2012 Senior Thesis Project Reports

Iliano Cervesato[*] **Kemal Oflazer**[*] **Majd Sakr**[*]
Mark Stehlik[†] **Khaled Harras**[*] **Abderrahmen Mtibaa**[*]

May 2012
CMU-CS-QTR-115

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

[*]Qatar campus. [†]Computer Science Department

The editors of this report include the members of the Senior Thesis Committee on the Qatar campus and the students' advisors.

Abstract

This technical report collects the final reports of the undergraduate Computer Science majors from the Qatar Campus of Carnegie Mellon University who elected to complete a senior research thesis in the academic year 2011–12 as part of their degree. These projects have spanned the students' entire senior year, during which they have worked closely with their faculty advisors to plan and carry out their projects. This work counts as 18 units of academic credit each semester. In addition to doing the research, the students presented a brief midterm progress report each semester, presented a public poster session in December, presented an oral summary in the year-end campus-wide Meeting of the Minds and submitted a written thesis in May.

Keywords: Adaptive Networks, End-to-end Communication, Intermittent Networks, Hybrid networks.

Contents

SCOUT: Extending the Reach of Social-Based Context-Aware Ubiquitous Systems

Dania Abed Rabbou, Abderrahmen Mtibaa, Khaled A. Harras
School of Computer Science
Carnegie Mellon University
{dabedrab,amtibaa,kharras}@cmu.edu

***Abstract*—The proliferation of social-networks, localization systems, and high-end mobile devices has created a fertile ground for the development of systems that are aware of interests and adaptive to location. With the burgeoning of the domain of social-based context-aware systems, numerous challenges are becoming of increased importance. One such challenge, not addressed so far by the research community, is end-to-end communication between ubiquitous systems and their users. Communication in existing systems is either infrastructure-based or infrastructure-less. Infrastructure-based communication require users to stay connected to a server and thus assumes the availability of internet connectivity everywhere. In reality, internet connectivity may be absent, charged, energy-consuming, heterogeneous, and overloaded. As an alternative, infrastructure-less communication enables users to obtain information from neighbouring devices but the availability of information and the extent of its dissemination are dictated solely by user mobility and contacts.**

We realize the need for a new hybrid mode that leverages the centralized and distributed communication modes. Using the hybrid mode, a ubiquitous system disseminates a message to interested connected and unconnected users by first selecting a subset of connected users sufficient to reach the unconnected ones and secondly by forwarding the message from the recipients to neighbouring users who are most likely to meet the unconnected ones. We implemented a social-based context-aware ubiquitous system, SCOUT, as a tool to test our hybrid communication paradigm as well as our selection and forwarding algorithms. We validate our implementation of SCOUT, compare the performance of the hybrid paradigm against the existing ones and evaluate our forwarding technique using real datasets based on two metrics important in this domain namely success percentage and overhead. We show that the hybrid paradigm achieves a success nearly 80% and 25% more than the centralized and distributed paradigms respectively. We also show that our forwarding technique reduces overhead by as much as 50%.

I. Introduction

Social networking has recently gained unprecedented popularity among Internet users. The advent of smart phones equipped with mobile technology such as WiFi, 3G, 4G, and sensors, has exponentially increased the number of mobile phone users thus enriching the content of online social networks such as Facebook. This has increased the importance of social- and context-aware systems that leverage social information with location awareness to provide personalized services. Figure 1 illustrates how such systems renders Bobs shopping activity in a mall an exciting endeavor. Bob receives notifications based on his Facebook profile and location in positions A, B, and C. He is able to tune the world around

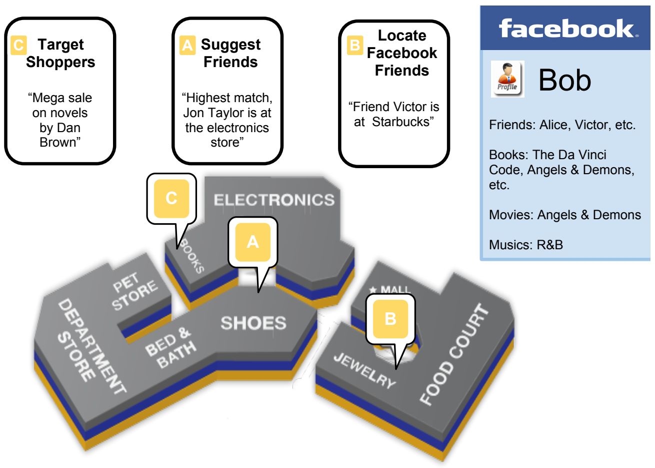

C Target Shoppers
"Mega sale on novels by Dan Brown"
A Suggest Friends
"Highest match, Jon Taylor is at the electronics store"
B Locate Facebook Friends
"Friend Victor is at Starbucks"
facebook
Bob
Friends: Alice, Victor, etc.
Books: The Da Vinci Code, Angels & Demons, etc.
Movies: Angels & Demons
Musics: R&B
ELECTRONICS
BOOKS
PET STORE
DEPARTMENT STORE
BED & BATH
SHOES
MALL
JEWELRY
FOOD COURT

Fig. 1. Bob is shopping in a mall and receiving real-time context-aware recommendations based on his on-line social-network profile.

him to his liking, discover surrounding people and places, and seize various social/business opportunities.

To receive context-aware recommendations in real-time, mobile devices must be connected to wireless networks. However, in reality, many devices are not connected all the time due to the absence of coverage or high energy consumption and cost (like 3G service). Moreover, wireless connectivity is intermittent in heterogeneous environments in which users move from one network to another and can suffer from contention under high load.

Previous systems in this domain were either infrastructure-based or infrastructure-less. Serendipity [10], Live Social Semantics [2], SPETA [11], and SocialFusion [5] are infrastructure-based systems that collect and store users social information in a server to provide context-aware recommendations about other users within Bluetooth range. While such a paradigm guarantees timeliness, it assumes persistent connectivity of users at all times. As an alternative paradigm to achieving the same functionality, WhozThat [4], Ad-Hoc Smart Spaces [6] and MobiClique [15] proposed a infrastructure-less Bluetooth-based architecture in which devices obtain context-aware messages from other devices in range. Successfully overcoming the connectivity requirement, this paradigm compromises on the extent of information dissemination and timeliness because delivery is solely dictated by user mobility and meetings.

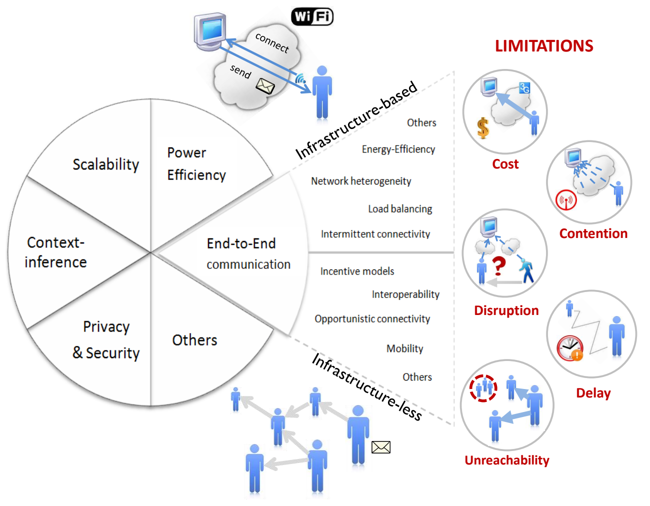


Fig. 2. Challenges in the domain of social context-awareness and the challenge we address, namely *end-to-end communication*. Communication either assumes and requires a fixed infrastructure or is infrastructure-less. Each has a set of drawbacks and therefore we propose a hybrid paradigm.

The shortcomings of existing paradigms call for a new hybrid paradigm that enables the reachability to unconnected users while still maintaining timeliness. Therefore, among the various critical challenges in the domain of social-context-awareness as depicted in Figure 2, we choose to address the end-to-end communication challenge.

We adopt a pragmatic approach towards solving the communication challenge. On one hand, we propose theoretical algorithms to realize the new hybrid paradigm and on the other hand, we implement a real ubiquitous system, SCOUT [1], that helps test and evaluate one component of our proposed solution and in the long term, will be deployed to collect real social and mobility data for the benefit of the research community.

Our main contribution is the proposal and evaluation of a new communication paradigm and social opportunistic forwarding technique that out-performs existing paradigms and well-known forwarding techniques respectively. Using a real-trace-based evaluation, we show that the hybrid paradigm achieves a success percentage nearly 80% and 25% more than the centralized and distributed paradigms respectively. We also show that our social forwarding algorithm achieves a success percentage comparable to the other techniques but at a reduced cost of nearly 50%. We also build a real social-based context-aware ubiquitous system to implement our selection algorithm.

The rest of the paper is divided as follows. Section II formally defines and illustrates the end-to-end communication problem. In section III we discuss the related work of an important component of hybrid communication which is the field of infrastructure-less or opportunistic communication. Next, we present two algorithms, selection and opportunistic forwarding, that achieve the hybrid paradigm in section IV. We introduce our ubiquitous system SCOUT [1] in section V along with its architecture and validation. In section VI we evaluate our opportunistic forwarding algorithm. Finally, we present our conclusion and future work in section VII.

II. Problem Definition

End-to-end communication generally refers to the transfer of data packets from a source to a destination. In the domain of ubiquitous systems, the packets are context-aware messages that originate from a server or devices and are destined to devices whose users are interested in the message. The challenge is to deliver the messages to the maximum number of connected and unconnected users in the shortest possible time and minimum cost.

More formally, Let N be the set of all users registered with a central system S. N is partitioned into two sets N_C, the set of connected users, and N_D, the set of unconnected users. *Given an interest I_k and a related message m_{I_k} generated by S at time t, we need to deliver the message to $N_{I_k} \subseteq N_C \cup N_D$, the set of users interested in I_k, with minimum delay δt and minimum number of message replicas.*

Much research has been done for minimizing the delay in end-to-end communication between a user and a server in an infrastructure-based environment. Therefore, connected users can directly receive messages with the minimal delay.

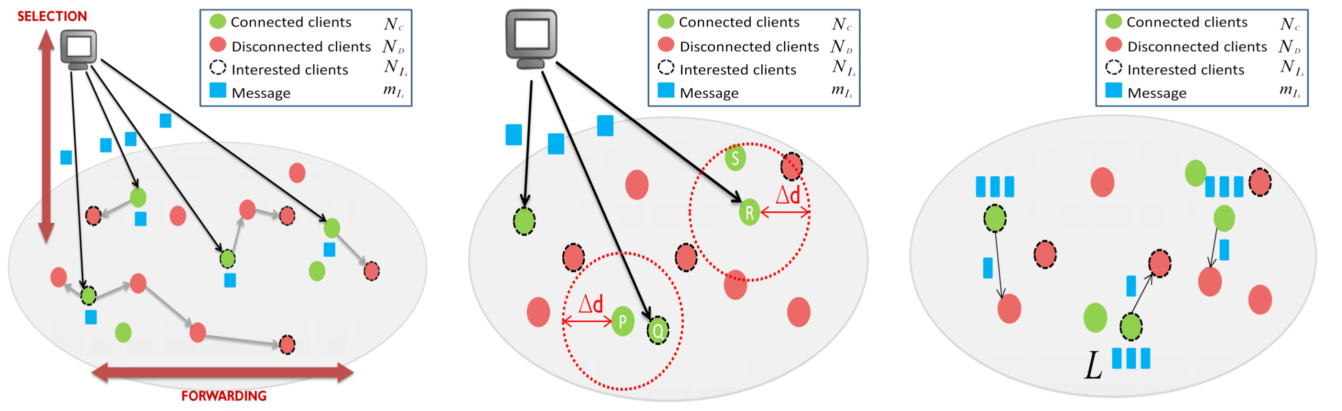
Unconnected users, however, can acquire messages opportunistically using an infrastructure-free paradigm. *Opportunistic communication* refers to the exchange of messages between users when they come in communication range of one another. The research community has proposed general techniques for opportunistic communication which we discuss in the following section. Note that in the following sections, we refer to users as nodes.

III. Related Work

Opportunistic communication techniques can be classified into two categories: Non-social and Social.

A pioneer work in the non-social category is Epidemic Routing [17] in which a message-carrying node transfers a replica of the message to every node it encounters within communication range. Essentially, this technique unconditionally floods a message throughout the entire network. While, it overflows node buffers with messages, it achieves the minimal delay to reach destination nodes. Evolutions of Epidemic Routing are MaxProp [7] which prioritizes messages to flood and Spray and Wait [16] which sprays or transfers only some L replicas of a message to nodes within communication range and waits until one of the nodes meets the destination. In order to reduce the cost or number of replicas, RAPID [3] and PRoPHET [19] proposed probabilistic techniques in which nodes transfer replicas only to other nodes that have a high probability of encountering a destination. This information is obtained from the history of encounters. Another technique is Message Ferry [18] that deploys ferries (like robots or vehicles) to carry and deliver messages between nodes.

The category of social techniques is a recent one which exploits social-relationships among nodes to identify to transfers replicas to most popular nodes. In SimBet [9] and BubbleRap [12], each node is assigned a centrality value which is



(a) Selection and forwarding (b) Selection by the server (c) Forwarding by the message-carrying nodes

Fig. 3. Illustration of the selection and forwarding phases

equal to the number of its social friends and message replicas are transferred to nodes with high centrality. PeopleRank [14] is another social technique inspired by PageRank that gives weights to nodes according to the number of important nodes, nodes with high weights, they are socially linked to.

IV. Our Solution

To disseminate a context-aware message originating from a server to a set of interested connected or unconnected nodes, our solution consists of two phases: *Selection* and *Opportunistic Forwarding*. Opportunistic forwarding refers to the exchange of messages between nodes when they come in communication range. While selection is performed by the server, opportunistic forwarding is performed by all the message-carrying nodes as shown Figure 3 (a).

A. Selection

We assume that the server, S, stores the social profiles consisting of basic information, friends and interests of all registered nodes N. S also builds and maintains a social graph of N nodes in which a direct link exists between two nodes if they are listed as friends in their social profiles. Using this graph, S computes the betweenness centrality as:

$$\forall n \in N, C(n) = \sum_{i=1}^{N} \sum_{j=1}^{i-1} \frac{p_{ij}(n)}{p_{ij}} \qquad (1)$$

where p_{ij} is the number of shortest paths between i and j and $p_{ij}(n)$ is the number of such paths that pass through n. Intuitively, the betweenness centrality measures the extent to which a node can facilitate communication between other nodes. At any time t, $\forall n \in N, n \in N_C \vee n \in N_D$. When a message m_{I_k} is generated by S, it computes the subset of interested nodes N_{I_k} as follows:

$$\forall n \in N, n \in N_{I_k} \quad if \ I_k \in Interests(n) \qquad (2)$$

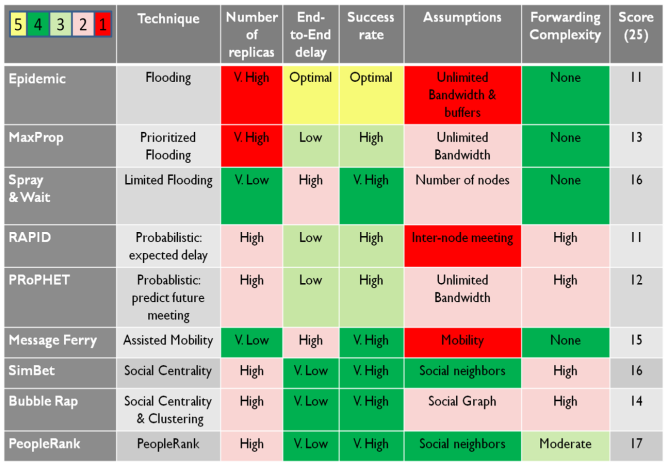
S sends the message to all the connected nodes in $N_{I_k} \cap N_C$. However, unconnected nodes in $N_{I_k} \cap N_D$ must obtain the message opportunistically from other message-carrying nodes. To increase the chances of message delivery, S transfers the message to additional connected nodes as follows (depicted in Figure 3 (b)). $\forall n \in N_C - N_{I_k}$:

- Find all nodes in the circular area whose center is n and radius is $\triangle d$.
- For each such node m,
  1) if m received m_{I_k} then a copy of the message exists in that area and hence n will not receive it. For example, node P does not receive the message because the interested and connected node Q had received it.
  2) if m did not receive m_{I_k} then n will receive a copy if $C(n) > C(m)$, otherwise m will receive it. For example, node R receives the message because $C(R) > C(S)$.

B. Opportunistic Forwarding

Once the selected nodes receive the message, they begin forwarding the message opportunistically so that it eventually reaches the interested nodes. We assume that all nodes have unlimited buffers and they are willing to be message carriers. We use one or more of the opportunistic communication techniques discussed in Section III. We qualitatively evaluate and compare the techniques based on five metrics (as shown in Figure 4) that are important to the domain of social-context-aware ubiquitous systems . Each metric is assigned a color according to its performance where yellow and red signify the optimal and worst performance and are assigned 5 and 1 respectively. The overall performance of the technique is then computed as the sum of the values assigned to each metric.

We note that the social techniques, particularly PeopleRank, perform well in terms of end-to-end delay and success rate. However, the cost in terms of number of replicas can be improved. For this, we consider non-social techniques Spray-



5 4 3 2 1	Technique	Number of replicas	End-to-End delay	Success rate	Assumptions	Forwarding Complexity	Score (25)
Epidemic	Flooding	V. High	Optimal	Optimal	Unlimited Bandwidth & buffers	None	11
MaxProp	Prioritized Flooding	V. High	Low	High	Unlimited Bandwidth	None	13
Spray & Wait	Limited Flooding	V. Low	High	V. High	Number of nodes	None	16
RAPID	Probabilistic: expected delay	High	Low	High	Inter-node meeting	High	11
PRoPHET	Probablistic: predict future meeting	High	Low	High	Unlimited Bandwidth	High	12
Message Ferry	Assisted Mobility	V. Low	High	V. High	Mobility	None	15
SimBet	Social Centrality	High	V. Low	V. High	Social neighbors	High	16
Bubble Rap	Social Centrality & Clustering	High	V. Low	V. High	Social Graph	High	14
PeopleRank	PeopleRank	High	V. Low	V. High	Social neighbors	Moderate	17

Fig. 4. Qualitative comparison between opportunistic techniques

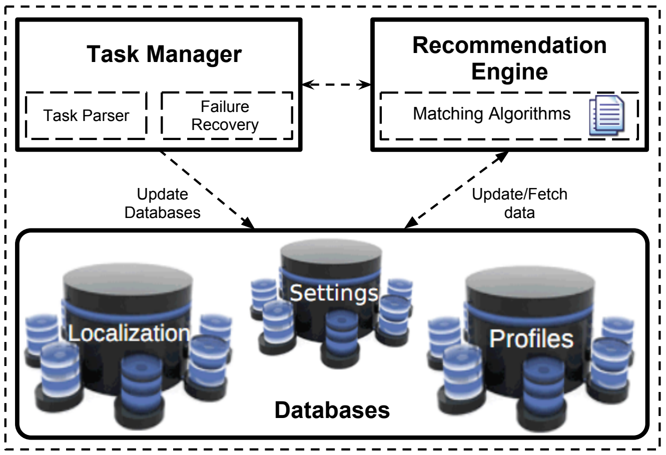


Fig. 5. High level architecture of SCOUT

and-Wait and Message Ferry which have a very low cost. Therefore, we adopt the highest score non-social technique, Spray-and-Wait, and social technique, PeopleRank. We combine these two forwarding techniques to create a protocol, **Social Spray and Wait (SSNW)**, expounded below.

Each node in the network computes its PeopleRank as follows:

$$PeR(n) = (1 - d) + d \sum_{m \in SN(n)} \frac{PeR(m)}{|SN(m)|} \tag{3}$$

where $SN(n)$ is the set of nodes that node n physically encountered, referred to as social neighbours, and d is a damping factor defined as the probability that the social relation between nodes helps improve the rank of these nodes.

When message-carrying node n encounters another node m i.e. they become within communication range, the following steps occur as shown in Figure 3 (c):

- m sends its list of interests to n. If $I_k \in Interests(m)$, n sends a replica to m. Otherwise, m is not interested in the message but can act as a carrier.
- m sends the list of messages in its buffer and its PeopleRank. n sends a replica to m if $m_{I_k} \notin Messages(m)$ and $PeR(m) > PeR(n)$. A higher PeopleRank value indicates that a node is more socially linked and is hence more likely to meet the destinations.

All message-carrying nodes repeat this process of forwarding until they have each sprayed a total of L copies of the message in the network.

V. Implementation of a Ubiquitous System: SCOUT

While attempting to theoretically solving the communication challenge by reviewing the literature and proposing algorithms, we implement a prototype of a social-based context-aware ubiquitous system, SCOUT [1], that adopts an infrastructure-based communication paradigm. Our intention is to use SCOUT to implement and evaluate our selection algorithm as detailed in section IV-A, and integrate it with the forwarding algorithm to fully realize a system based on a hybrid paradigm. In the following sections, we present a brief description of the components of SCOUT and its stress-testing results. For more details on its architecture, implementation and other testing results, please refer to the Appendix.

A. Components

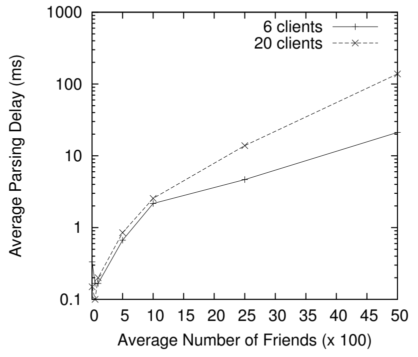
SCOUT is based on a client-server architecture where the client retrieves social information from Facebook including basic information, friends and interests, and sends it to the server. The server maintains the social, location and custom-settings data in databases and performs matching to provide real-time recommendations or messages to interested users. As shown in Figure 5, SCOUT consists of a task manager, three databases, and recommendation engine.

The *task manager* receives and delegated tasks from clients to the task parser and recommendation engine as well as sends real-time messages back to interested- connected- clients. The *task parser* parses social profiles and updates the profiles database. The *databases* store social profiles, location and custom settings of all registered users. The *recommendation engine* is responsible for periodically examining the context of available messages and traversing the databases to generate the set of users interested in in each.

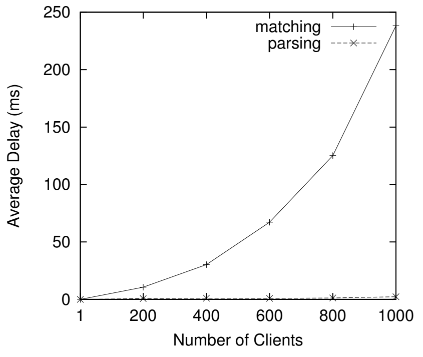
B. Stress-Testing

To assess the stability of SCOUT, we subject the server to a high load of concurrent client connections and measure ***(i)*** *average parsing delay $avgP_{delay}$:* Average time incurred by the task parser to parse a given user's social information, and ***(ii)*** *Average matching delay $avgM_{delay}$:* Average time incurred by the recommendation engine to identify if a client is interested in a message. For measuring $avgM_{delay}$, we assume that there exists a message for which we require to examine client entries in the social, location and settings databases to identify interested ones.

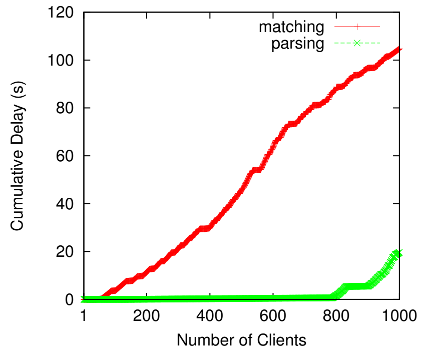
Figure 6 (a) plots $avgP_{delay}$ for 6 and 20 concurrent clients with respect to different profile lengths which are predominated by the average number of friends (y-axis in log



(a) Comparison between $avgP_{delay}$ of 6 and 20 clients



(b) $avgP_{delay}$ and $avgM_{delay}$



(c) Cumulative delays

Fig. 6. Results for stress testing SCOUT to measure its stability and robustness

scale). We observe that the $avgP_{delay}$ exponentially increases with the increase in profile size. However, the delay for very large profiles remains reasonably small; less than 130 ms to parse 20 profiles with an average number of friends 5000.

Figure 6 (b) plots $avgP_{delay}$ and $avgM_{delay}$ for 1 to 1000 concurrent clients connections. It is clear that $avgP_{delay}$ is almost constant. On the other hand, $avgM_{delay}$ exponentially rises with an increase in the number of clients. This is mainly because the matching is more time-intensive task that involves accessing and traversing databases as well as matching interests of clients to the context of messages. In the future, we will investigate more efficient matching techniques.

Figure 6 (c) plots the cumulative delays and depicts that the cumulative parsing delay is negligible compared to the cumulative matching delay. Eighty percent of the parsing tasks execute in less than 0.1 ms. Despite its linear increase with the number of clients, the cumulative matching delay curves down for the last 20% because of two reasons: the number of new clients is not significantly increasing and the system gives the matching tasks more priority over the parsing ones. For other testing results, please refer to the Appendix.

VI. Evaluation of Opportunistic Forwarding

As presented in section IV, our solution consists of two phases: ***(i)*** *selection:* Identifying the set of recipients, and ***(ii)*** *forwarding:* disseminating the message based on social information.

In the previous section, we presented our prototype SCOUT that we build for implementing and testing phase (i). However, due to time constraints we did not implement the selection process. In this section, we present our evaluation of the opportunistic forwarding phase including set-up, experimental datasets and results.

A. Set-up

We use a simulator to emulate the mobility of users and the contacts between them. We initially implemented our algorithm as discussed in Section IV-B on a well-known simulator in opportunistic forwarding, the ONE [1] . However, because we could not control the users traffic as we needed, we implemented our own simulator. The simulator reads a *contact file* which contains entries of Bluetooth encounters between pairs of nodes and performs forwarding based on three techniques:

- *Epidemic:* Here each a message-carrying node forwards a copy of the message to every node it encounters. Hence, the message is unconditionally flooded in the network and the forwarding continues even after all destinations receive a copy.
- *Spray and Wait (SNW):* This achieves controlled flooding of a message. A message-carrying node forwards a copy to the first L nodes it encounters. The spray value L depends on the network size and is chosen based on heuristics.
- *Social Spray and Wait (SSNW):* This uses social information for forwarding decisions and works as discussed in section IV-B. In short, each node is aware of its social friends, stores its PeR value, updates it when encountering a social friend and forwards a copy to encountered nodes of higher PeR.

B. Experimental Data Sets

The contact files we use for the simulation process are two real data sets, Haggle and St Andrews, which are two state-of-art human mobility traces publicly available at CRAWDAD [2] [3]

InfoCom06 Dataset: This trace was collected with 78 participants during the IEEE Infocom 2006 conference. Participants were asked to carry experimental iMote devices at all times. The devices performed Bluetooth scans every 2 seconds and logged contacts with other experimental devices (referred to as internal contacts) and other external Bluetooth devices including cell phones, PDAs etc. Besides the participants,

[1]http://www.netlab.tkk.fi/tutkimus/dtn/theone/
[2]crawdad.cs.dartmouth.edu/meta.php?name=cambridge/haggle
[3]crawdad.cs.dartmouth.edu/meta.php?name=st_andrews/sassy

Data set	Data Collected	Participants	Duration	Location
Haggle	Bluetooth Sightings	98	3 days	Infocom06
St Andrews	Bluetooth Sightings & FB Friends	27	79 days	Univ. of St Andrews

TABLE I
SUMMARY OF THE TWO DATASETS USED FOR EVALUATION OF OPPORTUNISTIC FORWARDING

the experiment included 20 stationary nodes which have more powerful battery and extended radio range (around 100 meters).

St Andrews Dataset: This trace consists of sensor mote encounter records and corresponding Social Network data (Facebook friends) of a group of 27 participants at University of St Andrews. The participants, students and staff, carried iMote devices during a period of 79 days. The devices performed Bluetooth scans every 6.67 seconds and logged contacts with other experimental devices. In addition, the trace also provides the participants Facebook friends that can could be used to generate a social graph.

C. Metrics and Parameters

We measure two important metrics in the domain of social- and context-awareness namely ***(i)*** *Success Percentage:* Percentage of the destinations or interested nodes in N_{I_k} that receive a copy of m_{I_k}, and ***(ii)*** *Overhead:* Number of non-interested nodes in $N - N_{I_k}$ that receive a copy of m_{I_k} in transit.

For each metric, we vary the parameters ***(i)*** *Spray value L:* The total number of copies each selected nodes must spray, and ***(iv)*** *Delay:* The time after which the message is dropped from the buffers of all nodes and is no longer forwarded. This delay is usually small for ubiquitous systems due to their real-timeliness.

D. Experiments

In this section we present the evaluation results of two experiments expounded below. In all the experiments, the delay parameter assumes a maximum value of one day because of the real-timeliness requirement of ubiquitous systems.

1) Comparison of Communication Paradigms: The aim of this experiment is to compare the performance of the three communication paradigms namely infrastructure-based, infrastructure-less and hybrid. We use the Haggle dataset to measure their performance in terms of success percentage. Among the 98 nodes, we randomly select the subset N_C and add the rest of the nodes to N_D.

Using the infrastructure-based paradigm, all the nodes in N_C receive m_{I_k}. For the infrastructure-less paradigm, a node is randomly selected from N_C to receive the message and in turn forwards it to other nodes. Finally, in the hybrid paradigm, all nodes in N_C receive the message (like the infrastructure-based) and each node then forwards it (like the infrastructure-less).

In Figure 7, we plot the success percentage for each paradigm against a delay of upto 1 day. Figure 7(a) uses the Epidemic forwarding technique. The hybrid paradigm reaches 20% more than the distributed paradigm and 95% more than the centralized paradigm. Figure 7(b) uses the SNW forwarding technique. Similarly, the hybrid paradigm reaches 30% than the distributed paradigm and 70% more than the centralized paradigm. Therefore, the hybrid paradigm outperforms the existing paradigms for all delays and the two forwarding protocols.

2) Comparison of Forwarding Protocols: Next we compare our forwarding protocol SSNW detailed in Section IV-B to Epidemic and SNW. For this we use the St Andrews dataset that additionally provides social information needed for forwarding decisions in SSNW. We measure the two metrics, success percentage and overhead, and vary the two parameters, L and delay.

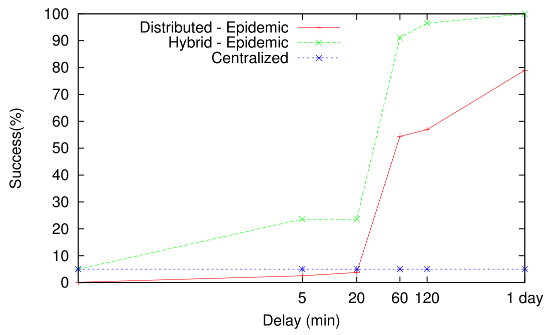
While Figure 8 (a) and (b) use a spray value $L = 4$, Figure 8 (c) and (d) use $L = 8$. Figures (a) and (c) illustrate that SSNW achieves a comparable success percentage to Epidemic and SNW. In fact, with only a limited number of copies, our success percentage is close to the optimal value attained by Epidemic. However, (b) and (c) show that SSNW achieves nearly 50% reduction in the overhead compared to Epidemic and 20% compared to SNW. Therefore, SSNW attains a comparable success percentage at a reduced cost.

VII. Conclusion & Future Work

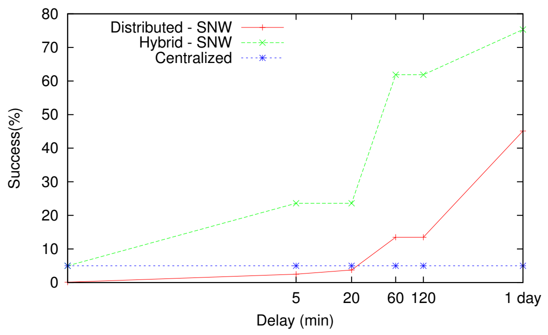
In order to extend the reachability of ubiquitous systems, we have proposed a new hybrid communication paradigm that leverages the existing infrastructure-based and infrastructure-less paradigms. We have also proposed a forwarding protocol (SSNW) that uses social data for forwarding decisions.

We have compared the hybrid paradigm to the existing ones using real datasets and we have shown how it significantly increases the success percentage of ubiquitous systems. We have then compared SSNW to well-known opportunistic protocols, Epidemic and Spray and Wait (SNW), and showed that SSNW achieves comparable performance with a much lower overhead.

In the future, we plan to re-run the same experiments using scaled datasets with larger number of participants. We plan to implement the selection algorithm as discussed in section IV-A on SCOUT and integrate it with the opportunistic forwarding algorithm as discussed in section IV-B to achieve a real system based on the hybrid paradigm. We then plan to deploy SCOUT in the Carnegie Mellon University in Qatar to gather mobility and social data and make it available to the research community for various purposes. In the long term, we hope to address similar challenging problems in the domain.

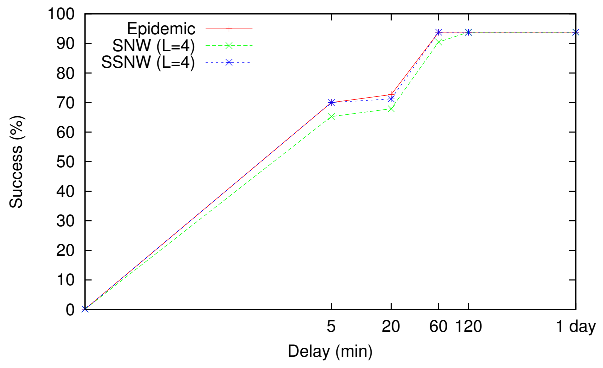


(a) Forwarding based on Epidemic algorithm

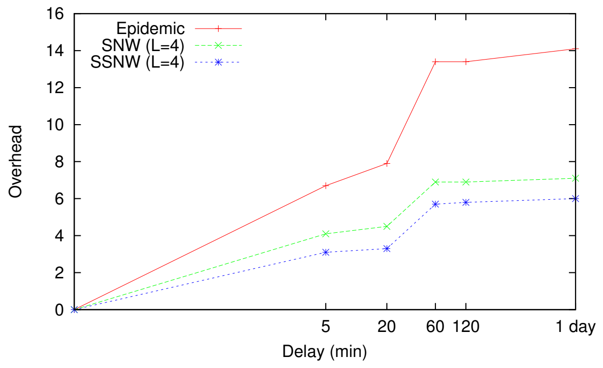


(b) Forwarding based on the Spray and Wait (SNW) algorithm ($L = 4$)

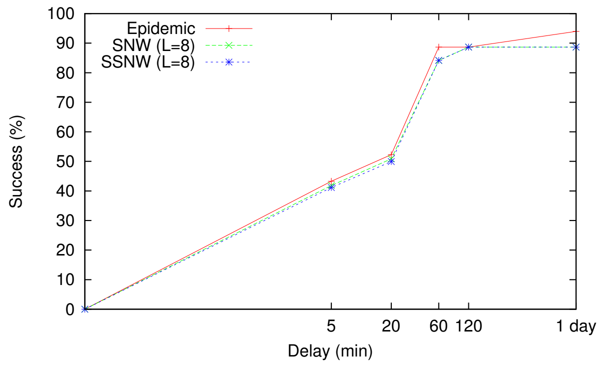
Fig. 7. Results of experiment 1 which measures the performance of the three communication paradigms namely infrastructure-based, infrastructure-less and hybrid using the success percentage metric. The two graphs differ by the forwarding protocol used by the message-carrying nodes. As shown above, the hybrid paradigm out-performs the existing paradigms.



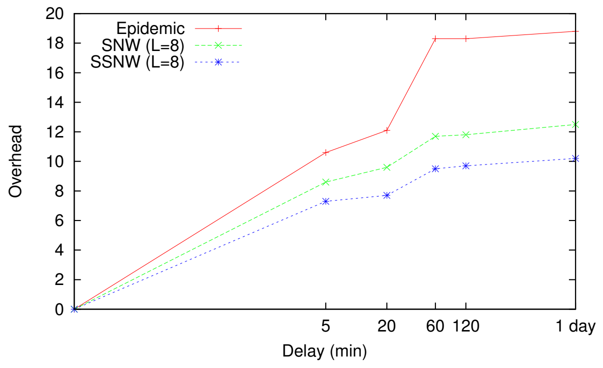
(a)



(b)



(c)



(d)

Fig. 8. Results of experiment 2 which compares the performance of our SSNW forwarding protocol to Epidemic and Spray and Wait (SNW). Our SSNW protocol achieves comparable performance at a much lower overhead.

APPENDIX

A. SCOUT Architecture

In this section, we describe the architecture of SCOUT. We first provide a high level system overview where we discuss the fundamental functionalities that enable mobile users to bridge the gap between their social world and daily life. Afterwards, we describe the various components of our architecture that achieve these functionalities.

1) System Overview: SCOUT is a generic system encompassing the fundamental functionalities of a social context-aware ubiquitous system that enables a multitude of mobile applications. The system is based on a client-server architecture where the server, shown in Figure 5, consists of a task manager, a recommendation engine, and three databases that store social profiles, location information, and client settings.

The server operates in two modes (*i*) $reactive$ and (*ii*) $proactive$. It is in the reactive mode when clients running different mobile applications built on top of SCOUT seek recommendations and explicitly request the server to provide them. For example, Bob in Figure 1 can request a friend-finding application to find his friends in the vicinity. On the other hand, the server is in the proactive mode when recommendations seek clients. The server pro-actively sends recommendations to relevant clients depending on their availability. SCOUT, for instance, can pro-actively contact Bob about Dan Brown's books promotion based on his interests and presence in the mall.

In general, clients connected to SCOUT's server via applications, submit tasks and await recommendations sent proactively, or in response to tasks explicitly submitted by the user. Tasks are received by the *task manager* that assigns a new id, T_{id}. The task manager handles concurrent tasks and tracks the completion of each task by updating its status at time t, $T_{id}(t)$.

The type of incoming tasks pertain to contextual data or recommendation requests. Depending on its type, the task manager delegates a task either to the *task parser* or *recommendation engine*. The task parser parses a task and updates the appropriate databases. The recommendation engine, on the other hand, runs a set of matching algorithms, generates recommendation results, and dispatches the results back to the client via the task manager. On successful completion, the task manager sets $T_{id}(t)$ to *DONE* and removes the task from its list. In the case of processing errors, the task manager reports back exceptions to the client.

We now discuss the details of the task manager, recommendation engine, and the databases needed. Algorithm 1 depicts and summarizes the server operation in reactive and proactive modes, the functionality of these components, and the interactions between them.

2) Task Manager: The task manager is responsible for receiving and delegating tasks from the clients to the task parser and recommendation engine, sending reactive and proactive recommendation results back to clients, and handling failure recovery.

The task manager uses the task parser to process tasks pertaining to contextual information like social, location, and settings data by manipulating and updating the corresponding databases. The task manager delegates the other tasks, namely recommendation requests, to the recommendation engine. Both the task parser and recommendation engine communicate results back to the task manager. The manager awaits results in the form of acknowledgements, output results, or exceptions. Acknowledgements are then sent by the task parser on successful database manipulation operations, output data by the recommendation engine in order to forward to the client application, and exceptions by both. The $T_{id}(t)$ of a task that successfully finishes is set to *DONE* and the task is removed from the processing pool.

The task manager handles failures using the failure recovery sub-component. There are two types of failures: task failure and connection disruption failure. Various possible errors lead to task failure in the task parser or recommendation engine, one of which is erroneous input data. In this case, the task parser and recommendation engine return exceptions to report back to the client application. Communication failure occurs when, for example, device owners switch off their devices and are unreachable. Results generated in response to a client's request and other results generated pro-actively, are assigned a time-to-live period, ttl, computed based on application specific profiling and heuristics. The manager sets $T_{id}(t)$ to pending, establishes an opportunistic connection, and periodically resends data either until it expires or the connection is successfully re-established.

3) Databases: The databases store *client profiles*, *location* and *settings* data which we refer to as DB_{SOC}, DB_{LOC} and DB_{SET} respectively. While the task manager has write privileges to insert and update the databases, the recommendation engine with read privileges only accesses the stored information.

DB_{SOC}*:* The client profiles database contains social data derived from a social-networking website like Facebook or Google+. This includes basic information, friends, family, interests and activities.

DB_{SET}*:* This database stores additional settings information obtained from the subscribed clients and used by the matching algorithms to tune recommendations. A client subscribed to different applications has different settings per application. Such settings include the zoom or area of interest, ranking of personal interests, types and parameters for proactive notifications, recommendation thresholds, privacy and visibility settings, etc.

DB_{LOC}*:* The location database contains outdoor or indoor coordinates (x_t, y_t, z_t) at time t for all clients. SCOUT is not restricted to any localizing technology [13] [8] but is rather integratable with any. This allows users to dynamically set their interest zone and empowers applications built on top of SCOUT to use a suitable localizing technology that fulfils the prescribed location-granularity needed by an application. For instance, while it is sufficient for a marketing application to know about the "existence" of a user in an indoor space, a

Algorithm 1 SCOUT Server Operation

```
Definition: C is a client registered with SCOUT and assigned an identifier id. T is a task associated with C, identified by id, categorized by type and tagged with one of the following states (UPDATING, MATCHING, SENDING, PENDING, DONE) at time t.
1: $\forall\ C_{id}$ where $C_{id} \in \{1, ..., |DB_{SOC}|\}$
2: while $Is\_Connected(C_{id})$ do
3:   if $receive\_task(C_{id}, T_{id}, msg)$ then
4:                                         // This is the Reactive mode
5:     if $Type(T_{id}) == context\_data$ then
6:       $T_{id}(t) \leftarrow UPDATING$
7:       $Update(DB_{SOC|LOC|SET}, msg)$
8:     else
9:       $T_{id}(t) \leftarrow MATCHING$
10:      $C_{settings} \leftarrow Fetch(C_{id}, DB_{SET})$
11:      $matched\_clients\_list \leftarrow Match(C_{id}, C_{settings})$
12:    end if
13:                                        // This is the Proactive mode
14:  else
15:    $T_{id} \leftarrow Create\_task(C_{id})$
16:    $C_{settings} \leftarrow Fetch(C_{id}, DB_{SET})$
17:    $matched\_clients\_list \leftarrow Match(C_{id}, C_{settings})$
18:  end if
19:  $T_{id}(t) \leftarrow PENDING$
20:  $ttl \leftarrow$ precomputed period based on heuristics
21:  repeat
22:    if $Is\_reachable(C_{id})$ then
23:      $T_{id}(t) \leftarrow SENDING$
24:      $Send(C_{id}, matched\_clients\_list)$
25:      $T_{id}(t) \leftarrow DONE$
26:    end if
27:  until $(t > ttl) \lor (T_{id}(t) == DONE)$
28: end while
```

friend-matcher application requires high-granularity location to facilitate situated interactions.

4) Recommendation Engine: The recommendation engine receives recommendation requests from the task manager and examines their type; different client applications may request different recommendation information. For example, requests for friends within proximity and surrounding users with specific interests are two types of recommendation requests. Depending on the type, the engine retrieves data from the databases and elects different matching algorithms that satisfy the request. Finally, the engine dispatches the computed list of matches back to the client via the task manager. The summary of the operation of all the components within the server are shown in Algorithm 1.

B. SCOUT-Based Prototype

In this section, we begin with a discussion of the challenges faced in the realization of SCOUT. Afterwards, we discuss the implement a proof-of-concept prototype of SCOUT's server, as well as a mobile client application to test our client-server design and prototype.

1) Challenges: There were numerous challenges that needed to be addressed when implementing the design of SCOUT. The most important challenges include the following:

Disk vs. memory storage: The repository of profile data, location information and user settings are frequently accesses by the recommendation engine. This demands that the fetching delay is minimal. Databases are traditionally known for data storage. However, they require occasional disk access which may not exploit locality as much as memory-based solutions. The choice between *disk-based* and *memory-based* certainly impacts the responsiveness of SCOUT.

Load distribution: Appropriately distributing the load between the client and server is critical for a real-time system. The question is whether to assign the burden of fetching updated social information to the client application or the server. The former requires that the client periodically retrieve and dispatche social data to the server. The latter requires that the server stores sensitive client credentials and bounds the server to changeable communication APIs. For example, Facebook's recent upgrade from the REST API to the Graph API would change the implementation of the server[4]. Again, a trade off exists between communication and storage overhead.

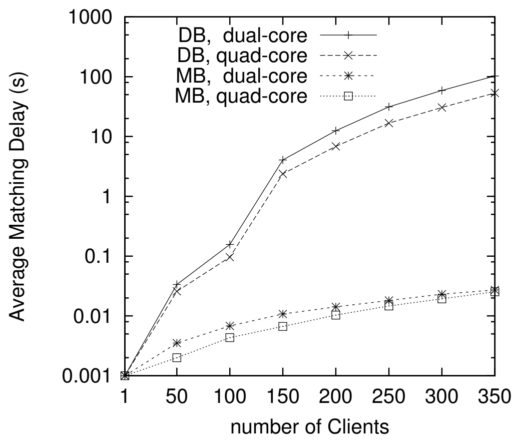
Client application type: The client application can either be a web-based or native application. Frameworks such as PhoneGap[5] make it easier to build native applications on different phone platforms using web-technologies. A comparison between the web-based, native, and PhoneGap-based, highlights the strengths and weaknesses of each [**?**]. While web-based applications are portable, native applications enable access to the phone's hardware and facilitate the design of convenient, complex and high-performance graphical user interfaces. PhoneGap-based applications serve as a middle-ground solution.

Privacy and Security: SCOUT houses social information of all users who entrust it for a high level of security. Hence, security of communication and storage is a key challenge in this domain. We do not address it in the current prototype.

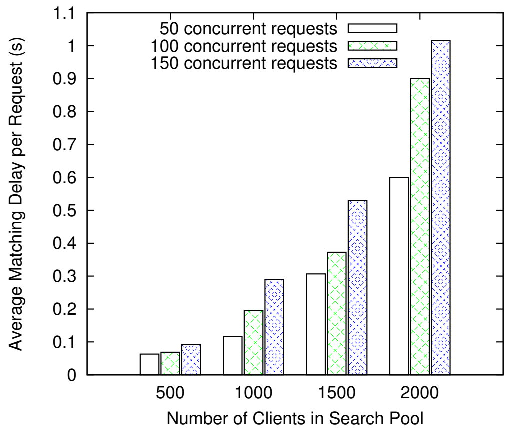
2) Implementation: We implement a multi-threaded server with the following components of our design: (*a*) task manager, (*b*) task parser, (*c*) databases, and (*d*) recommendation engine in reactive mode. We currently do not implement the recommendation engine in proactive. We also implement a mobile application that provides three functionalities namely finding existing friends, recommending new friends, and seeking targeted advertisements. For the purpose of the prototype, we generate mobility traces for clients in the building using the Random Way-point mobility model.

Server: The server delegates the task of fetching social information to the client. The storage mode of profile, location, and settings data affects the responsiveness of the recommendation engine and consequently the server. We propose two storage designs, *disk-based* (DB) and *memory-based* (MB). For the DB design, we create an Entity Relationship Diagram (ERD) that formally models the mesh of databases along with the interactions between them, and implement data structures that emulate the proposed ERD; the design is not shown due to space limitation. The MB design consolidates all information pertaining to a given client in one memory location. While

[4]http://developers.facebook.com/docs/reference/api/
[5]http://www.phonegap.com/



(a) $avgM_{delay}$ comparison between DB and MB storage designs



(b) $avgM_{delay}$ per request with large client pools

Fig. 9. Computation of $avgM_{delay}$

the MB design may incur redundancy, it helps aggregate the social information for ease of access by the recommendation engine.

Client: To test SCOUT, we implement a native Android mobile application that communicates with Facebook and supports the functionalities of finding existing friends, recommending potential ones, and seeking targeted advertisements. Using the client application, a user connects to Facebook to retrieve profile information including personal data, education, employment history, friends...etc, each as a JSON object. The client then juxtaposes all the JSON objects to create a composite one. Finally, it establishes a connection to the SCOUT server, sends the JSON object, awaits results on the various functionalities mentioned above. On receiving recommendation results and relevant information, the application processes these results and displays them in a convenient graphical user interface.

C. Prototype Evaluation

In this section, we briefly describe our experimental set-up, define the evaluation metrics, and in the end, we discuss our evaluation results.

1) Experimental Setup: The description of both the SCOUT server prototype and client application built on android are in the previous section. We run our prototype server on a quad-core Intel Xeon 2.83 GHz machine. We start the server and use a script that generates N_t client tasks at a parametrized rate λ_t for a given time t. In our experiments, we investigate the impact of high λ_t on the server's response delay; we run experiments with λ_t upto 1000 clients/sec that subjects the server to a high load. This assumption, while appears to be surrealistic, is used to evaluate SCOUT under high load. We synthesized Facebook social profiles as JSON objects generated using our profile generator *ProfGen*. *ProfGen* takes many parameters such as N the number of social profiles, nf the average number of friends, and ni the maximum number of interests. Unless stated otherwise, we choose $nf = 130$ which is the average number of friends in Facebook[6].

2) Performance Metrics: We evaluate SCOUT for *responsiveness*, which also reflects the system's stability and robustness. We measure the responsiveness of the prototype based on two metrics, *parsing delay* and *matching delay*. Parsing delay is the time incurred by the task parser to parse a given client messages (e.g., JSON objects of the client's social profiles). We measure the parsing delay P_{delay} as the time difference between t_{beg}, the time at which the parser receives a task T_{id} from the task manager, and t_{end}, the time it finishes parsing.

$$P_{delay}(T_{id}) = t_{end}(T_{id}) - t_{beg}(T_{id}) \tag{4}$$

Therefore, we compute the average parsing delay $avgP_{delay}$ described in equation 5.

$$avgP_{delay} = \frac{1}{N_t} \times \sum_{1}^{N_t} P_{delay}(T_{id}) \tag{5}$$

Similar to the parsing delay, we compute the matching delay M_{delay} and the average matching delay $avgM_{delay}$ as the period of time incurred by the recommendation engine to run one or more matching algorithms and sends back the corresponding results to the task manager. The overall server's delay, S_{delay}, is the summation of the parsing and matching delays.

$$S_{delay}(T_{id}) = P_{delay}(T_{id}) + M_{delay}(T_{id}) \tag{6}$$

3) Results: First we analyze the matching delay incurred by the recommendation engine. We run N_t tasks where N_t varies from 1 to 350 concurrent tasks. We compare the average matching delays $avgM_{delay}$ of (*i*) the DB and MB designs on the quad-core machine, and (*ii*) each design on dual- and quad-core machines. The results shown in Figure 9 (a) that

[6]http://www.facebook.com/press/info.php?statistics

the average matching delay for 350 clients decreases from approximately a minute in the case of the DB design to less than a second with the MB design. The performance gain with the MB design is nearly 100%. As for the impact of the number of cores on each design, the difference between the average matching delays on the dual- and quad-core machines is 33% for the DB design as compared to 8% with the MB design. This is mainly because the MB design consolidates information in same memory location and exploits locality. For the rest of paper, our analysis is based solely on the *MB design*.

Since the overall delay is predominated by the matching delay, we focus in our final experiment on computing the average matching delay per request, $avgM_{delay}$, to process 50, 100 and 150 concurrent recommendation requests. Each of these requests are matched against a varying client pool size of 500, 1000, 1500 and 2000. This helps us study the matching delay when SCOUT is in a stable state having its databases populated with a certain number of clients, a fraction of which request recommendations. Figure 9 (b) shows the responsiveness of SCOUT exponentially grows as the pool size grows. However, we observe very acceptable performance values where in extreme cases of responding to 150 requests, matching each one against 2000 profiles, each request is served within an average time of one second.

REFERENCES

[1] D. Abed Rabbou, A. Mtibaa, and K. Harras. Leveraging social networking and indoor localization for context-aware ubiquitous systems. In *Proceedings of the 5th IFIP international conference on new technologies, mobility and security*, NTMS. IEEE Communications Society, 2012.

[2] H. Alini, M. Szomszor, C. Cattuto, W. V. den Broeck, G. Correndo, and A. Barrat. Live social semantics. 2009.

[3] A. Balasubramanian, B. Levine, and A. Venkataramani. Dtn routing as a resource allocation problem. *SIGCOMM Comput. Commun. Rev.*, 37(4):373–384, 2007.

[4] A. Beach, M. Gartrell, S. Akkala, J. Elston, J. Kelley, K. Nishimoto, B. Ray, S. Razgulin, K. Sundaresan, B. Surendar, M. Terada, and R. Han. Whozthat? evolving an ecosystem for context-aware mobile social networks. *IEEE Network*, 22(4):50–55, 2008.

[5] A. Beach, M. Gartrell, X. Xing, R. Han, Q. Lv, S. Mishra, and K. Seada. Fusing mobile, sensor, and social data to fully enable context-aware computing. In *Proceedings of the Eleventh Workshop on Mobile Computing Systems &#38; Applications*, HotMobile '10, pages 60–65, New York, NY, USA, 2010. ACM.

[6] A. Brodt and S. Sathish. Together we are strong&#8212; towards ad-hoc smart spaces. In *Proceedings of the 2009 IEEE International Conference on Pervasive Computing and Communications*, pages 1–4, Washington, DC, USA, 2009. IEEE Computer Society.

[7] J. Burgess, B. Gallagher, D. Jensen, and B. Levine. Maxprop: Routing for vehicle-based disruption-tolerant networking. In *Proceedings of IEEE Infocom*, 2006.

[8] K. Chintalapudi, A. P. Iyer, and V. N. Padmanabhan. Indoor localization without the pain. In N. H. Vaidya, S. Banerjee, and D. Katabi, editors, *MOBICOM*, pages 173–184. ACM, 2010.

[9] E. M. Daly and M. Haahr. Social network analysis for routing in disconnected delay-tolerant manets. In *Proceedings of the 8th ACM international symposium on Mobile ad hoc networking and computing*, MobiHoc '07, pages 32–40, New York, NY, USA, 2007. ACM.

[10] A. Foster and N. Ford. Serendipity and information seeking: an empirical study. *Journal of Documentation*, 59(3):321–340, May 2003.

[11] A. Garca-Crespo, J. Chamizo, I. Rivera, M. Mencke, R. Colomo-Palacios, and J. M. Gmez-Berbs. Speta: Social pervasive e-tourism advisor. *Telematics and Informatics*, 26(3):306–315, 2009.

[12] P. Hui, J. Crowcroft, and E. Yoneki. Bubble rap: social-based forwarding in delay tolerant networks. In *Proceedings of the 9th ACM international symposium on Mobile ad hoc networking and computing*, MobiHoc '08, pages 241–250, New York, NY, USA, 2008. ACM.

[13] H. Liu, H. Darabi, P. Banerjee, and J. Liu. Survey of wireless indoor positioning techniques and systems. *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on*, 37(6):1067 –1080, nov. 2007.

[14] A. Mtibaa, M. May, C. Diot, and M. Ammar. Peoplerank: social opportunistic forwarding. In *Proceedings of the 29th conference on Information communications*, INFOCOM'10, pages 111–115, Piscataway, NJ, USA, 2010. IEEE Press.

[15] A. K. Pietilinen, E. Oliver, J. LeBrun, G. Varghese, and C. Diot. Mobiclique: middleware for mobile social networking. pages 49–54, 2009.

[16] T. Spyropoulos, K. Psounis, and C. S. Raghavendra. Spray and wait: an efficient routing scheme for intermittently connected mobile networks. In *Proceedings of the 2005 ACM SIGCOMM workshop on Delay-tolerant networking*, WDTN '05, pages 252–259, New York, NY, USA, 2005. ACM.

[17] A. Vahdat and D. Becker. Epidemic routing for partially-connected ad hoc networks. Technical Report CS-2000-06, UCSD, 2000.

[18] W. Zhao, M. Ammar, and E. Zegura. A message ferrying approach for data delivery in sparse mobile ad hoc networks. In *Proceedings of the 5th ACM international symposium on Mobile ad hoc networking and computing*, MobiHoc '04, pages 187–198, New York, NY, USA, 2004. ACM.

[19] Z. Zhou and L. M. Ni. Prophet address allocation for large scale manets. Technical Report MSU-CSE-02-20, Department of Computer Science, Michigan State University, East Lansing, Michigan, July 2002.