype, and Google (and uses the chat utilities on all three platforms); LinkedIn; and the Team Fortress 2 (TF2) game server. We'll also assume that previously deployed social sensors running on Bob's behalf are observing the social signals generated within the application contexts.

These social sensors report Bob's activity to SKS, subject to a personalized filter stored and applied by his aggregator. For example, let's say the IM activity is aggregated into a value recorded on SKS that gives more weight to Google Chat than to Skype chat activity, the latter being used mainly for work interactions. This personalization filter can update on rare occasions — due to significant changes in activity patterns or to new social sensors deployed — or it can remain on a default setting that weighs all signals equally.

SofaSurfer will query SKS based on a geographical location and return a list of social contacts in the area, ordered by a social strength with Bob. Among these contacts, "friends of friends" might also appear, subject to how Bob has specified his application-related preferences. For example, the application might let Bob specify that friends of friends connected via TF2 interactions are trusted enough if they're also linked on Facebook.

If certain necessary social signals aren't available to a user, the application will identify them on installation. An out-of-band service lists the various implementations of sensors and their social signals, and users are prompted to agree to deploy missing sensors. The aggregator, as the user's personal assistant, provides the credentials for these sensors (such as, in Bob's case, the Facebook password to access wall posts). The aggregator deploys sensors based on where the social signal is (such as a Facebook application) or on user-controlled platforms (such as a TF2 sensor running on Bob's desktop. The user's cognitive load is determined by the level of sophistication desired for social inferences — from none for default, one-size-fits-all settings to relatively high for fully personalized.

Social Sensors

We define social sensors as applications that run on a particular user's behalf and record social relations from that user's perspective by observing one or more social signals. A social sensor's output includes the relationship's actors: an implicit ego and an explicit alter. Group relationships can easily be broken down into a collection of records of the form `[ego, alter`$_i$`]`. Social sensors also output

- the "type" of interaction, labeled to a predefined term hardcoded in the sensor implementation; and
- a numerical value that represents the interaction's intensity, measured in the domain-specific context.

Three observations can offer insights into social sensor design. First, sensors should be able to use the wealth of existing social signals produced by Internet forums, websites, blogs, OSN applications, or mobile phone sensors.[4,6] These social signals might not be always available; for example, mobile phones might be turned off or users might not be active on a website. The interaction rate in the signal stream also varies from user to user, subject to social preferences or domain-specific abilities (such as those in multiplayer gaming). Some signals, such as up/down votes on Reddit, might encode signed (positive/negative) data. Finally, some signals are likely to encode asymmetric, directed social interactions: having a comment up-voted is different from up-voting another's comment.

Second, sensors should be deployable in different environments and thus be lightweight and function with intermittent connectivity. The sensor type could dictate the deployment environment: for example, a sociometric badge must be colocated with the signal it records (the user's real-world activities), but an online gaming sensor might run on the game's host server or the player's workstation and thus access the social signal remotely. Depending on the deployment scenario, the sensor design is subject to various constraints related to resource consumption (a battery on a mobile device, for example, or CPU and network communication on remote servers when many sensors coexist) and intermittent connectivity (as when mobile phones can be outside coverage areas or switched off).

Third, signals are domain-specific, and their interpretation is implementation-specific. Consequently, some sensors might operate on more than one social signal: for example, a sensor built to analyze IM conversations[8] might process social signals from GoogleTalk, Skype, AIM, and Facebook. Some sensors might consume the same social signal and yield different outputs. Sensors should be able to produce output with multiple labels; a single IM sensor mining a conversation, for example, might produce output labeled both "homework" and "sports." Finally, sensors might report information at different levels of sophistication: some might report the number of times two users communicated via email about a certain topic, whereas others might report an already processed relationship strength, using an existing model.[10]

These observations converge on requirements related to scalability, availability, versatility, and impact on the deployment environment. The HTTP protocol is a good candidate for communication between sensors and social signals because many sophisticated social signals are already being exported through rich Web interfaces or APIs, including those for Facebook, Reddit, Steam, and Twitter. Because sensors can't be expected to have 100 percent availability, they should default to pulling data from social signals. This design decision also dovetails with the choice of HTTP as the default transport protocol: HTTP is a "pull" technology at heart. While leaving the decision of when and what data to sample from social signals to sensors is good design, it doesn't preclude social sensor developers from building more complicated publish-subscribe mechanisms.

The Personal Aggregator

Each sensor running on ego's behalf sends a message of the form `[alter, label, weight]` to ego's aggregator. This aggregator refines the weights and labels sent by sensors according to users' personal preferences, composing and fusing the output of relevant sensors and sending refined output to the SKS. This separation of concerns lets social sensors manage data collection while the aggregator tunes that data to the user's personal style. Users interact with their sensors through the aggregator, which thus serves as a centralized point of control for the ecosystem of user-deployed social sensors.

The aggregator has three main responsibilities: sensor setup, personalized aggregation of social data received from the sensors, and identity management.

Sensor Setup

The aggregator provides a centralized management point for users and a minimal configuration interface for sensors. Users instantiate sensors via their aggregator in a process that uniquely identifies the sensor, sets up communication channels (including encrypted communication between the sensor and aggregator), and indicates any sensor-specific configuration settings (such as username/password to access a given signal).

Personalized Aggregation of Social Data

Aggregators can combine sensor outputs to provide greater accuracy. Users should be able to configure the aggregator to weigh one sensor's output more than another, to transform labels, to ignore a signal altogether when other signals are present, and various other combinator functions. For example, one such function could help distinguish between relevant relationships and familiar strangers: a calendar sensor would tag an interaction as "gym," and — in the absence of other types of interactions with alter — ego's aggregator would ignore the fact that alter and ego are in (Bluetooth-reported) physical proximity for half an hour twice a week.

Identity Management

Ego's identity is known to the aggregator in all its various incarnations of different user names on different systems. But this isn't the case for alter's identity, which can be recorded under different user names from sensors reporting on different social signals. The aggregator acts as the user's traditional address book: the user is responsible for connecting an alter's different identities based on out-of-band, real-life information. When such information doesn't exist

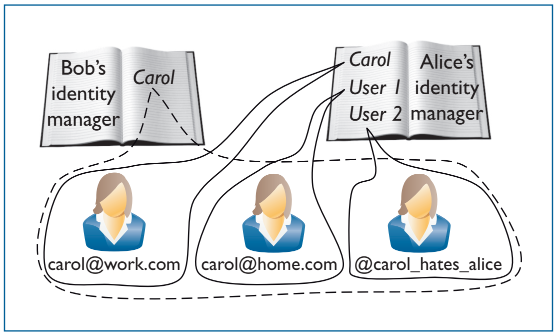

Figure 2. Sample identity management for Bob's and Alice's aggregators. Bob has linked all three of Carol's identities and can thus collate interactions with those identities as interactions with Carol. Alice, however, has only Carol's work email linked, and thus interactions via Carol's personal email or Twitter account are treated as interactions with different individuals.

(as in real life), a Sybil situation — where different individuals see different identities of the same alter — is possible.

In Figure 2, Carol is identified in Bob's aggregator by a work email, a personal email, and a Twitter username "@carol_hates_alice." In Alice's aggregator, however, Carol is identified only by her work email. Alice is unaware that "@carol_hates_alice" is actually Carol from work, and thus Alice's aggregator treats sensor output with Carol as different alters, depending on whether the interaction was via Carol's work email, personal email, or Twitter. Bob's decision to reveal the real-life identity of "@carol_hates_alice" to Alice remains his own personal decision, unmediated by the software infrastructure.

The aggregator's identity management functionality can fuse sensor outputs of users across social signals, but it can also filter noisy interactions — such as ignoring interactions with alters without a previously known identity.[11]

Social Knowledge Service

The aggregator sends processed social information to the SKS, which records it in a social graph. SKSs can be designed in any architecture. A centralized architecture can better control information access and accurately mine the social graph, but it can have issues with scalability and earning users' trust. Alternatively, an SKS can run directly on user mobile phones.[12,13] In this case, the aggregator is tightly coupled with the social data management service. However, a request that traverses a larger neighborhood in the graph — for example, one that's three hops away from ego — can be accurately answered only if everyone in a radius of three from ego has their mobile phones on and connected at the time of the request.

More persistent decentralized SKS architectures are based on peer-to-peer (P2P) infrastructures.[1,14,15] We previously proposed such a service, Prometheus,[1] a P2P service for user-controlled social data management. Prometheus stores social data on user-contributed trusted peers. The trusted peer on which a given user's data resides is selected based on the social relationship between said user and the trusted peer's owner. An additional benefit is that queries between socially close users can be mined locally on the peer that stores their data.

Prometheus maintains a weighted, directed social multigraph in which multiple edges can connect two users, each edge labeled with the type of social interaction it represents and assigned a weight that corresponds to the interaction's intensity. Prometheus treats the social graph as a first class data object and exposes an API to applications while providing fine-grained, user-selected restrictions on what data the applications can access. The API implements a basic set of distributed social-inference functions, allowing developers to test whether two users are directly connected in the social graph; retrieve a user's n-hop neighborhood; and retrieve a list of relations within a given geographic distance from a user. Each of these functions can filter the results based on a particular interaction label and a minimum weight. The API also provides more sophisticated inferences, such as a measure of social strength between two users.

We prototyped and evaluated Prometheus on a large-scale Internet deployment, with both real and simulated social application workloads. Our results demonstrate the feasibility of mining the distributed social graph for complex social-inference queries.

A Proof-of-Concept Social Sensor

Implementing a social sensor and testing it on a real signal forces us to deal with several of the challenges described earlier. We chose to implement a social sensor that reads the in-game interactions among players of TF2, an online, team-based, first-person shooter game. We focused on this type of social signal for several reasons. First, in-game behavior closely mirrors real-world social behavior.[16] Thus, a social sensor sensing in-game interactions could identify patterns relevant outside the specific domain in which they were produced. A byproduct is that such a sensor might be of interest to sociologists. Second, recent studies show that social relationships are a critical aspect of an enjoyable online gaming experience and that such relationships persist beyond the confines of the gaming environment.[17] Consequently, the output of a social sensor deployed in a gaming environment can be useful to applications that transcend the virtual world of gaming.

Implementation Conditions

We implemented a social sensor for TF2 under the following conditions. First, the TF2 sensor reads the social signals remotely; the game-hosting servers are already under load and can't run sensors on hosted players' behalf. Consequently, we deployed the TF2 sensor as a stand-alone application or a game plug-in running on the user device (desktop, mobile phone, and so on).

Second, a sensor can only access the social signal of the user it represents — that is, it can read only the events in which the user is involved. This is particularly important for gaming, where access to global state information can enable cheating by providing information about events a player can't directly observe. Third, because the social signal is made available by the third-party server that hosts the game, the sensor has no control over how long that signal is stored. A server might decide to maintain the social signals in a circular buffer or to erase the recorded signals periodically. Finally, the social signal we analyze has multiple types of events encoded in it, each with a different production rate.

The Team Fortress 2 Social Signal

The critically acclaimed and highly popular TF2 game supports up to 32 simultaneous players on a server and is played on thousands of servers at any time. TF2's basic premise is that two teams are in constant struggle for world domination and try to control specific points on a map or simply eliminate all of the opposing team's members. When players connect to a server, they first choose a team. Players earn points for various in-game actions, including killing or assisting in killing the opposing team's members.

We've acquired detailed game-play logs of a 32-simultaneous-player server located in California. Our logs span just over two months — from 1 April 2011 to 8 June 2011 — and consist of various events involving 10,354 players. Some of these events denote interactions, so we refer to raw data as *events* and to our implementation-specific interpretation of that raw data as *interactions*. We consider five representative in-game interactions as the basis of the TF2 social signal:

- *PlayerKilled*. One player is killed by another.
- *TriadicKillAssist*. One player killed a second player with help from a third player.
- *Domination*. An award is given to a player who killed or assisted in killing another player four consecutive times (without being killed by the other player).
- *Revenge*. A player gets revenge by killing his dominator.
- *PlayerExtinguished*. A player extinguished the flames consuming one of his teammates.

An analysis of TF2's social signal provided a better understanding of the domain-specific requirements for the TF2 sensor design. First, as expected, the frequency of gaming events varies over time and population and is subject to daily patterns and player skill. For example, roughly 7 percent of the players joined the game, disconnected before interacting with anyone, and never returned; the majority of players averaged about four events per minute; and the top 19 most-active players averaged more than eight events per minute.

Second, the number of interactions between players varies dramatically, likely representing different relationship strengths. Moreover, these relationships as seen in the in-game interaction logs don't correlate with the declared relationships on the Steam Community, an OSN for gamers that includes those in the TF2 logs: 40 percent of declared friends (6,020 out of 15,383 pairs) interacted at least once; fewer than 1 percent of nondeclared pairs (480,788 pairs) interacted; and yet the number of nondeclared friends who interacted is 80 times larger than that of declared friends who interacted. This observation confirms previous studies[18] and stresses the importance of considering interactions (collected by social sensors) instead of declared relationships.

Third, different components of the same social signal have different frequency patterns. *Domination* and *Revenge* events were at least four times less frequent than *PlayerKilled* and *TriadicKillAssist* events, and *Revenge* events happened less often than *Domination* events. In addition, *TriadicKillAssist* events occurred at a rate close to the aggregated rate of all the other event types.

Finally, the output of the TF2 social sensor is not only domain-specific, but can also be dictated by the application that will consume it. For example, one possible output is a relationship strength that accounts for the overall number of interactions with another user. Another option is to distinguish between cooperative and antagonistic interactions, thus returning two different types of relationships, each with its own strength. As we now describe, we took the latter approach.

The Team Fortress 2 Social Sensor

The TF2 sensor periodically polls the user's social signals and outputs to the user's aggregator a series of `[`$\texttt{alter}_i$`, `$\texttt{label}_k$`, `$\texttt{weight}_{i,k}$`]` tuples, where $\texttt{alter}_i$ represents each player with which the user interacted. The polling and reporting frequencies are configurable on a per-user basis. A sensor can return multiple labels, which are predefined and domain-specific. Our TF2 implementation returns two labels: "foe" marks antagonistic interactions (that is, *PlayerKilled*, *Domination*, and *Revenge* events); and "friend" marks cooperative interactions (such as *TriadicKillAssist* and *Extinguish* events).

Our design decisions are based on three objectives:

- reduced sensor load on the server that provides the social signal,
- meaningful weights on the foe and friend relationships, and
- support for configurable output based on the application-specific requirements communicated through the user aggregator.

To keep the sensor load low on the server that provides the social signals, the sensor must adapt its polling rate to each signal's variable state in an "online" fashion. This condition

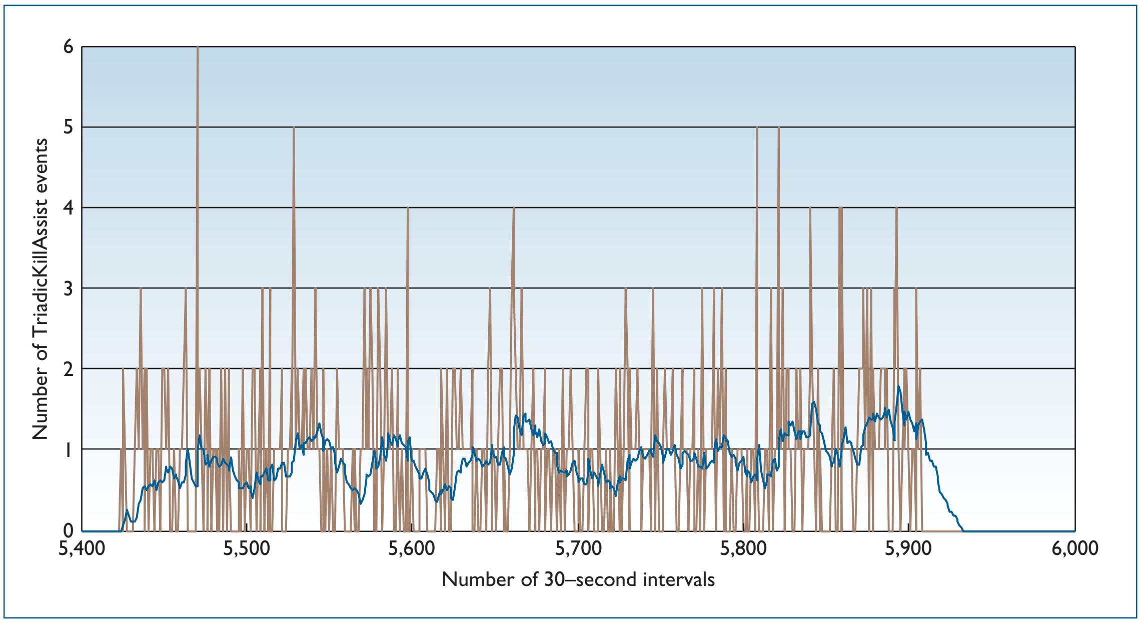

Figure 3. Example prediction for the normalized least-mean square (NLMS) adaptive filter technique. This shows the measured (brown) and predicted (dashed blue) TriadicKillAssist *signal for a five-hour time frame, using the NLMS technique with the following parameter values: filter length* H $=$ *25; step size* $\mu = 0.01$*; and interval duration* k $=$ *30 seconds.*

disqualifies learning-based prediction techniques that require considerable data history and time to train. To predict the social signal amplitude quickly and accurately over an interval of duration *k* (that is, the number of events generated), we compared four well-established adaptive sampling techniques over a range of parameters:

- single-neuron linear network (NN),
- least-mean squares (LMS) adaptive finite-impulse response (FIR) filter,
- normalized least-mean square (NLMS) adaptive filter, and
- recursive least squares (RLS) adaptive FIR filter.

All techniques demonstrate that, with only 5 to 15 minutes of history, they accurately predict the social signal's amplitude and thus can successfully adjust their polling rate to limit the load on the signal-producing server. In terms of speed convergence, RLS and NN were the slowest, and NLMS and LMS were equally fast. Figure 3 shows how the NLMS technique predicts the measured signal containing the *TriadicKill-Assist* events for a five-hour time period. This technique accurately predicts no activity in the social signal at the end of the interval, which lets the social sensor reduce its polling rate. The overall mean-squared error for predicting this signal over the two-month period is 0.0488 events.

To quantify foe/friend relationships, we used the percentile ranking of the number of friend/foe interactions that ego had with alter. That is, an output from ego's sensor `[alter`$_n$`, foe, 0.5]` indicates that ego and `alter`$_n$ had at least as many foe interactions as 50 percent of all alters that ego had foe interactions with. The benefits of this approach are multifold. A percentile ranking normalizes to the number of players interacted with, and thus varies less with the number of interactions. This dampens the effects of skill gaps among players. If we normalized to the total number of interactions — that is, the number of `ego-alter`$_n$ interactions over the total number of interactions ego had, the difference between the weights for `alter`$_n$ and `alter`$_m$ would depend heavily on their relative difference in skill level (or, more specifically, their rate of event production). This is a problem because significant differences exist between highly skilled and average players; a percentile ranking permits a more comparable output between users and over time.

We configure the frequency of output to the aggregator based on application-specific requirements. When the user installs a socially aware application, it specifies its requirements (including the label it operates on and the time-resolution it expects from sensor data) to the aggregator. The aggregator then tunes sensors to meet these requirements. For example, an application that helps a user select a team to join might be interested in the most recent relationships using high friendship weights as a measure of "playing chemistry," while the SofaSurfer application might use a longer history of friend and foe interactions (in addition to declared friendship in the Steam Community or Facebook) as a measure of trust.

Lessons for Sensor Design and Implementation

Our proof-of-concept implementation provides several lessons applicable to general social sensors.

First, the frequency of the interaction-based social signal varies significantly across the population and over time, subject to daily patterns and user lifestyles. Therefore, polling and reporting frequency should be configurable and adaptable on a per-user basis. In our study, established adaptive sampling techniques with minimum time and system overhead proved appropriate.

Second, by understanding the social signal, the sensor designer can identify different signal components with different frequency patterns and select the most representative to observe. Sensor designers can exploit such differentiations between signal components and apply different policy in polling each signal component, or even just select the signal's most representative component and ignore the rest. In this case, for example, *Dominations* are a coarse-grained representation of *PlayerKilled* events, so the TF2 sensor could be configured to sample only the former for highly active players.

Finally, the sensor output must be the outcome of a normalizing technique that adjusts to the social signal input and makes the edge weights' outputs comparable across users and over time. In addition, the output format and semantic can be correlated to the applications that will consume the sensor signal.

Our previous work described in detail the upper part of the social hourglass: a social knowledge management service and proof-of-case social applications. The work we present here draws the big picture, focusing on social sensors and the personal aggregator in the infrastructure's lower half.

We learned various lessons in designing and implementing the full social hourglass infrastructure. Starting from the architecture's bottom layer, it became evident that sensors are domain-specific. Their design and deployment must mitigate trade-offs between information accuracy, privacy concerns, and resource constraints. The personal aggregator reduces the problem space from billions of sensors mining an ocean of social signals to the number of users in the system. Furthermore, the social knowledge service in our design addresses the scalability challenge by aggregating multiple users' social information on a computing node connected to other user-contributed nodes in a self-organizing P2P infrastructure. The API we propose exposes basic social inferences on an augmented social graph, but we're expanding it by using a domain-specific language that will let us compose our basic API into more sophisticated social inferences.

Protecting the privacy of aggregated social information is a key issue. Unfortunately, this problem already exists on social aggregators such as Spokeo, which provide (for a fee) an unsettling collection of information — including age, property value, address, and family members — from various online services. Our architecture offers user control at multiple points (sensors, aggregator, and social knowledge service), giving users fine-grained personal control of how their social data is transmitted, stored, and used. Specifically, users approve sensor deployment, which can occur either on the data source's location (thus offering no additional privacy exposure) or on user-controlled devices. Further, users can select data granularity, and data is encrypted when transmitted and stored on SKS. In addition, we add another layer of privacy by letting users correlate alters away from the prying eyes of a centralized authority, as well as by letting them transform data before making it available to applications. At minimum, our infrastructure can make smarter use of the social information already available online, but personalized by the user to reduce noise. Ultimately, a rich and flexible API and practical privacy control is what applications and users need to truly exploit the potential of the tide of social information now exposed on the Web, through many mobile devices and pervasive connectivity.

The decoupled social hourglass architecture provides increased resilience to faulty components. The aggregation of different social signals can also alleviate the damage of erroneous signals and even of manipulative data. For stronger resilience, more intelligence could be placed in the SKS design and its API's sophistication. For example, maintaining a directed interaction graph in SKS distinguishes one-way spamming from a two-way relationship; social inferences that consider the social interaction's direction (that is, who emails whom) also limit the damage.[19]

However, many engineering and conceptual challenges must be addressed for wide adoption, ranging from defining community-accepted standards for social sensor output to dealing with data heterogeneity due to various implementations of the same social sensor. Sophisticated social sensors could sense multiple signals and output multiple types of inferred relationships, each with its own strength. Containing the damage that a faulty social sensor could inflict is vital for the usability of social applications and the long-term health of a social data management infrastructure.

Acknowledgments

The US National Science Foundation supported this research under grants CNS 0952420 and CNS 0831785.

References

1. N. Kourtellis et al., "Prometheus: User-Controlled P2P Social Data Management for Socially-Aware Applications," *Proc. 11th Int'l Middleware Conf.*, LNCS 6452, Springer, 2010, pp. 212–231.
2. J.S. Kong et al., "Collaborative Spam Filtering Using E-Mail Networks," *Computer*, vol. 39, no. 8, 2006, pp. 67–73.
3. M. Rahman, A. El Saddik, and W. Gueaieb, "Building Dynamic Social Network from Sensory Data Feed," *IEEE Trans. Instrumentation and Measurement*, vol. 59, no. 5, 2010, pp. 1327–1341.
4. E. Miluzzo et al., "Sensing Meets Mobile Social Networks: The Design, Implementation and Evaluation of the Cenceme Application," *Proc. ACM Conf. Embedded Network Sensor Systems*, ACM Press, 2008, pp. 337–350.
5. H. Lu et al., "Energy Efficient Unobtrusive Speaker Identification on Mobile Phones," *Proc. 9th Int'l Conf. Pervasive Computing*, LNCS 6696, Springer, 2011, pp. 188–205.
6. D.O. Olgun, P.A. Gloor, and A.S. Pentland, "Capturing Individual and Group Behavior with Wearable Sensors," *Proc. AAAI Spring Symp. Human Behavior Modeling*, AAAI, 2009, pp. 68–74.
7. J.M. Hellerstein and D.L. Tennenhouse, *Searching for Jim Gray: A Technical Overview*, tech. report UCB/EECS-2010-142, EECS Dept., Univ. of California, Berkeley, Dec. 2010.
8. P. Singla and M. Richardson, "Yes, There Is a Correlation: From Social Networks to Personal Behavior on the Web," *Proc. 17th Int'l Conf. World Wide Web*, ACM Press, 2008, pp. 655–664.
9. J.-P. Onnela et al., "Structure and Tie Strengths in Mobile Communication Networks," *Proc. Nat'l Academy of Sciences*, vol. 104, no. 18, 2007, pp. 7332–7336.
10. R. Xiang, J. Neville, and M. Rogati, "Modeling Relationship Strength in Online Social Networks," *Proc. 19th Int'l Conf. World Wide Web*, ACM Press, 2010, pp. 981–990.
11. T. Lovett et al., "The Calendar as a Sensor: Analysis and Improvement Using Data Fusion with Social Networks and Location," *Proc. 12th ACM Int'l Conf. Ubiquitous Computing*, ACM Press, 2010, pp. 3–12.
12. A.-K. Pietiläinen et al., "MobiClique: Middleware for Mobile Social Networking," *Proc. 2nd Workshop Online Social Networks*, ACM Press, 2009, pp. 49–54.
13. A. Toninelli, A. Pathak, and V. Issarny, "Yarta: A Middleware for Managing Mobile Social Ecosystems," *Proc. 6th Int'l Conf. Advances in Grid and Pervasive Computing*, IEEE Press, 2011, pp. 209–220.
14. S. Buchegger et al., "PeerSoN: P2P Social Networking: Early Experiences and Insights," *Proc. 2nd Workshop on Social Network Systems*, ACM Press, 2009, pp. 46–52.
15. K. Graffi et al., "LifeSocial.KOM: A Secure and P2P-Based Solution for Online Social Networks," *Consumer Communications and Networking Conf.*, IEEE Press, 2011, pp. 554–558.
16. M. Szell and S. Thurner, "Measuring Social Dynamics in a Massive Multiplayer Online Game," *Social Networks*, vol. 32, no. 4, 2010, pp. 313–329.
17. Y. Xu et al., "Sociable Killers: Understanding Social Relationships in an Online First-Person Shooter Game," *Proc. ACM Conf. Computer Supported Cooperative Work*, ACM Press, 2011, pp. 197–206.
18. C. Wilson et al., "User Interactions in Social Networks and Their Implications," *Proc. 4th ACM European Conf. Computer Systems*, ACM Press, 2009, pp. 205–218.
19. J. Blackburn, N. Kourtellis, and A. Iamnitchi, "Vulnerability in Socially-Informed Peer-to-Peer Systems," *Proc. 4th ACM EuroSys Workshop Social Network Systems*, ACM Press, 2011; doi:10.1145/1989656.1989663.

Adriana Iamnitchi is an associate professor in the Department of Computer Science and Engineering at the

University of South Florida. Her research interests are in distributed systems, with an emphasis on designing, implementing, and experimenting with services and applications for large-scale networked systems; this work spans peer-to-peer networks, data management in distributed scientific collaborations, and social and cloud computing. Iamnitchi has a PhD in computer science from the University of Chicago. She's a member of IEEE and the ACM. Contact her at anda@cse.usf.edu.

Jeremy Blackburn is a PhD candidate in the Department of Computer Science and Engineering at the University of South Florida. His research interests are in large-scale distributed systems, focusing on measurement, analysis, and design of social network systems. Blackburn has an MSc in computer science from the University of South Florida. Contact him at jhblackb@mail.usf.edu.

Nicolas Kourtellis is a PhD candidate in the Department of Computer Science and Engineering at the University of South Florida. His research interests include the design of socially aware distributed systems, social networks analysis, mobile computing, and socially aware applications and services. Kourtellis has an MSc in computer science from the University of South Florida and an MEng in electrical and computer engineering from National Technical University of Athens. He's a full member of the Tau Beta Pi, Eta Kappa Nu, and Phi Kappa Phi honor societies and a student member of IEEE and the ACM. Contact him at nkourtel@mail.usf.edu.