Enabling social networking in ad hoc networks of mobile phones

Emre Sarigöl[1] Oriana Riva[1] Patrick Stuedi[2] Gustavo Alonso[1]

[1] Systems Group, Department of Computer Science, ETH Zurich

[2] Microsoft Research

{emres,riva,alonso}@inf.ethz.ch, pstuedi@microsoft.com

ABSTRACT

This demonstration presents AdSocial, a software platform supporting social network applications in ad hoc networks. AdSocial targets small-scale scenarios such as friends playing a game on the train or co-workers sharing calendar information. Moreover, AdSocial is specifically designed to run on resource-constrained mobile devices, such as mobile phones. By using a simple and efficient data piggybacking mechanism, AdSocial applications share data by using any of the many existing routing protocols for ad hoc networks and without requiring any modification to the protocols themselves. The goal of this demonstration is to show the functionality that AdSocial can support with a very low overhead in an ad hoc network of 10–15 Nokia N810 handhelds. Conference participants will be able to establish voice-video calls, chat, or play games while moving around thus configuring a mobile and multi-hop ad hoc network.

1. INTRODUCTION

Whether for work or for entertainment, the mobile phone has positioned itself as a universal device that provides users with access to a wide variety of services. In particular, the availability of short-range wireless technology such as WiFi and Bluetooth makes it possible to build a new class of self-organizing applications running in ad hoc networks of mobile phones. Rather than focusing on traditional ad hoc network applications such as military battlefield, emergency rescue operations, or entity tracking, we are interested in small scale networks established, for instance, to play a game with friends on the train, to make a social connection with people in a bar, or to share photos and documents in office or household environments.

To support this kind of applications we built AdSocial, a complete software stack and middleware platform supporting social networks and related applications in ad hoc networks. AdSocial offers a rich set of functions like chat, video, VoIP, buddies search, and games, and it is specifically developed to target the resource-constraints of small mobile devices. Its current prototype has been tested in ad hoc networks of 20+ Nokia N810 Internet Tablets and it has been used in several field trials. These were organized in office environments and during a 3-day retreat where it was successfully used by 40+ members of our research group both in indoor and outdoor settings. In the demonstration, we will demonstrate AdSocial using 10–15 Nokia N810 Internet Tablets. In the following we give a more detailed description of AdSocial.

VLDB '09, August 24-28, 2009, Lyon, France

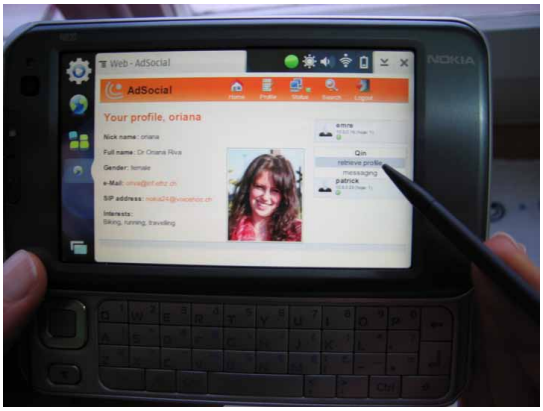
2. ADSOCIAL

We outline the main features of AdSocial, its software architecture, and its data model. We also give examples of several applications that have been integrated in it.

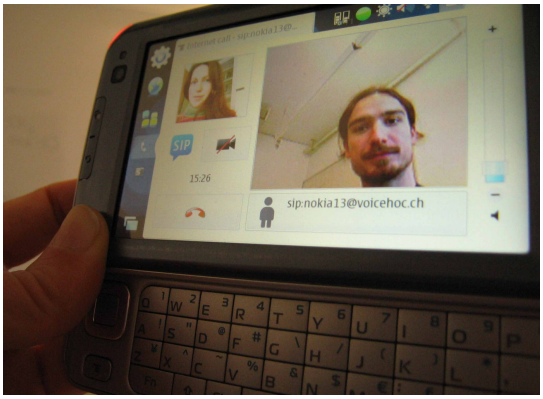
2.1 Features

AdSocial integrates diverse functionalities typical of social network sites such as Facebook, Orkut, or MySpace. Figure 1 shows a Nokia N810 Internet tablet running AdSocial. The snapshot in Figure 1(a) shows the profile of the user Oriana. The left hand side of the profile contains personal information, such as a photo, her interests, and her email and SIP addresses. On the right hand side, the nearby *buddies* are displayed. Unlike typical online social networks where a user's list of buddies consists of friends explicitly selected by the user, in ad hoc social networks, buddies are nearby users whose presence has been detected by AdSocial. As shown in the figure, users can retrieve the profile of any nearby buddy by right-clicking on the buddy's icon and start a chat session with them. Alternatively, they can also search for buddies matching some specific interest. Currently, interests are matched using a simple string matching algorithm, but in the future more advanced matching support can be added.

AdSocial is designed to support spontaneous interactions and provide the basis for running collaborative applications. It currently supports presence detection, games (e.g., Quake I [6] as shown in Figure 1(c)) as well as chat, voice and video calls (see Figure 1(b)). For example, if a user retrieves the profile of a nearby buddy, and the profile contains the SIP address of the buddy, he/she will be able to establish a voice/video session by simply clicking on the SIP address and enabling the device camera.



(a) Profile, user status and buddies



(b) Voice-video call

(c) Game application

Figure 1. AdSocial applications running on a N810 handheld

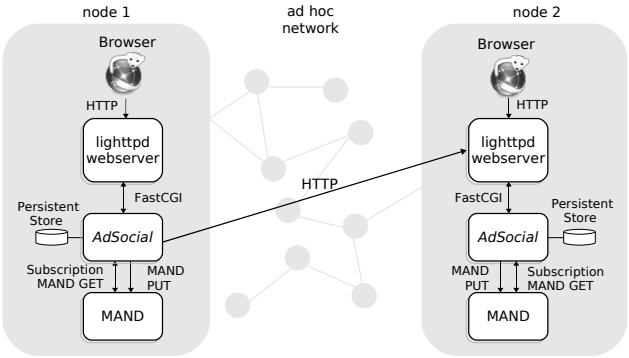


Figure 2. AdSocial system architecture.

2.2 System Architecture

AdSocial is implemented as a local web application running in a regular web browser (see Figure 2). The local *browser* connects to the *lighttpd web server* [4] that redirects the calls to AdSocial via a *FastCGI* interface [2].

The whole presentation logic is written in JavaScript and executed by the local web browser (e.g., Firefox). The interaction between the presentation logic and AdSocial is done using local *AJAX* polling. The browser periodically queries the local AdSocial instance to get updates about the availability of nearby buddies and refreshes the displayed information.

Information about nearby buddies such as their presence status, SIP address, or interests is retrieved directly from the web server of the corresponding buddy in the ad hoc network. To locate a buddy and to determine the address of a buddy's webserver, AdSocial uses *MAND* (Mobile Ad hoc Network Directory) [7], a distributed directory service for ad hoc networks.

2.3 MAND Data Model

MAND provides the abstraction of a distributed tuple space in ad hoc networks. Tuple spaces like Linda [3] are well-known shared memory models. Data is represented as elementary data structures, called tuples, and the memory, called tuple space, is a multiset of tuples. Sharing of data among processes occurs by writing tuples (i.e., adding tuples to the tuple space) or reading tuples (i.e., reading and eventually removing tuples from the tuple space).

Likewise, the MAND tuple space supports three basic operations:

- `put(tuple)`: to insert a `tuple` in the ad hoc network;
- `get(tuple_request)` to look up for a certain tuple available in the ad hoc network; the operation places a `tuple_request` in the network;
- `put(tuple_delete)` to remove a tuple previously made available in the ad hoc network; the operation places a `tuple_delete` in the network.

A tuple consists of a sequence of typed fields. It contains a `key`, a `value`, an `owner`, a `scope`, a `lifetime`, and a `version` field. A `key` specifies the type of the `tuple` and the `value`

represents its content. The `owner` is the identifier of the node generating the tuple. The `scope` field specifies for how many hops at most the tuple must be disseminated in the network. The `lifetime` field specifies for how long a tuple is stored locally at a node. The `version` field specifies a replacement schema among tuples with the same key. Tuples with older versions and expired lifetime are automatically erased from the tuple space. Therefore, a tuple is uniquely identified by the pair `<key,version>`.

In the case of AdSocial's presence information dissemination, a user publishing its online status can, for instance, submit a tuple like the following, where he advertises his name and his status:

```
tuple = {
  key = "adsocial"
  value = 15.10.5.2:80|adsocial|Patrick-available
  version = 1
  owner = "15.10.5.2"
  scope = 4
  lifetime = 100
}
```

`Tuple_request` and `tuple_delete` have the same structure of a normal tuple with the only difference that the `value` field is optional. The search criteria is specified using the `key`, `value`, and `version` parameters. `Owner`, `scope`, and `lifetime` refer to the actual `tuple_request`, but they are not used in selecting the tuple to be returned.

Both one-time requests (`lifetime = 0`) and subscriptions (`lifetime >0`) are supported. Once a user has placed a `tuple_request` into the network, matching tuples are returned. For example, to be notified about other buddies' status changes, a user can insert the following `tuple_request`:

```
tuple_request= {
  key = "adsocial"
  value = s:String
  version = *
  owner = "15.10.5.3"
  scope = 5
  lifetime = 50
}
```

The `value` field offers more flexibility compared to the other fields. It can contain both *actuals* and *formals*. Actuals are values (as the fields of the tuple's value shown in the first example) while formals are types such as integer, string, and float. In the `tuple_request` above, the `value` field contains a formal: it indicates that a `String` type is expected in the `value` fields of matching tuples.

Therefore, a `tuple` and a `tuple_request` (or a `tuple_delete`) match if they have *1)* matching `keys`, *2)* matching `formals` and `actuals` in the corresponding `value` fields, and *3)* matching `versions`.

A special parameter * permits specifying the "any" relationship. This parameter can only be used in the value and version fields of a `tuple_request` or a `tuple_delete`. For example, the previous `tuple_request` is meant to search for tuples whose `version` can be any value. However, to limit the number of tuples returned in the case of tuples with long lifetime, a user can also specify `version=LAST`; in this case, only one tuple per each matching key will be returned, i.e., the one with the highest version.

Similarly, the same concepts apply to a `tuple_delete`. In the common case, a `tuple_delete` will contain a certain `key` and have `value = *`, `version = *`, and `lifetime = 0`. This means that the command will delete all tuples with the given key, regardless of their value and version.

In order to allow more flexibility, `keys` and `values` can be organized in multiple subkeys and multiple subvalues. For instance, in the case of a game such as the Quake I game that AdSocial supports, a game server may use `key=adquake` to send tuples to all active players who are (by default) subscribed to game updates (i.e,. they have sent a `tuple_request` with `key=adquake`). When the Quake server wants to send an update that interests only a subset of the active players (e.g., because they are located in the same game region), then the server can inject a tuple with subkeys and subvalues like the following:

```
tuple= {
  key = "player1|player2|player3"
  value = room3a|numobstacles25|level9|57
  version = 23
  owner = "15.10.5.10"
  scope = 3
  lifetime = 20
}
```

Keys can "match exactly", as in first pair of `tuple` and `tuple_request` described above, or "partially", as in the example shown below: the following `tuple_request` matches both tuples above, i.e., the one with `key = "adsocial"` and the one with `key = "player1|player2|player3"`.

```
tuple_request = {
  key = "adsocial|adquake|player1"
  value = *
  version = LAST
  owner = "15.10.5.7"
  scope = 4
  lifetime = 100
}
```

Although MAND provides explicit support for `tuple_delete` operations, given the unreliability of the underlying wireless network and nodes mobility, this operation cannot always guarantee to delete the specified tuple on all nodes where the tuple was previously stored. Except for critical operations, the simpler and more reliable way to guarantee a global deletion of a certain tuple, is to use the `lifetime` parameter of the tuple itself. The `tuple_delete` operation should be used only in those cases where a tuple was disseminated with a lifetime that largely exceeds the currently required lifetime.

2.4 Implementation

Applications built on top of AdSocial interact with the MAND system through its three basic operations. All application messages are wrapped into tuple objects that are then distributed by MAND. Communication can be of two types.

One possibility is to disseminate tuples and tuple requests across the ad how network by piggybacking messages onto routing packets exchanged by the underlying routing protocol. This communication channel is appropriate for disseminating small messages, that are not time-critical, and that can tolerate losses. Depending on the underlying routing protocol, the piggybacking space in routing messages can vary from 255 (the size of an AODV [5]

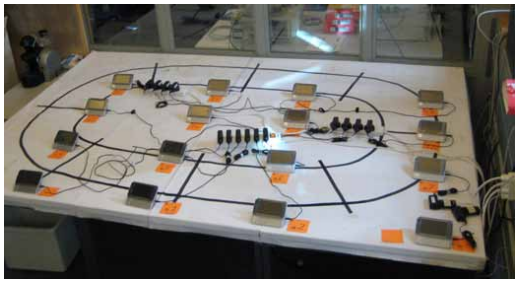


Figure 3. AdSocial testbed with 16 N810 handhelds.

extension) to 1200 (with OLSR) bytes. For instance, the presence service, previously described, uses this channel to periodically broadcast presence updates to nearby buddies. The SIP protocol uses this channel to disseminate SIP proxy addresses. The piggybacking-based approach provides message-efficiency and can work with many routing protocols without requiring any modification.

The second possibility is to use unicast messages. This is the channel to be used to transfer time-critical messages that do not need to be propagated to the entire network. When a SIP-based call is established the voice and video traffic goes through this channel. The multi-player game uses this channel for exchanging real-time game updates.

3. EXPERIENCES WITH ADSOCIAL

AdSocial has been deployed using Nokia N810 Internet Tablets, which although relatively powerful computing platforms, yet present several limitations in supporting the demands of a continuous social networking application like AdSocial. Nokia N810 tablets have 400 MHz processor, offer 128 MB of RAM, and run the Internet Tablet OS based on Linux 2.6.18. Figure 3 shows a snapshot of our testbed.

AdSocial is a solid platform that has been already used in several user trials employing 20+ N810 handhelds. In one indoor trial we placed the handhelds in our department building at ETH Zurich. Devices were placed in offices across three floors and users could set up voice-video calls or play games in groups. Another trial was organized during the retreat of our Systems group in a small town in the mountains of Switzerland. In this case, 30+ users used AdSocial to call each other from room to room, chat, and keep track of users' location in the hotel.

4. DEMONSTRATION HIGHLIGHTS

In this demonstration we use 10-15 Nokia N810 Internet Tablets. The ad hoc network is established using the OLSR [1] routing protocol. We use a transmission power of 10 mW in order to create a multi-hop topology even in a limited space. The services enabled by AdSocial particularly fit the conference scenario. Conference participants will be provided with N810 devices and will be able to interact with AdSocial while residing in different spots of the conference room(s) and possibly moving around. We will demonstrate how AdSocial can turn out to be a very useful and convenient tool for communicating in a crowded conference venue, and participants will gain practical evidence of the user experience that can be provided.

More specifically conference attendees will be able to:

- Create their own personal profile, specify interests to be advertised in the network, publish their profile and set their current status (available, busy, etc.); a basic profile can also be created on a laptop thus avoiding typing on the small keyboard of the N810.
- Initiate search for buddies with matching interests: the user inserst a few key words and buddies with matching interested are highlighted on the buddies list.
- Chat with multiple buddies at the same time, as in typical IM applications
- Set up voice-video calls to other buddies: the user can retrieve the profile of another buddy and by clicking on its SIP address, the call is established. Additionally, the user can click on the camera icon and enable video.
- Play Quake: users can play Quake with other buddies (maximum 16 players are supported in one game). A user who wants to initiate a game with others, can select one or more buddies from its list of buddies and send them game invitations. While playing Quake, by moving the cursor on a specific player the user can see the name and other basic profile information of the corresponding buddy-player.

In addition to interacting with AdSocial, by means of a simple applet to be started on the handheld, conference participants will be able to view how the routing topology varies over time and space, and can monitor several performance metrics such as number of exchanged packets, bytes sent and received, or number of lost packets. This will allow conference participants to understand in more details how AdSocial, MAND, and the underlying routing protocol interact.

Acknowledgments

The work presented in this paper was partly supported by the National Competence Center in Research on Mobile Information and Communication Systems (NCCR-MICS), a center supported by the Swiss National Science Foundation under the grant number 5005-67322, and partly by the ETH Fellowship Program. N810 handhelds were provided by Nokia's Sensor Planet initiative.

References

[1] T. Clausen, P. J. (editors), C. Adjih, A. Laouiti, P. Minet, P. Muhlethaler, A. Qayyum, and L.Viennot. Optimized Link State Routing Protocol (OLSR). RFC 3626, October 2003. Network Working Group.

[2] Open extension to CGI, 2009. http://www.fastcgi.com.

[3] D. Gelernter. Generative communication in Linda. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 7(1):80–112, 1985.

[4] Lighttpd web server, 2009. http://www.lighttpd.net.

[5] C. Perkins and E. Royer. Ad-Hoc On-Demand Distance Vector Routing. In *Proceedings of the 2nd IEEE Workshop on Mobile Computing Systems and Applications (WMCSA'99)*, pages 90–100, February 1999.

[6] Quake I, 2009. http://quakeone.com/.

[7] P. Stuedi. *From Theory to Practice: Fundamental Properties and Services of Mobile Ad Hoc Networks.* PhD thesis, Swiss Federal Institute of Technology Zurich, November 2008.