Experiments in Mobile Social Networking

Anna-Kaisa Pietilainen, Earl Oliver, Jason LeBrun,
George Varghese, Jon Crowcroft, Christophe Diot
Thomson, Paris, France.

Thomson Technical Report
Number: **CR-PRL-2008-02-0003**
Date: February 7th 2008

Abstract: In this report we consider a mobile ad hoc network setting where users of Bluetooth enabled devices meet and communicate opportunistically as when random people meet in a cafe, or researchers meet at a conference. Ad hoc opportunistic contacts are built on the basis of pre-defined relationships in online social networks. Our approach distinguishes itself from previous work in the area by three characteristics: the removal of a need for a central server to conduct exchanges, the focus on the transitive closure of relationships, and the use of existing social networks as a reference point for understanding whether an exchange is desirable. We design MobiClique, a social interaction communication software package that we implement on smartphones. We develop three applications: ad hoc social connections, epidemic newsgroups and asynchronous messaging. We describe our experience with 28 users at a networking conference. We discover that despite the usual experimental hazards, MobiClique was successful at building a local social network and delivering more than 300 user generated messages over 3 days through multi-hop epidemic communication.

1. INTRODUCTION

Applications in the virtual world such as social networks and instant messaging have done much to remove the tyranny of geography. Beyond friendship and exchanges between two parties (which we will refer to as *dyadic* communication), virtual groups have proliferated, creating communities centered around interests varying from gaming to editing Wikipedia pages. Despite the increased power and reach of virtual communities, we postulate that the power of physical communities based on physical contact and closeness will continue to be an essential part of human relationships. Further, physical communities have a different set of capabilities that are complemented by but not subsumed by virtual communities. We can certainly email a buddy in the virtual world who is in another country and a different time zone; but we can only share a meal, or go to a play with a friend in the physical world.

Most users do belong to both virtual and physical communities today but it is our belief that few networking experiences leverage both worlds. Rather than look on virtual communities and physical communities as *competing* entities, we prefer to think of them as *complementary* entities.

This approach leads to interesting research problems. First, technologies that were built to create virtual communities must be extended to meet the constraints of physical communities. Second, we need to understand how virtual and physical communities can work together to leverage each other so that users can move between these two worlds in a way that enhances both worlds.

The MIT Serendipity project provided an interesting answer to the first point above [4]. They observed that the market for cell phones was growing exponentially even in 2004; today cell phone sales are ten times the sales of PCs and laptops, reaching 1 billion in 2006 alone [1]. Further, while a certain fraction of the world could probably not afford a PC, cell phones are becoming ubiquitous in terms of coverage with people in developing countries. In addition, most cell phones have low powered Bluetooth radio interface that provides cheap (and essentially free) communication between two devices that are reasonably close to each other. Serendipity uses the cell phone as a bridge between the physical and the virtual communities: the cell phones locate individuals in the user's proximity and communicate with a central server that contains information about users and provides several methods of matchmaking. They envisage applications in people meeting at conferences, enterprise introductions, and dating services. Similar ideas can be found in a number of commercial products.

In this paper, we extend the ideas of Serendipity in several significant ways, and describe the experimental results of a system (called MobiClique) designed around our new ideas. To describe our extensions, we observe that systems such as Serendipity have the following limitations that we address.

- **Use of a central server:** All existing systems use a central server that does the matchmaking between compatible users. Connecting through a central server is a serious restriction to the viral nature of the system.
- **Dyadic Communication:** The net result of an interaction is that two users are enabled to begin a physical communication. While this is a good thing, we believe it does not go far enough to build what we call an "ad hoc community".
- **No platform for further development:** While the existing social networks are more mature and hence possibly more open to the idea of open APIs, the current generation of social networking software is closed and likely to remain so for the near future.

In this context, the contributions of this paper are as follows:

Decentralized operation: MobiClique does not rely on centralized server(s) for the discovery and interaction process. We use direct peer interactions to create a community. Such direct interactions are both cheaper, more private, and may allow spontaneous and unforeseen interactions.

Leveraging an existing social network: Because of the interface and issues of harassing one's friends to join a specific social network, our system bootstraps from an existing social network. We choose to use Facebook because (1) it is currently the one of the largest and most popular online social network, and (2) its APIs are publicly available. Note that while MobiClique is designed to take advantage of contact opportunities, we consider that users will connect periodically to the virtual network (i.e. Facebook) to download relevant aspects of the social network that allow disconnected operation, and to update their profile based on the new social connections they have made with MobiClique.

Transitive closure of relationships: A MobiClique opportunistic relationship is not dyadic. Instead, temporary relationships create a graph just as in social networks. When two users decide to have an *exchange*, MobiClique provides a copy of each user's social community before the exchange. The result is that the community grows with each exchange.

Providing incentive: One of the arguments against collaborative opportunistic forwarding is the one of incentive. Forwarding a message for someone unknown

[1]http://www.eetimes.eu/uk/197000427

might be an issue as it drains batteries and uses phone resource. In order to address this issue, all data transmission are performed in a *social network overlay*, i.e. a device only forwards data for friends, or for people who share the same interests. We believe that this greatly increases the likelihood that users will participate in such a system.

Platform for development: While we have built some applications on top of the community substrate, we believe that an open API would be the basis for other unforeseen applications to be written by other programmers. Furthermore, we use our platform to gather data transfer statistics that can provide new insight into the behavior and expectations for data communications in an opportunistic networking environment.

Another major contribution is that we implement MobiClique on Windows Mobile smartphones. We report experimental analysis with 28 users attending a networking conference (CoNEXT 2007 held in New York, December 10-13, 2007), who use their smartphone to make new friends, to exchange messages and to publish information using an ad hoc newsgroup application. While one can theorize on the potential of new social software applications, and on the various research issues that they raise, experiments and careful measures are needed to capture the efficacy of such interactions.

Our implementation and design choices are motivated by social interaction among users. MobiClique is not a final design, but a prototype with many imperfections that will be analyzed and discussed in this paper. Once stable, MobiClique will be made available to researchers, users, and application designers.

2. COMMUNICATIONS ARCHITECTURE

MobiClique is a platform to discover, share, and utilize the "invisible" social networks that form around every mobile device user. MobiClique mimics human behavior in a social environment — look around, try to identify people or points of interests, prioritize, and finally build a social interaction with one or more people. As with human interaction, some of these activities might be "exclusive" of others.

The initial "look around" is implemented as a scanning phase, where a device tries to identify all potential candidates for social interaction. This defines the device neighborhood. It is followed by the identification step which consists of collecting the *social profiles* for each device in the neighborhood, and selecting the subset of candidates for social interaction. By a social profile we refer to the set of information associated with a particular user such as the friendly name and other personal details, a list of friends, the interests of the user and the groups the user belongs to (compare to a user profile in any common online social network).

The selection criteria for social interaction depends mostly on friendship, common social interests and relationships in virtual communities. Prioritization is then achieved using multiple criteria, including contact duration and frequency, active applications, etc. Up to this stage, only "social" information and contact statistics have been exchanged. Content comes next, once a "link" (and more exactly a peer network) has been established between the user and its social ad-hoc neighbors.

Our social networking protocol follows this progression of contact and information exchange, but uses opportunistic networking technologies to streamline the process in order to meet new, previously unknown people and exchange information with them in a variety of scenarios.

2.1 A Social Networking Protocol

The first step in the social networking protocol is to discover devices in the neighborhood. We use the scanning features of Bluetooth to identify this neighborhood. For each device found in the neighborhood, Bluetooth returns a MAC address. In the following sections, the *scanning node* designates the node which has just completed an entire Bluetooth discovery. A *neighboring node* designates any one of the nodes that was returned as a result of a discovery. Each participant in the system has a globally unique identifier (GUID) that represents that person in the social networking space.

Once the neighborhood is known, the scanning device attempts to establish a direct connection with each neighboring node, in order to get its user's social profile. This is what we call the *social metadata update* phase. Once completed, it is followed by an *application data exchange* phase where the scanning device reconnects with the neighboring devices to whom it has data to send. Our social networking protocol executes this continuous loop of social metadata updates followed by application data exchanges. The periodicity can be tuned depending on various human and technology factors.

Note that every connection between two devices begins with an authentication exchange. In the current prototype, the authentication exchange is simply the exchange of the device's user GUID. If the GUID exchanged does not match the MAC address of the device, then communication is aborted. However, in future implementations it will be possible to match a user GUID on multiple MAC addresses.

The following two subsections provide details about the design of the social metadata update and application data exchange protocols.

2.1.1 Social Metadata Update

During the social metadata update phase, the scanning node receives metadata update from neighboring devices. We chose to implement a one-way data trans-

Algorithm 1 Social Networking Protocol

```
$scan\_results$ ←Bluetooth MAC from last discovery
for $neighbor$ in $scan\_results$ do
    $changesguid$ ← Authenticate($neighbor$)
    if $changes == True$ then
        $guids$ ← RequestInfo($neighbor$)
        for $guid$ in $guids$ do
            if notInDb($guid$) then
                RequestFriendlyName($guid$)
            end if
        end for
    end if
end for
for $neighbor$ in $scan\_results$ do
    $dataList$ ← GetDataFor($neighbor$)
    if len($dataList$) $> 0$ then
        SendDataOffer($neighbor$)
        for $item$ in $datalist$ do
            SendDataItem($neighbor$, $item$)
        end for
        while StillConnected($neighbor$) do
            $item_i d$ ← ReceiveAck($neighbor$)
            MarkSent($item_i d$)
        end while
    end if
end for
```

fer as it significantly simplifies the complexity of the implementation. However, there is no technical reason for not performing a symmetric update.

During the authentication step described above, the neighboring node returns to the scanning node a checksum representing its current social profile. If the neighboring node has made changes to its profile since the last contact with the scanning node, the checksum has changed and the neighboring node sends its new profile to the scanning node. If the checksum has not changed, no update is performed.

2.1.2 Application Data Exchange

As before, this phase begins with an authentication exchange, which is followed by a message indicating the number of data items to be transferred. Then data items are transferred one at a time; each data item starts with a header that contains message payload size, destination, and a unique message identifier followed by the actual data chunk.

A policy must be chosen for determining the order in which to connect to neighbors, as well as for selecting which items to send first. In the current prototype, we connect to neighbors in the following order: friends, shared interests, friends-of-friends. Data items are simply passed in a first-come first-sent manner. This policy is simple and will have to be reviewed in future work in the light of experimental observations.

As the neighboring node receives data items, it responds with an acknowledgment per data item. The scanning node uses these acknowledgments to mark data items in the database as successfully sent so it does not unnecessarily re-transmit.

2.2 Applications

We have developed three applications to run on top of the MobiClique architecture. The application framework has been designed to make it easy for third party developers to write new applications. We expect the list of applications to grow once our API stabilizes and is made public.

The first application simply displays to the user the set of neighboring devices with their social profile and provides an interface to manage the user's social network (i.e. add and remove friends, change interests). The user is informed of the presence of a friend or a potential new friend by a set of distinctive rings or vibrations. A potential new friend is currently defined as a friend of a friend or as someone sharing at least one interest in their social profile. This *social network management application* also provides the bridge between the ad hoc social network stored within MobiClique and an existing online social network such as Facebook.

We have deployed a *Facebook plug-in* in order to initialize a user's social profile with realistic social information. After downloading a user's online social network on the device, we utilize their Facebook ID (a value globally unique to Facebook users) as their MobiClique GUID. Similarly, Facebook users belong to groups which are represented by fixed and unique GUIDs. We use Facebook group GUIDs as a simple solution to the complex problem of representing the taxonomy of interest topics. During periods of Internet connectivity, MobiClique users can synchronize their ad hoc social network back to their Facebook account.

The second *asynchronous messaging* application was developed to allow friends to message each other. This application is modeled around a conventional instant messaging application. It displays a contact list containing the user's friends. Selecting a friend opens a window containing previously exchanged messages in chronological order. Users may reply to a discussion thread or create a new message destined to their friends. Note that these messages can be relayed by intermediate node toward destination given that these intermediates nodes are either friends of the sender or of the destination.

The third *epidemic newsgroup* application allows discussions among multiple participants, based on a specific topic of interest (e.g. newsgroup). It is best compared to a mobile ad hoc "Usenet" [2]. The user interface

[2]http://en.wikipedia.org/wiki/Usenet

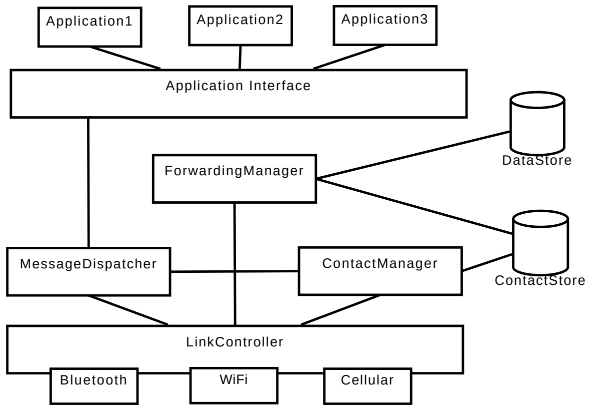

Figure 1: MobiClique Architecture Overview

is similar to the asynchronous messaging application, but the message threads are displayed within the context of a particular interest group, rather than for a particular friend. The common interest is also used as a message forwarding criteria.

3. SYSTEM ARCHITECTURE

In this section we present an overview of MobiClique's socially driven software architecture. The central components of the architecture are the two databases, one for contacts and social profiles, and one for application data. We first describe these two "stores", and then we discuss the modules which manage and act upon the data in these stores. The relationship between the main components of the system is outlined in figure 1.

The **Contact Store** provides persistent storage of device contact statistics, including MAC addresses for each device seen, and the user metadata. The user metadata (i.e. the social profile) consists of the user name and some optional details such as an email, a list of friends and a list of interests. The Contact Store maintains relationships between the system-specific GUIDs for users, friends, interests and applications. Internally, MAC addresses are used to uniquely identify neighboring devices. The Contact Store can manage a user with multiple devices, or with a device containing multiple -interfaces, by associating the user's GUID with each MAC address. The Contact Store is updated during the metadata update phase.

The **Data Store** is responsible for storing and providing efficient access to the data entering the system through the Application Interface and data exchanged between MobiClique devices. This store maintains *data items*, which are autonomous pieces of data. Each data item may have one or more destinations associated with it. The Data Store also stores information about neighbors who have already received a given item, and a TTL for each item, to prevent redundant transfers.

The **Link Controller** is the lowest layer in the MobiClique architecture. The Link Controller provides a common interface for sending and receiving data across a variety of network interfaces. The Link Controller is responsible for periodically scanning for neighboring devices or access points. This component receives data on any of the available interfaces. The Link Controller provides a common API for all network interfaces that it supports and it hides network technology specifics from the rest of the application. The Link Controller is further discussed in [13].

The **Network Message Dispatcher** component receives interface agnostic messages from the Link Controller and dispatches them to the proper MobiClique component, after determining whether a message is a control message or a data message. Data messages are passed to the Application Interface, control messages are passed to the Contact Manager.

The **Contact Manager** coordinates the communication and social metadata updates with the neighboring devices. It uses the the Contact Store to persistently store all friends, interests, and other meta data associated with encountered MobiClique users. The Contact Manager maintains changes to the set of MobiClique users in the current neighborhood which is used by other components like the Forwarding Manager and applications.

The **Forwarding Manager** matches data to be forwarded with potential destinations or next-hops, using information found in the Contact Store. For each message stored in transit, it makes a decision on what the next hop should be, based on the current state of the neighborhood. It also controls the aging of messages by using a TTL. The Forwarding Manager has been designed to support an arbitrary number of forwarding algorithms; however, the current version of MobiClique only supports epidemic forwarding within nodes who are friends, friends of friends or share similar interests.

The **Application Interface** is designed so that developers can easily create new applications that use MobiClique functionality. Each MobiClique application has a unique GUID that it registers with the Application Interface at startup. The application then simply communicates with the MobiClique system using a loop-back socket. The Application Interface decides how to handle local messages by using the application GUID found in these messages.

We have implemented MobiClique in C++ to run on a variety of mobile devices. Our current prototype runs on Windows Mobile, but by abstracting the operating systems APIs from the core application, our code can easily port to Symbian and other C++ based platforms. Our implementation is compact (less than 500 KB in size), has a low memory footprint at runtime, modular design to support experimentation and growth, and supports devices with a heterogeneous set of wireless capabilities.

4. EXPERIMENTS

This section begins with an outline of the research objectives addressed by our experiments followed by a description of the experimental conditions. We also describe some problems we encountered during the experimental campaigns.

4.1 Objectives

The core objective of the MobiClique experiments is to gain understanding of the interaction between our social networking protocol, the users' social behavior, and our software implementation. From the system design point of view, we want to find the bottlenecks of the system, and verify if our design choices and the selected parameters are reasonable. We also try to understand how users interact with the proposed applications, what kind of an application interface is needed, and whether the core system offers adequate support for ad hoc social networking.

Our architecture relies on Bluetooth for device discovery and data communications. The basic device discovery performance metric we look at is the number of devices detected on each scan. The set of detected devices will contain any Bluetooth devices in range. We use the following terminology for the rest of the discussion. We refer to MobiClique devices as *internal* and the other non-experimental devices as *external*.

We define a *contact* as the time interval from the first sighting to the last sighting that is followed by at least two scans where the device is no longer present. In other words, if a device is missed in one Bluetooth scan but visible in the next scan, we consider that to be part of a longer contact. We do this to overcome the inherent limitations in Bluetooth device detection that can occasionally miss devices in a scan though the device is actually in proximity. This definition of contact was also used in previous studies [6]. In addition to contact times between a pair of devices, we measure the *inter-contact time* which is the time interval between two consecutive contacts.

Finally, we want to analyze the communications side of the system, namely the opportunistic message forwarding. It is important to understand how the epidemic message forwarding behaves in terms of message delivery success ratios, delays and the number of hops the messages take on their way to the destination(s).

4.2 Experimental Setup

We perform two separate experiments in conference environments. The initial experiment is designed to gain more insight into the ad hoc Bluetooth communications and design choices. It was performed with a subset of the MobiClique architecture, namely the Link Controller and a simple test driver on top of it.

We scan for Bluetooth contacts every two minutes for a duration of 10.24 seconds as recommended in the Bluetooth standard [1]. Laboratory tests have showed that that this scan duration is sufficient to detect all neighboring devices within a 10 meter radius [13]. The scanning interval of two minutes is chosen based on the previous human mobility experiments [6]. Also it is long enough to allow devices to transfer a reasonable amount of data (approx. 4 MB) and short enough that it does not waste energy by doing redundant scans [13].

After the discovery, we initiate data transfers between detected devices in two modes: sending 50 KB of random data to each contact, or sending only 1 MB of random data to a single contact. We measure the success rates and data throughput in each mode both on the client and server sides. The application requires no user interaction and the participants are simply asked to keep the devices charged and with them all the time.

In the second experiment we deploy the complete MobiClique architecture with all three applications. The implementation follows the proposed architecture except for the integration with Facebook. We disable this feature for three reasons. First, we found that downloading from the Facebook API is very slow due to many independent queries on a user's social network. Second, requiring that participants of our study load their profiles from Facebook also complicates the experiment process in return for little research value. Third, a simple crawl of Facebook revealed that there was little intersection between the Facebook profiles of conference participants.

Instead, we initialized the identity of each participant from a pre-defined list of participants the first time that the MobiClique application runs on a device. The experiment participant is then asked to select her friends from the same list. Limiting the list of friends to the set of conference participants is obviously not representative of their true social network; however, we believe that the social properties of our experiment are still valid.

Finally, without Facebook integration, we need a means for users to explicitly express their interests. We include an additional screen within MobiClique that allows a user to select from many preconfigured interest groups. These include research topics and various social topics. As in Facebook, users are free to add or remove friends and change their interest topics at any time.

Once the user has manually boot strapped her social network, MobiClique begins to execute the social communications algorithm. We also include an additional background messaging application for the purpose of experimentation. This *dummy application* runs on each device and generates random data in 50 KB chunks every 10 minutes. The destination for each data item is selected at random from each user's set of interests. Therefore, this application behaves like the newsgroup application, but the data is simply not visible to the user. This application guaranteed that even if users did not socialize, we would collect experimental data. Messages generated by the dummy application were granted a TTL of one hour to prevent messages from building up and flooding the ad hoc network. Having a one hour TTL also differentiated the dummy application from the asynchronous messaging and newsgroup applications, which had a TTL of one day.

	MASS 2007	CoNEXT 2007
Time	Oct 8-11	Dec 10-13
Devices	29	28
Trace (offline)	2.8d (27.1h)	2.2d (32.1h)
Unique Bluetooth devices	990 29 (internal)	2024 28 (internal)
Bluetooth contacts	21804 11702 (internal)	15109 10143 (internal)

Table 1: The experiments.

The first experiment was performed during MASS 2007 (Pisa, Italy) and the second at CoNEXT 2007 (New York, USA). We distributed around 30 Windows Mobile smartphones to a preselected set of participants on the first day of the conference and collected the devices back in the end of the conference. Table 1 summarizes the basic experiment setup and the collected data set characteristics. Both experiments lasted for three days. We define the offline time as the time during which (within the full trace time) the device was not running the application. The offline times result mainly from application crashes and depleted batteries. To compare the scope of the collected data, we calculate the number of unique Bluetooth devices seen and the number of Bluetooth contacts. At MASS we detect far less Bluetooth devices than at CoNEXT; however, the total number of contacts is higher at MASS. This is mainly explained by the different lengths of the collected traces and some issues in Bluetooth device discovery discussed further in 5.

4.3 Experimental Hazards

Field trials of experimental applications are often victim to unforeseen conditions; our experiments were no exception.

Clock synchronization: Clock synchronization, a property we take for granted in connected environments, can have a significant impact on the post-experiment analysis of experimentation in a mobile environment.

We began each experiment by synchronizing the device clocks to a single laptop when installing the MobiClique application. Despite manually configuring each phone, when analyzing the data we found that many devices were not time synchronized. Following the MASS experiment we found that mobile devices self synchronize with one of two sources: the cellular network and users' own laptops. For the convenience of not carrying two cell phones, several participants installed their own SIM cards in the devices; it caused several devices to synchronize their clocks with the connected cellular network. Other users in the experiment paired their devices with their Windows laptops that had ActiveSync (Microsoft's smartphone synchronization software) installed, which caused the device's time to synchronize to the time on the laptop.

As expected, mobile devices at the CoNEXT experiment were returned with non-synchronized clocks. We were able to reverse the clock skew through a series of scripts that associated sending and receiving times between pairs of devices. We expect clock synchronization to be a common problem in mobile experiments.

Battery depletion: Frequent Bluetooth scanning, data transfers, and frequent SD card I/O caused a significant drain on the battery. In the MASS experiment, despite verbally informing participants to charge their devices nightly, most devices were not charged frequently enough throughout the experiment creating long offline periods. For the CoNEXT trial we included a daemon application that monitored the battery level and reported warnings to the user. The warnings produced a vibration and audio alert, and displayed a message in the foreground of the display. The warnings became increasingly disturbing as the battery level fell below 10%.

User interface: One of the goals of the second experiment was to include real applications to experiment with real traffic and to encourage active user participation. However, due to the prototype nature of the application, some aspects of the user interface were still unfinished. This affected the user activity and some of the features, like the messaging application and the friend notifications were not fully understood by the users.

5. SYSTEM OBSERVATIONS

We begin the analysis of our experiment in mobile social networking with a discussion of system attributes that affect our study. We find that the introduction of real applications has a significant impact on the performance of MobiClique. We discuss how using Bluetooth have an adverse affect on mobile social applications. We also look into the impact of the database design on the performance of the system.

5.1 Bluetooth Limitations

The use of Bluetooth has distinct advantages. Bluetooth is short range, which conveniently limits the social context to devices within approximately 10 meters. Bluetooth data rate is slow compared to 802.11 (approximately 50 KB/s), but it is sufficient for many practical applications. Finally, Bluetooth consumes little power and can run continuously on most mobile devices [2]. Unfortunately, Bluetooth, as a cable replacement technology, was not designed for applications of the kind we envisage. Bluetooth devices discover each other by sending and receiving inquiry beacons. Once a device discovers neighbors, it switches to the "page" state to setup a new connection and share setup information. Pairs of devices subsequently move into the "connected" state to exchange data. While in the connected state, Bluetooth inquiries by neighboring devices *are unanswered* causing physically adjacent devices to be mutually non-discoverable.

This limitation of Bluetooth affects our application by preventing many opportunistic connections between socially connected users. We will see how these limitations impacts our experience in Section 6.2. One solution that we plan to implement is known as Bluetooth Pooling [9]. It is a technique for propagating presence among a set of neighboring Bluetooth devices. Each device stores the set of MAC addresses found during a inquiry scan and relays that set to each neighboring device that it connects to.

5.2 Database Design

Mobile devices are commonly characterized by their limited battery life, storage, RAM, and CPU. The integration of a database has several distinct advantages in the mobile environment. It provides guarantees on consistency of persistent data in the midst of power failure. Storage costs are minimized using efficient data structures, and when executing a query, most modern databases can adapt to low memory conditions to avoid having to later abort a transaction. However, when deploying a database on a mobile device, the database schema and subsequent queries can have a significant impact on the performance of the application during opportunistic connections.

In the data exchange phase of our social networking protocol, the Data Store is queried three times to find data to forward to a MobiClique device. On average, we found that finding an item in the database took around 1.4 seconds and inserting an item in the database (50 KB) took approximately 0.6 seconds. Although short in duration, these times were alarming considering that none of the databases grew to more than 41 items (approx. 2 MB worth of data). We expected lookup and insertion times for this small data size to be much lower. Using a simple microbenchmark that reproduced the behavior of the ad hoc newsgroup application, we observed that finding an item rose to a staggering 90 seconds when querying a database with 575 data items! This result was consistent across both the devices internal Flash memory and a removable micro SD card.

The poor performance is partly a consequence of the way MobiClique searches for data items. In each selection query, we retrieve a set of data items for a given set of destinations. The selection query is further constrained to items that have not been previously sent to the neighboring device. This prevents neighbors with long or frequent contacts from exchanging redundant data. This highly functional query consists of an expensive four-way join over three database tables. By removing the support for sending a single data item to multiple destination addresses, the complexity of the selection query would dramatically shorten, and consequently the time spent on database operations would be shorter.

During CoNEXT we monitored the available RAM on each device. Throughout the experiments our application used approx. 0.4 MB of the available memory. Globally, the average memory usage across all devices was around 28.1 MB (64.5%). We found that many participants used the devices to check websites, fetch emails, and take photos with the built in camera. The use of applications external to MobiClique was visible as memory allocation spikes. A key result of this observation is that despite heavy wireless I/O, frequent use, and an application exchanging large amounts of dummy data, the device had large amounts of unused memory. We believe that storing the table of sent data items (information that is not required to be persistent) in memory or caching subsets of the other tables would provide a significant performance gain compared to the current solution where everything is stored in the database.

Fortunately, short message TTLs coupled with limited workload prevented the design of our database from affecting the experiment. However, more sophisticated applications sharing larger amounts of data would quickly degrade in performance using the current design. We are confident that a simple redesign to the query structure would improve matters greatly.

6. COMMUNICATIONS PERFORMANCE

In this section we provide a detailed analysis of MobiClique. We begin the discussion by application usage. We then consider the three phases of the social networking protocol: the detection of contacts and the initial connection setup for the metadata update (Section 6.2), and the message forwarding (Section 6.3).

6.1 Application Usage

In this section we overview the MobiClique application usage during the CoNEXT experiment. We make no claim that our application usage is representative of how mobile users would use a larger-scale social mobile network. However, as a first experiment collecting such a data, we think this sets the context for understanding the detailed performance analysis presented in the following sections.

We conducted a small survey on the usage of online social networking applications and opinions on ad hoc social networking among the participants. Out of the 28 participants, 24 confirmed having one or more accounts in online social networks. Facebook was the most popular service with 16 users, followed closely by LinkedIn with 13 users. Orkut was also mentioned a few times. Only 4 participants were not using any online social networking application. We think that the fact that most of the experimentalists were familiar with some online social networking service, helped them to easily understand the idea and goals of our applications.

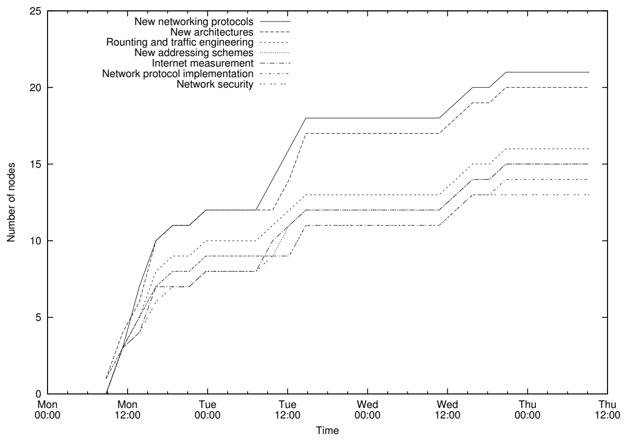

Figure 2: Most popular interest groups over time

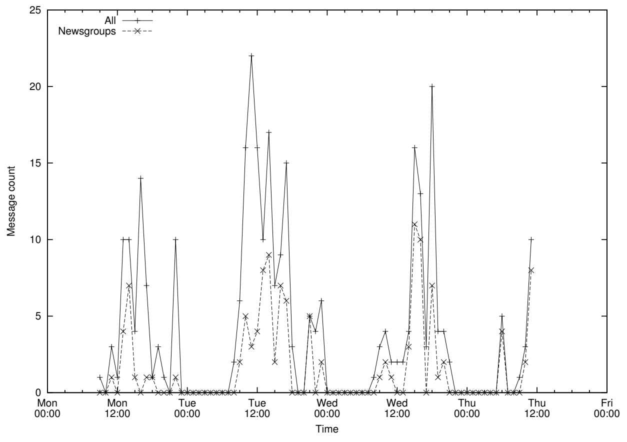

Figure 3: User generated messages per hour

```
1.  MobiClique discussion  (23)
2.  Network protocol evaluation (21)
3.  MobiClique feedback (17)
4.  Networked games (13)
5.  Testing (9)
6.  Thoughts on social-mobile networking (6)
7.  Beer drinkers anonymous (5)
8.  Wireless and mobile networks (4)
9.  Network protocol implementation (3)
10. Ad hoc and sensor networks (3)
```

Figure 4: Most active interests groups

When looking at the usage of the social networking application, we observe that 50% of the participants did not add friends at the initialization. The average number of initial friends was 4.23, some people having up to 17 friends initially. During the course of the experiment, the friend addition rate is 3.61 new friends per participant. The rate is affected by the small set of participants and by some user interface issues that lowered the user activity. Similar to the friends list, users could select and modify a set of interests. The predefined list included interests varying from conference topics to social activities and feedback. Figure 2 shows the number of users having selected an interests over the time. The figure includes only the most popular interest groups.

The asynchronous messaging application and the newsgroup application record 302 user created messages during the experiment. Of these messages 181 were destined directly to a friend and 121 to an interest group. The message generation over time is shown in Figure 3. Interestingly the most active discussion groups are not exactly the same as the most popular groups in terms of the number of users. Figure 4 lists the most active discussion topics (number of sent messages in parenthesis). The MobiClique "feedback" and "testing" groups receive a lot of messages followed closely by the more popular technical topics.

We found that 64% of the asynchronous messages reach their destination while only 48% of the newsgroup message do. The better success rate of the asynchronous messaging could be partly due to the fact that many people tested the system with their friends by sending messages to each other while both users were physically in contact and could verify that the message was indeed delivered. The newsgroup messages success rate is only a rough (and probably pessimistic) estimation since the users could change interests over the course of the experiment. We count the success rate based on the people interested on the topic at the message creation time.

6.2 Contact Detection

We start the analysis by looking at the number of devices discovered during the Bluetooth scans. The distribution of detected devices in the MASS experiment is shown in Figure 5 and at CoNEXT in Figure 6. The distribution is calculated over the working hours (from 9am to 6pm). By looking at the median line, we can

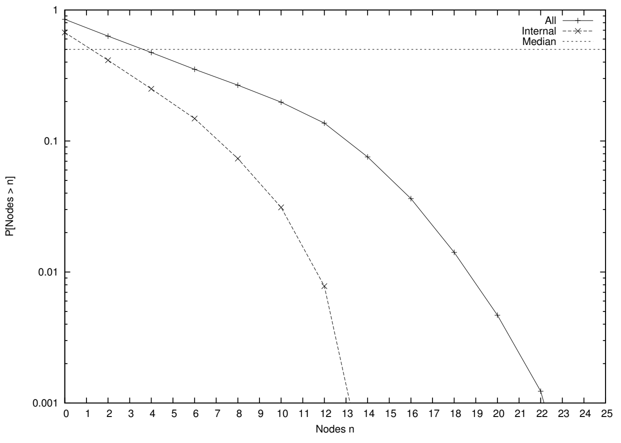

Figure 5: Discovered devices during a scan (MASS)

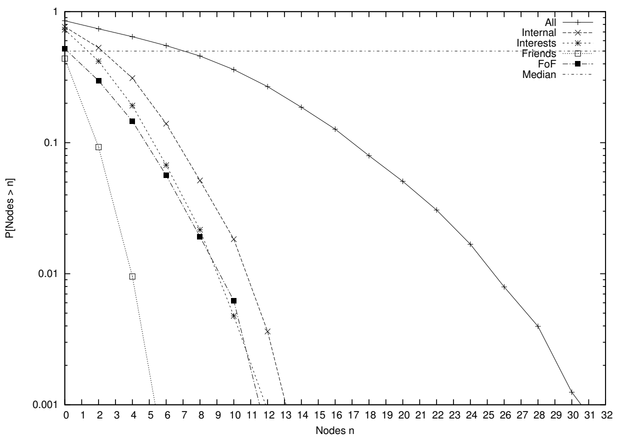

Figure 6: Discovered devices during a scan (CoNEXT)

see that at MASS in 50% of the scans we see at most 4 Bluetooth devices, of which only 1 is internal. This together with the fact that overall we detect less unique Bluetooth MAC addresses at MASS than at CoNEXT (see Table 1) implies that there were few external devices around and that the internal nodes were sparsely located. On the other hand, at CoNEXT we observe higher device density: for example looking at the median line again, 50% of the scans detect around 7 devices, about 2 of them being internal MobiClique users. The low number of internal device sightings is still surprising considering the environment where most of the devices were located in a single room during the conference hours. We explain this lower rate by the limitations of Bluetooth device discovery that were discussed in 5.1.

For the CoNEXT experiment, we calculate also the number of internal nodes sharing at least one interest with the scanning node, and the number of friends and friends of friends (FoF) among the scanned nodes. While these numbers present information that we extract from the collected data *after* the experiment, they still give an idea of the potential of the ad hoc social network formed in the experiment. As expected, due to the small set of nodes and the limited list of interest groups, we detect almost everytime a node that shares an interest with the scanning node and allows thus a data transfer to take place if the node has matching messages available. A friend was scanned in the neighborhood in 30% of the time, and a friend of a friend in 40% of the scans. These are an encouraging numbers but emphasis the need for improving the detection rate as we still miss many opportunities.

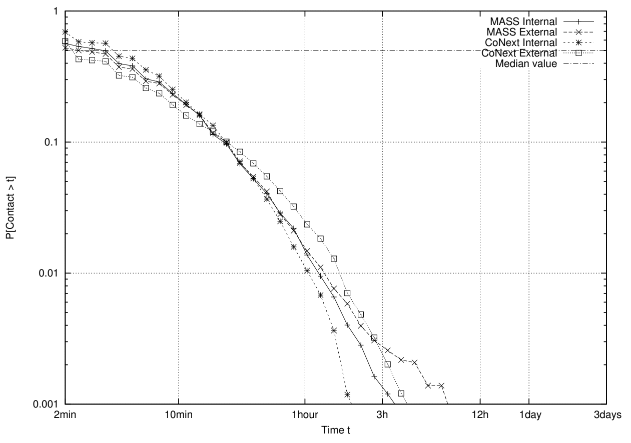

Figure 7: Bluetooth contact times

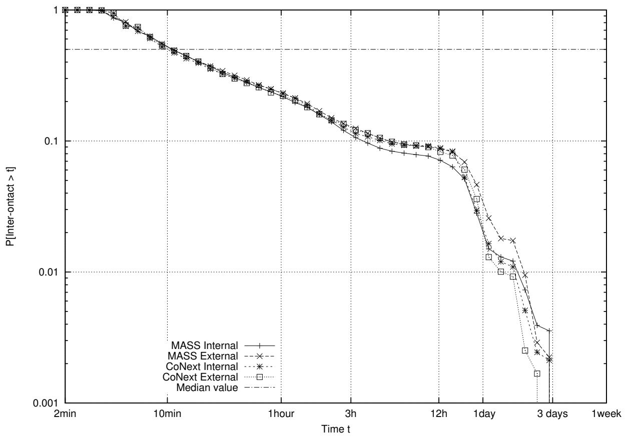

Figure 8: Bluetooth inter-contact times

Two important parameters for opportunistic networking are the contact time and the inter-contact time between a pair of nodes [6]. Figure 7 and Figure 8 illustrate the differences in contact and inter-contact time distributions in the MASS and CoNEXT experiments. In both experiments we see that approximately 50% of contacts last around 2 to 4 minutes. This correspond to a device being seen only in one or two scans. The contact times follow the same general trend as presented in [6]. However, in both the MASS and CoNEXT results, we observe that the contact times are significantly lower in particular towards the tail of the distribution. While the previous experiment report contact durations up to 1 day or more, in our data the longest contacts are well below that (2 to 10 hours). We attribute this decrease in contact time to application activity (as discussed in 5.1). From the inter-contact time distribution we can see that the median time is around 10 minutes and the tail goes up to the experiment duration of three days. The inter-contact time patterns are similar in both experiments and follow the power law shape observed and analyzed in previous work [6].

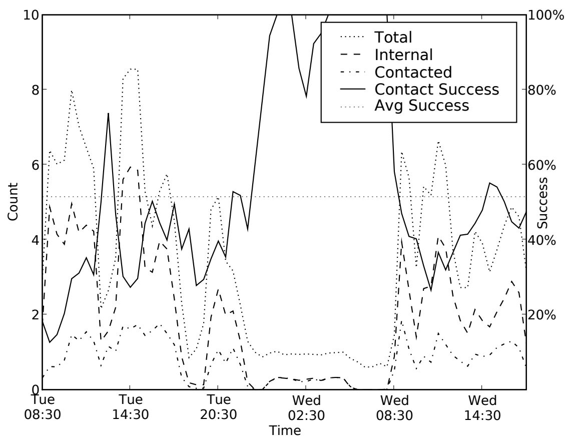

Figure 9: Bluetooth connections over time

Figure 9 shows the evolution of the average number of nodes seen during a scan, the number of MobiClique participants seen during a scan, and the number of nodes for which a successful socket connection was made before the next scan, averaged over 30 minute intervals. On the right side of the same figure, we show the connection success rates. On average, we manage to successfully connect to around 50% of the internal nodes we scan. At night time the success rate goes up to 100%; this could result from two experimentalist sharing a hotel room and detecting each other in an almost interference free environment.

By looking at this graph, we can also observe a visible correlation between socket connection success and the number of detected devices: the more devices we detect, the less successful we are in establishing a connection. This observation leads us to propose that systems such

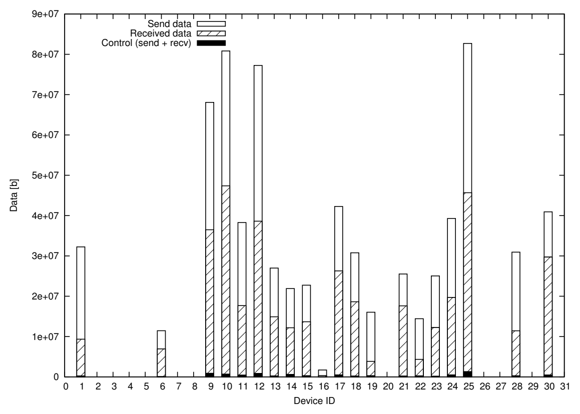

Figure 10: Send and received data per participant

as MobiClique should be more aggressive in attempting to make connections when there are more devices present. Our belief is that most of the failures to connect to a device are random and would not persist if we tried to connect again. Additionally, we think that we could improve the socket establishment success rates by exchanging further information about the adjacent nodes beyond the one hop neighbors and the neighboring nodes' activity in order to make smarter decisions about when and to whom to connect.

A successful socket connection is followed by the social metadata update phase. It takes around 20 seconds per scan round. The associated control data overhead is negligible as can be seen from Figure 10. The picture shows the amount of application data send and received together with the total control traffic. The tiny black area at the bottom of the bars represents the number of control bytes (authentication and social metadata traffic) sent and received. At glance this figure also shows qualitatively that the system is balanced. Each node sends and receives a fairly equal amount of data, some nodes being more popular (or just more online) than the others.

We believe that both the discovery and the connection establishment success rates could be improved by exchanging more detailed context information (such as the neighborhood and node activity) between the nodes. In addition, to increase the number of successful connections, nodes should try to connect each neighbors multiple times after it has been scanned.

6.3 Message Forwarding

The fundamental evaluation metrics for most networking systems are the message delivery success rate and the delay associated with message delivery. In this section we analyze these metrics. We separate the user generated messages for interest groups and friends, and the dummy application messages. The results for the various message types are remarkably different mainly due to the different TTL values (one hour for dummy messages and 24 hours for the messaging application). Also, the number of destinations (one for friend messages and several for interest based messages) explains some of the differences. Over the course of the experiment we record 302 user generated messages and around 6500 dummy application generated messages.

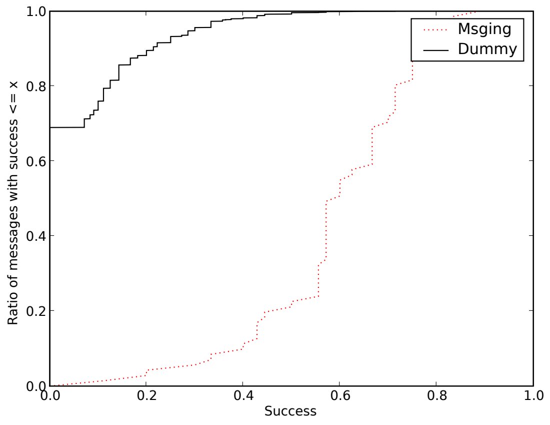

Figure 11: Delivery success rate for interest messages

Figure 11 shows the CDF of the message delivery success rates for interest based messages. In this figure, the x-axis represents the proportion of destinations reached by a message. The y-axis represents the proportion of messages generated that reached a proportion of destinations less than or equal to the value indicated on the curve. The success rate of the dummy application messages is still quite poor; around 70% of the messages fail to reach any interested node, and majority of them reach only 50% of the destinations. This can be explained by the short TTL and the way we select the message destinations. The dummy application would send data to a random interest group within the user's interests. The destination could thus be one of the not so popular groups, and the set of potential receivers would be very small. The newsgroup messages have a higher success rate. Half of the messages reach at least 50% of the intended destinations and the maximum success ratio goes over 80%. The asynchronous messages (not in the figure) have a success rate of 64%.

Figure 12 plots the message delay to reach a certain number of nodes for the dummy application, the friend messages and the interest groups. We calculate the delay as the difference between the time a message is generated, and the time it is received by an interested recipient. So, messages destined for an individual have one

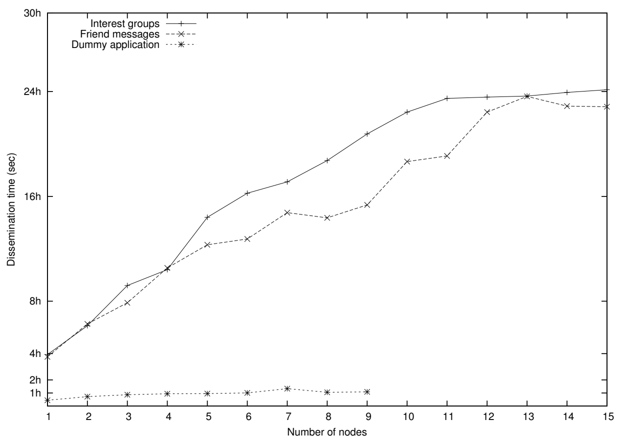

Figure 12: Message delays to reach a number of nodes

delay associated with them, but messages destined for an interest group have multiple associated delays (one per group member). The dummy application delays stay around one hour on average (corresponds the actual TTL) while the friend and interest based messages' delays seem to grow steadily up until around one day (the TTL) as the messages spread to a larger number of nodes. We can see clearly that the one hour TTL is short and the dummy application messages never reach more than 9 nodes at maximum while the messages with larger TTL reach up to 16 nodes. The big difference in reaching even a smaller number of nodes between the two types of messages is most likely due to the fact that the number of dummy messages generated is an order of magnitude larger than the user generated messages.

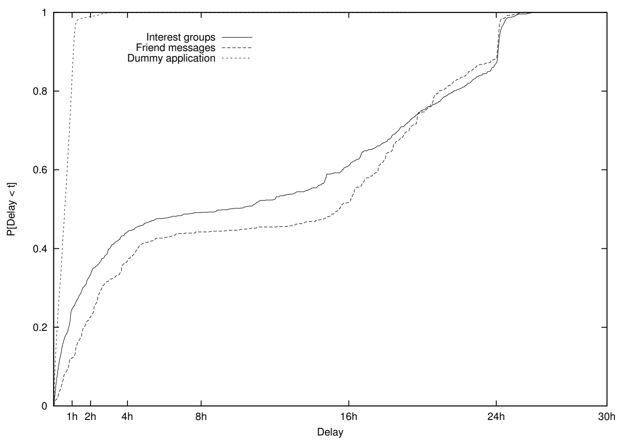

Figure 13: Message delay distribution

Figure 13 shows the delay distributions for all the three applications generating message traffic. The dummy application shows a nearly uniform distribution of de-

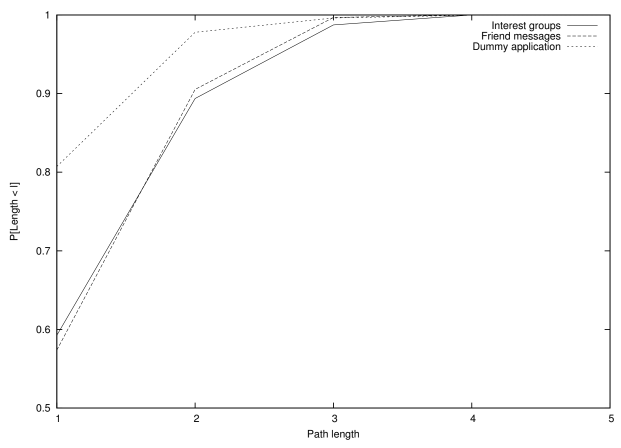

Figure 14: Path length distribution

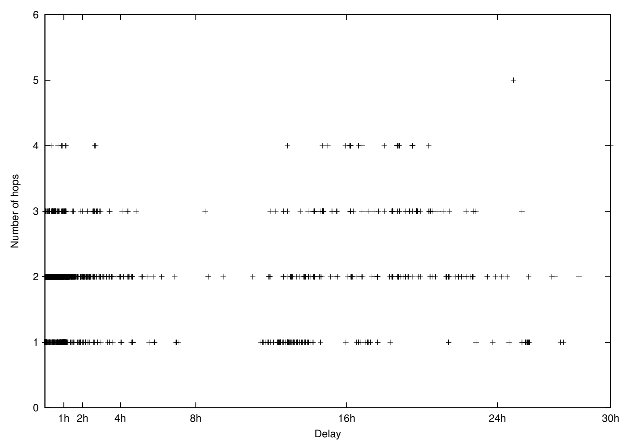

Figure 15: Path length vs. delay

lays up to one hour. For the friend and interest based messages we observe generally larger delays as was shown already above. There are some messages for each application that seem to pass the TTL. This results from the TTL being an absolute value and that the fact that devices were not time synchronized during the whole experiment; devices whose clock was behind the message generating node would keep forwarding the message longer that the intended TTL. In addition, the expired messages were purged from the Data Store only upon a new incoming message. Thus it was possible that a device could have forwarded stale messages even after the TTL was reached.

The figure shows an interesting division in delays for user generated messages – around 40% of the messages have a delay equal or less than 5 hours, and another 40% go over 16 hours. This can be explained by the diurnal behavior of the participants. Messages generated during the morning hours propagate until the end of the day (the first part of the curve) and expire before the next day. Those messages that are generated very late in the afternoon show up in the beginning of the curve, and again in the end as they persist in the devices over the night and are forwarded further the next day.

Next, we take a look to the forwarding paths. In Figure 14 we show the distribution of the path lengths for all the messages that were sent at least over one hop. The dummy application messages reach their destinations mostly over one hop (80%), the longest path being 4 hops. The user generated messages follow slightly longer paths in general. The average path length is between one and two hops, and the longest path we record in this experiment is five hops. This observation seems to align with recent work on the diameter of opportunistic networks [3]. We also think that it is an interesting result that even in a setting such as a conference environment where a limited number of devices communicate within a small area, the fastest way to reach some of the destinations is not always simple direct delivery; thus more advanced forwarding algorithms are needed.

Finally, we look into the message paths and the corresponding delays in Figure 15. Most of the observations that were made about the message delay distributions above are also visible in this graph: the message concentration at the short delays ($\leq$1 hour) consists of the dummy application messages and the rest is user generated traffic. Similarly, the gap in messages between 6 and 10 hours and delays larger than the TTL were discussed above. In general, there seems to be no clear correlation between the length of the path and the time it takes to forward the message over the path. Consequently, we propose that the TTL could be based on the number of hops the message has taken instead of the time. This matches the observations made in [3] where the authors argue that discarding messages after a relatively small number of hops does not decrease the performance considerably.

This section showed a clear relation between the message TTL and the message forwarding performance. The one hour TTL of the dummy application is clearly too short. The asynchronous messaging and newsgroups perform better as they have a longer TTL. However, these observations are not enough to determine an optimal TTL value which depends on multiple parameters not evaluated in this experiment. However, the next MobiClique communication protocol will experiment with a hop-count TTL.

7. RELATED WORK

Pocket Switched Networks (PSN) [14] is a communication paradigm that exploits contacts between mobile devices to transfer data in a store-carry-forward fashion. PSN falls under the more general concept of delay or disruption tolerant networking [5]. To our knowledge, we are the first ones to implement and evaluate a complete opportunistic communications system prototype with a set of applications. The Haggle Project and the DTN Research Group have produced PC-based reference implementations. There have been efforts to port the DTN reference implementation to a mobile platform (Symbian) [8]; however, this is only a preliminary solution.

Spurred by the exponential growth of online social networks, social networking is rapidly emerging as a hot research topic. MIT's Serendipity project [4] is one of the first projects exploring the mobile aspects of this topic. There have been many socially motivated technologies since Serendipity that are based on Bluetooth proximity detection. Notably, Nokia Sensor [12] is designed to detect neighboring devices and exchange messages and client defined profiles with them. The Wireless Rope project [10] used Bluetooth device detection to analyze the social context of the mobile devices and its affects on group dynamics. Cityware[3] from the University of Bath has explored problems related to "mobile computing landscape" [11]. They have developed a mobile application to share context (address books) and build common ground between nearby users [7].

Recently, most of the work in the mobile social domain has been commercial. Dodgeball[4] and MeetMoi[5] allows users to signal a central server with their current location. The system then notifies nearby friends and/or crushes in the area. Jaiku Mobile[6] allows mobile users to share location, availability, and current status. Twitter[7] allows users to inform their social communities about their current activity. Finally, LoveGety[8], a Japanese solution to the mate finding problem, allows users to find mates (contacts that match a desired profile) through opportunistic Bluetooth contacts.

8. CONCLUSION

MobiClique is ad hoc social software designed to build on virtual communities ala Facebook in order to facilitate the construction of opportunistic communities exploiting physical contacts made through cell phones. It brings together for the first time the strengths and complementary features of physical and virtual communities, starting form the observation that it is easier to meet potential friends in the street that in front of a computer.

We have designed and prototyped MobiClique on Windows Mobile smart phones, with all the imperfections that one might expect from a first prototype. Around 30 devices were distributed for experimentation, together with three applications during a networking conference. The first nice result is that it worked: messages were exchanged through multiple hops, and users made new friends among conference participants. The data set we have collected has similar properties when compared to previously analyzed datasets in term of contact and inter-contact times, and network diameter.

We extracted multiple lessons that we will to use to improve the next implementation: (1) we must simplify the query process as our system spends too much time accessing content in the memory, (2) we must improve the scanning and connection techniques, and (3) we need to add a TTL hop count for all messages. This first experiment will also help us to improve the functionality and the usability of the GUI.

We expect the next experiment to involve many more participants. The next version of MobiClique will be made publicly available in order to build a community of users, design more applications, and to collect more data to support research in ad hoc opportunistic communication. We hope MobiClique will foster the vision of going beyond building ad hoc *contacts* to building ad hoc *communities* that are enriched by a new generation of ad hoc applications.

[3]http://www.cityware.org.uk/

[4]http://www.dodgeball.com/

[5]http://www.meetmoi.com

[6]http://www.jaiku.com/mobile/

[7]http://twitter.com/

[8]http://www.wired.com/culture/lifestyle/news/1998/05/12342

9. REFERENCES

[1] Bluetooth Special Interest Group (SIG). *Specification of the Bluetooth System. Core Package version: 1.2*, November 2003.

[2] J.-C. Cano, J.-M. Cano, E. González, C. Calafate, and P. Manzoni. How does energy consumption impact performance in bluetooth? *SIGMETRICS Perform. Eval. Rev.*, 35(3):7–9, 2007.

[3] A. Chaintreau, A. Mtibaa, L. Massoulié, and C. Diot. Diameter of opportunistic mobile networks. In *Proceedings of ACM Sigcomm CoNext*, December 2007.

[4] N. Eagle and A. Pentland. Social serendipity: Mobilizing social software. *IEEE Pervasive Computing*, 4(2), 2005.

[5] K. Fall. A delay-tolerant network architecture for challenged internets. In *SIGCOMM '03: Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications*, 2003.

[6] P. Hui, A. Chaintreau, J. Scott, R. Gass, J. Crowcroft, and C. Diot. Pocket switched networks and the consequences of human mobility in conference environments. In *Proceedings of ACM SIGCOMM first workshop on delay tolerant networking and related topics*, 2005.

[7] V. Kostakos, E. ONeill., and A. Shahi. Building Common Ground for Face to Face Interactions by Sharing Mobile Device Context. *Lecture Notes in*

Computer Science, 3987, 2006.

[8] O. Mukhtar and J. Ott. Backup and bypass: Introducing dtn-based ad-hoc networking to mobile phones. In *Proceedings of the RealMAN Workshop*, 2006.

[9] R. Nair and M. Davis. Bluetooth pooling to enrich co-presence information. *Adjunct Proceedings of the 7th International Conference on Ubiquitous Computing*, 2005.

[10] T. Nicolai, E. Yoneki, N. Behrens, and H. Kenn. Exploring social context with the wireless rope. In *Proceedings of the OTM Workshop MONET*, 2006.

[11] E. ONeill, T. Kindberg, AF gen Schieck, T. Jones, A. Penn, and DS Fraser. Instrumenting the city: Developing methods for observing and understanding the digital cityscape. In *Proceedings of UbiComp*, 2006.

[12] P. Persson and Y. Jung. Nokia sensor: from research to product. In *DUX '05: Proceedings of the 2005 conference on Designing for User eXperience*, 2005.

[13] A-K Pietiläinen. Measuring human mobility. Master's thesis, Helsinki University of Technology, 2007. Available from: `http://www.thlab.net/~apietila/hut.pdf`.

[14] J. Su, J. Scott, P. Hui, J. Crowcroft, C. Diot, A. Goel, E. de Lara, M. How Lim, and E. Upton. Haggle: Seamless networking for mobile applications. In *Proceedings of UbiComp*, 2007.