The Social Hourglass: an Infrastructure for Socially-aware Applications and Services

Adriana Iamnitchi, Jeremy Blackburn, Nicolas Kourtellis
Department of Computer Science and Engineering, University of South Florida, Tampa, FL, 33620
anda@cse.usf.edu, jhblackb@mail.usf.edu, nkourtel@mail.usf.edu

Abstract—This article proposes an infrastructure for enabling socially-aware applications and services. Following the inspiration of the hourglass architecture of the Internet, the article argues that a service that manages data from a variety of social sensors is likely to provide better accuracy in quantifying social relationships and better support for a variety of future social applications. We present an architecture that includes social sensors that capture and interpret social signals based on the interaction between two users, a personal aggregator of social information, a data management service that builds and maintains an augmented social graph, and a set of social inference functions as this service’s API for social applications. As a proof of concept, we build a social sensor for multiplayer online gaming interactions and draw a set of observations and lessons for future work.

II. THE SOCIAL HOURGLASS ARCHITECTURE

The abundance of social information exposed by recent Internet applications, such as social networking sites, has enabled the mining of information about the relationships between users to provide improved service. This approach has been exploited in various domains, most notably security and spam protection [5], emergency medical alerts [13], and recommendation systems like 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 declared friendship on Facebook or LinkedIn, or interaction-based, such as those inferred from recorded co-location traces [9], conversations [8], sociometric badges [10], or online gaming (Section VI). A vast diversity of 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, or phone call history.

Interaction-based social signals provide an unprecedented level of detail compared to the binary declared relationships typical of OSNs. First, they provide the opportunity to quantify the strength of the social relationship based on domain-specific metrics, such as quantity (e.g., phone call duration or number of characters in instant messaging exchanges) and frequency. Second, they convey more accurate information than declared relationships, which are generously created and rarely removed.

Given the opportunities of exploiting social information, we conjecture that it is time to move from the vertically integrated approach—in which one source of information is mined for a specific application—to a service infrastructure capable of synthesizing information from multiple types of social interactions and consequently capable of providing accurate and personalized support for a variety of social applications. Following the hourglass metaphor at the base of the Internet architecture, this infrastructure needs to 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 act as the enabler of a continuum of innovative social applications and services. For example, a couch surfing application could explore the social k-hop neighborhood of the requester for occasional hosting while traveling by selecting people with similar social interaction patterns, such as hiking or playing online games, both hob-
ties where relationships persist beyond the activity itself. A personal-cloud service can mine the implicit social incentives of a 2-hop neighborhood to recruit cycles for a computationally intensive search-and-rescue mission [4]. A context-aware phone-call filtering application (e.g., CallCensor [6]) may filter calls based on the social relationship between the user and caller, as well as people co-located with the user such that, for example, personal calls are automatically silenced during professional meetings but co-workers’ calls are let through.

An infrastructure that supports a variety of social applications would require the following:

- **Input from diverse social signals**, while differentiating information based on the type and intensity of social interactions. Exposing relationship type and strength allows applications to make use of not just the existence of a context-relevant relationship, but also to compare relationships. For example, the CallCensor application could be configured to accept calls from non-co-workers who have a strong social tie with the call recipient.

- **User-controlled fusion of social signals**. For some users, a social interaction during weekly online multi-user gaming may translate into trust outside the gaming context, for example, for occasional couch surfing when traveling. For others, a trusted social relationship may require a combination of more traditional interactions such as physical co-location, conversations over the phone, and sharing photographs from family vacations. The choice and the relative importance of the relevant social signals should be personalized and fully under the user’s control.

- **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 climbing partners for a trip by asking the climbing partners of her partners, 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.

The requirements stated above correspond loosely to the architectural components depicted in Figure 1. In this multi-layered architecture, the bottom layer encompasses the diversity of existing or future social signals. Without necessarily calling them such, many social signals, such as messaging [14], phone call patterns [11], or Facebook friendship declarations [18] have been collected and analyzed. While 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 specifically crafted to be consumed by an infrastructure like the one proposed here.

The next architectural layer consists of **social sensors**: applications running on behalf of a user on various platforms, such as mobile phones, web browsers, or 3rd party servers, that each analyzes 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 behalf of a user are then directed to the next layer, the personal aggregator. The personal aggregator, a trusted application typically running on a user-owned device, implements the fusion of multiple same activity social edges and the personalization of the user’s social relations.

Once personalized by the user, this digital representation of social relations is sent 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.

These architectural components support socially-aware applications as in the following example. Suppose a user installs a new application, SofaSurfer, that allows him to tap his social relations and identify who in his social circle to ask for hosting while on a low-budget road trip. At installation, the application checks with the user’s aggregator which of the required and optional social sensors the user is registered with. Assume the user has accounts on Facebook, Skype, Google (and uses the chat utilities on all these platforms), LinkedIn, and is active on a Team Fortress 2 (TF2) game server, and all the corresponding social signals are observed by previously deployed social sensors running on behalf of the user. These sensors consequently report the user’s activity to SKS, subject to a personalized filter stored and applied by the user’s aggregator. For example, the instant messaging activity is aggregated into a value recorded on SKS that gives more weight to Google Chat than to Skype chat activity, the latter being mainly used for work interactions. This personalization filter can be updated rarely—due to significant changes in activity patterns or to new social sensors deployed—or can be left to a default setting that weighs equally all signals. SofaSurfer will query SKS based on a geographical location and return a list of social contacts in the area, ordered by a social strength with the user. Among contacts can also be “friends of friends”, subject to user-specified application-related preferences. For example, the application might allow the user to specify that friends of friends connected via TF2 interactions are trusted enough if also linked on Facebook.

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

III. SOCIAL SENSORS

We define social sensors as applications that run on behalf of a particular user and record social relations from the perspective of that user by observing one or more social signals. The output of a social sensor includes the actors of the relationship: an implicit ego and an explicit alter. Group relationships can easily be broken down into a collection of records of the form [ego, alter,]. 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 intensity of that interaction measured in the domain-specific context.

The following three observations may inform the design of social sensors. First, sensors should be able to make use of the wealth of existing social signals produced by internet forums, websites, blogs, OSN applications or mobile phone sensors [10], [9]. These social signals may not be always available: for example, mobile phones may be turned off or users may not be active on a website. The rate of interactions in the signal stream is variable from user to user, subject to social preferences or domain-specific abilities (e.g., in multi-player gaming). Some signals, such as up/down votes on Reddit, may encode signed (positive/negative) data. And 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 a variety of environments, and consequently they should be lightweight and function with intermittent connectivity. The deployment environment may be dictated by the type of sensor: for example, a sociometric badge must be co-located with the signal it records (the user’s real world activities) but an online gaming sensor may run on the server that hosts the game, or on 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 (battery on mobile devices, CPU and network communication on remote servers when many sensors may coexist) and intermittent connectivity (mobile phones can be outside coverage areas or switched off).

Third, signals are domain-specific, and their interpretation is implementation-specific. Consequently, some sensors may operate on more than one social signal: for example, a sensor built to analyze IM conversations [14] may 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: e.g., a single IM sensor mining a conversation could produce output labeled both “homework” and “sports”. And finally, sensors might report information of different levels of sophistication: some may report the number of times two users communicated via email about a certain topic, while others might report an already processed relationship strength, using models like the ones presented in [18].

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: e.g., Facebook, Reddit, Steam, and Twitter. Because sensors cannot be expected to have 100% 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 also does not preclude more complicated publish/subscribe mechanisms from being built.

IV. THE PERSONAL AGGREGATOR

Each sensor running on behalf of ego sends a message of the form [alter, label, weight] to ego’s aggregator. The aggregator is charged with refining 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 allows social sensors to be in charge of collecting data and allows the aggregator to tune that data to the user’s personal style. Users interact with their sensors through the aggregator, and thus the aggregator serves as a centralized point of control for the eco-system of user-deployed social sensors.

The aggregator has three main responsibilities:

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 any sensor-specific configuration settings (e.g., username/password to access a given signal).

Personalized aggregation of social data received from sensors: Sensor outputs can be combined to provide greater accuracy. Users should be able to perform operations such as weighing the output from one sensor more than another, transforming labels, ignoring 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 will ignore 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 is not 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 the different identities of an alter based on out-of-band, real-life information. When such information does not exist, as in real life, a sybil situation 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 only identified by her work email. Alice is unaware that “@carol_hates_alice” is actually Carol from work, and thus her 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, not mediated by the software infrastructure.

While the aggregator’s identity management functionality can be used to fuse sensor outputs of users across social signals, it can also be used to filter noisy interactions, e.g., to ignore interactions with alters without a previously known identity as in [7].

V. Social Knowledge Service

Output from users’ aggregators is sent to the SKS, which records it in a social graph. SKSs may be designed in any architecture: a centralized architecture has the advantage of better controlling the access to information and accurately mining the social graph, but it has the challenges of scalability and earning users’ trust. Alternatively, such a service can run directly on user mobile phones [12], [16]: the aggregator is then tightly coupled with the social data management service. However, a request that traverses a larger neighborhood in the graph, e.g., 3 hops away from ego, can be accurately answered only if everybody in a radius of 3 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 infrastructures [2], [6], [3].

We previously proposed such a service, Prometheus [6], a peer-to-peer service for user-controlled social data management. Prometheus stores social data on user-contributed trusted peers. Which trusted peer a given user’s data resides on is a function of the social relationship between said user and the owner of the trusted peer. An additional benefit is that queries between socially close users can be mined locally on the peer storing their data. Prometheus maintains a weighted, directed social multi-graph, where multiple edges may connect two users, each edge labeled with the type of social interaction it represents and assigned a weight that corresponds to the intensity of that interaction. 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 can be accessed. 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 the n-hop neighborhood of a user; or retrieve a list of relations within a given geographic distance from a user. All 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, and demonstrated the feasibility of mining the distributed social graph for complex social inference queries.

VI. Proof-of-Concept Social Sensor: Implementation and Lessons

Implementing a social sensor and testing it on a real signal forces us to deal with several of the issues discussed in Section III. We chose to implement a social sensor that reads the in-game interactions between players in a team-based, first-person shooter online video game. We focused on this type of social signal for various reasons. First, in-game behavior closely mirrors real-world social behavior [15]. Thus, a social sensor sensing in-game interactions could identify patterns relevant outside the specific domain where they are produced. A byproduct is that such a sensor might be of interest to sociologists. Second, recent studies showed that social relationships are a critical aspect of an enjoyable online gaming experience and that these relationships persist outside of the confines of the gaming environment [19]. As a consequence, the output of a social sensor deployed in a gaming environment can be of use to applications that transcend the virtual world of gaming.

We implement a social sensor for TF2, a popular multiplayer online video game, under the following conditions. First, the TF2 sensor is reading the social signals remotely: the game-hosting servers are already under load and cannot be expected to run sensors on behalf of hosted players. Consequently, the TF2 sensor is deployed as a standalone application or a game plugin running on the user device (desktop, mobile phone, etc). Second, a sensor can only access the social signal of the user it represents, that is, it can strictly read only the events in which the user was involved. This is particularly important for gaming, where access to global state information can enable cheating by providing information about events not directly observable by a player. Third, because the social signal is made available by the 3rd party server that hosts the game,
there is no control over how long the social signal is stored. A server may 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 rate of production.

A. The Team Fortress 2 Social Signal

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

We have acquired detailed game play logs of a 32-simultaneous player server located in California. Our logs span just over 2 months from April 1 to June 8, 2011, and consist of various events involving 10,354 players. Some of these events denote interactions, and thus 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:

1) **PlayerKilled**: One player was killed by another.
2) **TriadicKillAssist**: One player killed a second player with the help of a third player.
3) **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).
4) **Revenge**: A player gets revenge by killing his dominator.
5) **PlayerExtinguished**: A player extinguished the flames consuming one of his teammates.

An analysis of the social signal in TF2 provided a better understanding of the domain-specific requirements for the TF2 sensor design. First, as expected, the frequency of gaming events is highly variable over time and population and subject to daily patterns and to players’ skills. For example, about 7% of the players joined the game, disconnected before interacting with anyone, and never returned; the majority of the players averaged about 4 events per minute; the top 19 most active players averaged over 8 events per minute.

Second, the number of interactions between players varies dramatically, likely representing very different relationship strengths. Moreover, these relationships as seen in the game interaction logs are not correlated with the declared relationships on the Steam Community, an online social network for gamers that includes the gamers in the TF2 logs: 40% of declared friends (6,020 out of 15,383 pairs) interacted at least once; fewer than 1% of non-declared pairs (480,788 pairs) interacted; but the number of non-declared friends who interacted is 80 times larger than that of declared friends who interacted. This observation confirms previous studies [17] 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 4 times less frequent than PlayerKilled and TriadicKillAssist events, and Revenges happened less often than Dominations. In addition, TriadicKillAssist events occurred at a rate close to the aggregated rate of all the other event types.

And 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. In the following we took the latter approach.

B. The TF2 Social Sensor

The TF2 sensor periodically polls the social signal of its user and outputs to the user’s aggregator a series of \([alter_{i}, label_{k}, weight_{i,k}]\) tuples, where \(alter_{i}\) represents each player with which the user interacted. The frequencies of polling as well as reporting are configurable on a per user basis. Multiple labels can be returned: they are pre-defined and domain-specific. Our TF2 implementation returns two such labels: “foe” marks antagonistic interactions, i.e., PlayerKilled, Domination, and Revenge events; and “friend” marks cooperative interactions, such as TriadicKillAssist and Extinguish events.

Our design decisions are imposed by the following objectives: i) reduced sensor load on the server that provides the social signal; ii) meaningful weights on the foe and friend relationships; iii) 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 the variable state of each signal in an “online” fashion. This condition disqualifies learning-based prediction techniques that require a large amount of data history and time to train. To predict quickly and accurately the social signal amplitude over an interval of duration \(k\) (i.e., 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 5–15 minutes of history they accurately predict the amplitude of the social signal 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 performed similarly fast. Figure 3 demonstrates the prediction of the measured signal with the NLMS technique for the signal containing the TriadicKillAssist events and for a 5-hour time period. Note that this technique accurately predicts no activity in the social signal at the end of the interval, which allows the social sensor to reduce its polling rate. The overall mean squared error for predicting this signal over the 2-month period is 0.0488 events.
For the quantification of the foe/friend relationships we used the percentile ranking of the number of friend/foe interactions that \( ego \) had with \( alter \). I.e., an output from \( ego \)'s sensor of \([ alter_n, foe, 0.5 \)] indicates that \( ego \) and \( alter_n \) had at least as many foe interactions as 50% of all alters that \( ego \) had foe interactions with. The benefits of this approach are manifold. A percentile ranking normalizes to the number of players interacted with, and thus varies less with the number of interactions. This is important as it dampens the effects of skill gaps between players. If we normalized to the total number of interactions, i.e., the number of \( ego-alter_n \) interactions over the total number of interaction \( ego \) had, the difference between the weights for \( alter_n \) and \( alter_m \) would be heavily dependent on their relative difference in skill level (or more specifically their rate of event production). This is a problem because there are significant differences between highly skilled players and the average player. Related to this, a percentile ranking allows for a more comparable output between users and over time.

Frequency of output to the aggregator is configured 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 application 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.

![Graph](image-url)

**Fig. 3.** Measured (blue) and predicted (dashed red) TriadicKillAssist signal for a 5-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.

**C. 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’s lifestyle. Therefore, the frequency of polling and reporting should be configurable and adaptable on a per user basis. Established adaptive sampling techniques with minimum time and system overhead proved to be appropriate in our study.

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. Therefore, sensor designers can exploit such differentiations between signal components and could apply different policy in the polling of each signal component, or even just select the most representative component of the signal, ignoring the rest. For example, while **Dominations** are a coarse-grained representation of **PlayerKilled** events, the **TF2** sensor could be configured to sample only the former for highly active players.

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

**VII. Summary and Discussions**

This paper introduces the social hourglass infrastructure that can mediate between socially-aware applications and the many social signals they can potentially consume. Our previous work described in detail the upper part of the social hourglass: a social knowledge management service and proof-of-case social applications. In this work we draw the big picture and focus on the lower half of the infrastructure, namely social sensors and the personal aggregator.

We learned various lessons in the process of designing and implementing the full social hourglass infrastructure. Starting from the bottom layer of the architecture, it became evident that sensors are domain-specific. Their design and deployment need to mitigate tradeoffs between the accuracy of information, 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 peer-to-peer infrastructure. The API we proposed exposes basic social inferences on an augmented social graph, but we are currently working on expanding it using a domain-specific language that can allow the composition of our basic API into more sophisticated social inferences.

An important issue is the privacy protection of aggregated social information. Unfortunately, this problem already exists on social aggregators such as Spokeo, who provide, for a fee, an unsettling collection of information from various online services, including age, property value, address, and family members. Our architecture provides for user control at multiple points (sensors, aggregator, social knowledge service), allowing for fine-grained personal control of how social data is transmitted, stored and used. Specifically, sensor deployment is under user’s approval and sensors are either deployed on the location of the data source (thus, with no additional privacy exposure) or on user-controlled devices; the user can select data granularity; data is encrypted when transmitted and when
stored on SKS. In addition, another layer of privacy is added by allowing users to correlate alters away from the prying eyes of a centralized authority, and by providing the ability to transform data before making it available to applications. At minimum, this infrastructure can provide benefits by making smarter use of the social information already available online, but personalized by the user to reduce noise. Ultimately, a rich and flexible API together with practical privacy control is what applications and users need for truly exploiting 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 an erroneous signal or even manipulative data. For stronger resilience, more intelligence can be placed in the design of SKS as well as the sophistication of its API: for example, maintaining a directed interaction graph in SKS distinguishes one-way spamming from a two-way relationship; social inferences that consider the direction of the social interaction (i.e., who emails whom) also limit the damage [1].

However, there are many engineering and conceptual challenges left to 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 can inflict is vital for the usability of social applications and the long term health of a social data management infrastructure.

ACKNOWLEDGEMENTS

This research was supported by the National Science Foundation under Grants No. CNS 0952420 and CNS 0831785.

REFERENCES


Nicolas Kourtellis is a PhD Candidate in the Department of Computer Science and Engineering at 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. He received a MSc in Computer Science from USF in 2008 and a Diploma (MEng) in Electrical and Computer Engineering from NTUA in 2006. He is a full member of the honor societies ΤΒΠ, ΗΚΝ and ΦΚΦ, and a student member of the IEEE and ACM.