MobUser: A new user-centric publish-subscribe service for mobile devices

(extended abstract of the MSc dissertation)

Mauro André Mendes Silva
Departamento de Engenharia Informática
Instituto Superior Técnico

Advisor: Doutor Carlos Ribeiro
Co-Advisor: Doutor João Leitão

***Abstract*—Mobile devices with wireless communication capabilities are a constant in our daily lives. This offers the possibility for such devices to exchange information for their users to assist in their daily lives. Consider for instance the exchange of information about points of interest in the locations that the user usually frequents. One should expect those devices to support such an application in a decentralized fashion, through pairwise interactions, without the interaction of the user, and avoiding to disclose information that might be sensitive to a central entity. Current solutions are not suitable for supporting such applications since most rely on some kind of centralized architecture (either brokers or rendezvous nodes), other solutions that don't rely on centralized solutions present a high overhead.**

To address this challenge, in this paper we propose MobUser, a novel topic-based publish-subscribe service specially tailored for supporting location-aware operation for mobile devices in a decentralized fashion. Extensive experimental work shows that MobUser offers not only better performance but also superior delivery rates when compared with a state-of-the-art mobile publish subscribe solution and a state-of-the-art gossip solution. We also show through a prototype implementation deployed on Android 4.0 devices, that the overhead and energy consumption of MobUser is acceptably low.

I. Introduction

Mobile devices with wireless communication capabilities are a constant in our daily lives, and users expect their devices to have available information at all times. Although access to centralized servers can be expensive, and sometimes even impossible, devices are often in the range of other devices that can potentially have useful information for their users. Considering that users are frequently on the move, the number of devices with which a device can exchange information becomes significantly high.

The main contribution of this paper was motivated by applications similar to 'Foursquare'[1]. We believe that supporting similar applications in a decentralized fashion is imperative to protect the privacy of users. We envision an application that would allow users, for instance, to exchange information concerning points of interest (coffees, museums, etc), without disclosing their habits or routines to a centralized, and potentially untrustworthy, third party.

[1]https://foursquare.com/about/new

To achieve this, we focus on the publish-subscribe abstraction. This abstraction perfectly fits our requirements since it allows decoupled communication over time and space[1]. Additionally, topic-based publish-subscribe, due to the inherent use of topics, presents the advantage of enabling devices to filter out information that simply would not fit the interests of it's user, allowing to make a more efficient use of limited memory available in these devices. In fact, the topic-based solutions are a perfect fit for our case-study application, since typically points of interest will be associated with categories as to simplify search.

The dissemination of information in a network with high mobility and with devices that are not always available presents significant challenges, such as:

- It is a complex task to ensure high levels of reliability without flooding the entire network[2]
- Mobile devices have limited energy and limited memory, which motivates solutions that are specially tailored to make an adequate use of these resources.

To the best of our knowledge, most of the existing publish-subscribe solutions have been designed for wired networks and do not take these limitations on account. Those that were designed for wireless networks are typically content-based solutions which have additional disadvantages, such as being required to rely on complex filtering techniques that consume additional power.

In this paper we propose a novel topic-based publish-subscribe solution for wireless ad-hoc networks, named MobUser, and experimentally compare it with a state-of-the-art solution [2]. Our results show that MobUser presents high reliability, which means that subscribers receive the events they have interest in with high probability, with a relatively low degree of average connectivity, and in an efficient fashion.

Our algorithm is portable and scalable and does not assume any routing infrastructure, being practical in several wireless networks (*e.g.* 802.11 or Bluetooth). Our algorithm incorporates a data management scheme to store events that limits memory usage, and also contributes to the dissemination of events belonging to less popular topics (*i.e.*, topics with a lower number of subscribers).

The rest of this paper is organized as follows: Section II discusses related work and compares several relevant approaches to our own work. Section III motivates, presents, and describes MobUser. In Section IV we evaluate and compare our work to two state-of-the-art solutions found in the literature. Section IX discusses a prototype of MobUser that was implemented and deployed on two Android devices and also discusses performance measurements that were performed in this context. Finally, Section X concludes this paper.

II. Related Work

This section discusses previously proposed solutions. We first discuss gossip dissemination solutions for disseminating information in an ad-hoc network. Then we compare our work with some publish-subscribe solutions in ad-hoc networks.

A. Gossip dissemination in Ad-Hoc Networks

Gossip, or epidemic, algorithms are a class of solutions in which periodically, a node chooses a set of random nodes to exchange information with. For the particular case of ad-hoc networks, a well known solution is the Opportunistic Gossip[3] which was introduced, (similar to our work) to leverage the mobility of nodes in these networks. Unfortunately, such solution is not aware of the users interests, therefore it spends a non-negligible amount of resources to store and disseminate information that is not useful for the user, approximating to the behavior of a flood dissemination.

Another gossip-based solution for ad-hoc networks, which presents some similarities to our work, is the Autonomous Gossip[4] which also leverages on node's mobility. However in this solution, data is seen as a living organism and the user profile (*i.e.*, it's interests) are seen as an ecosystem. In this system, data competes for space in the device's memory, data with low usefulness is discarded and not disseminated. This, however, makes this solution unable to guarantee that events on less popular topics are ever disseminated, having a negative impact on the reliability of the dissemination process.

Although pure gossip-based solutions are simple, they can suffer from the *broadcast storm problem*[5], which happens when a node receives new information and rebroadcasts this information immediately. If the neighbors do the same, then contention and collision problems arise in the network making communication among nodes impossible. Our solution does not suffer from this problem because we assume long delays between communications to save battery and, as such, it is not likely for all nodes in a region to become synchronized among them.

B. Publish-Subscribe in Ad-Hoc Networks

To the best of our knowledge, one of the first solutions to allow explicit mobility was JEDI[6]. This system had limited mobility support, essentially relying on two operations: moveOut and moveIn which enable nodes to disconnect and reconnect the network in the same or a different location in an explicit fashion, which made the publish-subscribe solution extremely complex.

Since the publication of JEDI, many systems have explored how to deal with mobility in publish-subscribe solutions for ad-hoc networks[7], [8], [9], [2], as the decoupled nature of the publish-subscribe paradigm is well suited for these networks. Unfortunately, a large number of these solutions focus on content-based publish-subscribe[7], [10], which, as discussed previously, are not well suited for environments with resource constraints (processing power, energy, memory). Other solutions focus on centralized architectures or require the assistance of brokers[7], and therefore are unable to operate in a pure decentralized fashion.

Solutions like SpiderCast[8] and Sub-2-Sub[9] attempt to achieve fully distributed topic-based publish-subscribe solutions. Unfortunately, their design require the construction of topic-based overlay networks which, present excessive overhead in, and may not cope with, scenarios with high mobility where some nodes may be in low populated areas.

The solution which is more similar to ours in both functioning and assumptions is Mobility Friendly Publish-Subscribe (MFPS)[2]. Similar to our solution, MFPS assumes that nodes are mobile and they make no assumptions regarding the connection graph of the processes, in contrast with most existing solutions. Additionally, MFPS also relies on an one-hop communication model. MFPS, however, has some drawbacks. In the presence of lower subscription percentages to some topics it may happen that events are never disseminated. Also, MFPS presents an extra round of communication which, as we will demonstrate in our experimental results, lead to lower reliability in situations of low population density. Results comparing the performance of our solution with MFPS experimentally are presented in Section IV.

III. MobUser

In this section, we describe MobUser. We begin by presenting the assumed model, followed by a global overview of the protocol, which introduces the main components of our solution, that we describe in detail afterwards.

A. System Model

We aim at designing a solution for an urban style mobility pattern. Devices are expected to be mobile and have the ability to communicate directly among themselves when they are in each other's range. We assume that exchanged data between devices is partially stored by each device (and that there is a memory limit constraint on devices).

The network is assumed as being ad hoc, without any infrastructure device present. The number of devices with which each device can communicate is constantly changing over time and is not restricted to a maximum number.

The devices have limited energy, memory, and processing power. Devices communicate through the exchange of messages. It is also assumed that devices have some form of extracting GPS coordinates (*e.g.* Google Maps) to simplify the task of associating a location to events and also to define

the frequent locations (although the task of defining frequent locations could be done automatically if the device had a GPS receiver or other form of location discovery).

It is also assumed that the events are generated (*i.e.* published) by the users and that events generated by the users are not discarded from their own device(s). The motivation behind the generation of the events from the user can be either to save that information to latter access it or to explicitly share it with other users through a specially tailored application.

B. Global Overview

Our solution uses the neighbor discovery system made available by the underlying communication technology (*e.g.* 802.11 or Bluetooth). If such service is not available, a broadcast service can be used to perform this task.

We rely on a recurring task that is executed with a configurable interval. This task performs a neighbor discovery procedure which searches for devices executing the MobUser protocol. Each device has a user profile that consists of the set of topics subscribed by the user. The topics are defined at the moment of deployment and are globally known. When a new neighbor is found, both exchange *generated profiles*, which are generated using the profile defined by the user and knowledge of the least demanded topics to protect the privacy of the user.

After the exchange of profiles, devices exchange every event that has a topic subscribed by it's neighbor and that are locally available. The events are characterized by a name, a topic, a timestamp, a location and a reputation value. Events are stored in devices in three data structures: a list of recent events, a list of popular events, and a list of irrelevant events. These lists are kept sorted using different criteria. The recent events list is meant to store newer events, so events are ordered from the most recent to the least recent (*i.e.*, considering the first time the event was received). The popular events list is ordered from the event with highest reputation to the lowest one. The irrelevant events list serves to help disseminate events belonging to topics that have low number of subscriptions. To this end, events are kept in the list until they have been disseminated at least once. The information of how many times an event from the irrelevant events list has been disseminated is encoded into the reputation of that event and therefore, the list is sorted from the event with lowest to the one with highest reputation.

In order to make our solution location-aware, we handle the locations associated with events as a topic, as follows: locations are defined as a cell in a multilevel grid which at the lower level (level 0) are of approximately 1 square meter. GPS coordinates are transformed into level 0 cells by truncating each coordinate to a 0.00001 precision and then by removing the dot (*e.g.*, a coordinate 38.736762,-9.139343 becomes cell 3873676,-913934). Each higher level is comprised of a 10 by 10 area of the lower level (*e.g.* a level 1 cell 0,0 is comprised of level 0 cells from 0,0 through 10,10.).

Notice that, events always have 2 associated topics, a topic which is the location associated with the event and at least one topic which corresponds to topics associated with the semantics of the application.

C. Event storage

When received events are stored they are verified to check if they belong to a topic that was in the generated profile[2]. If this is not true, they are discarded. If they are associated to a topic in the generated profile, but not the user profile, then they are stored in the irrelevant events list. If events belong to the user profile both lists of popular and recent events are searched and if the event exists then it's reputation is increased. If the event was in the recent events list it is transferred to the popular events list. If the event was not stored in any of these two lists, it is added to the recent events list. Whenever a list reaches it's maximum capacity the event that was added to that list at most time is discarded win adding a new event.

D. Dissemination

The dissemination phase starts by performing a neighbor discovery. For each neighbor found, a temporary profile is generated (generated profile) which is comprised of the user profile enriched with the least demanded topics. The demand of a topic is a subjective vision that each device gathers from counters associated to the topics present in the profiles received in past interactions. The generated profile is then shared with the neighbor. Adding the least demanded topics to the generated profile has two effects. First, we obfuscate the real user profile, making more difficult the task of associating an user profile to a device. It also promotes events from less subscribed topics to flow faster among devices. Also, since the identifiers used in this system are only used for communication purposes, they can be randomized between interactions. This feature is implemented in our prototype, where we change the MAC address used in each interaction to further harden the task of associating an user profile to a particular device.

When a device receives a profile, it sends it's own (generated) profile as a response. It also searches the three previously mentioned lists for events that have a topic in common with the received profile. If events, with a topic in common with the received profile are found, they are sent to the neighbor from which the profile was received. These events can be sent in two ways. As a whole, in which every event is sent in the same packet, or in an interactive way, in which each event is only sent after receiving an event from the neighbor. The advantage of sending every event at the same time is having lower communication costs. The advantage of sending events interactively is mitigating selfish behavior, in which users try to have their devices to only receive information and never share any content.

Each event sent from the list of irrelevant events has it's reputation incremented every time it is disseminated. After

[2]Details of the generation of the profile are explained further ahead.

the events have been disseminated an event storage phase begins, which operation was described above.

IV. Evaluation

In this section we evaluate our solution. We begin by describing the experimental testbed, then we discuss the configuration used in each test and finally, we present the results of our experimental measurements.

A. Environment

To evaluate our solution we simulated our algorithm using the Network Simulator (v2.35) (also known as NS2)[3]. To simulate the movement we imported files generated from BonnMotion (v2.0)[11] and used the Manhattan Grid mobility model. The version of MFPS with which we compare our solution was implemented in NS2 according to [2].

B. Common settings

We simulated our protocol as an Agent directly on top of the 802.11b MAC layer with a maximum range of 10m, this range was selected since our protocol is supposed to be supported by any device and most common wireless technologies today are Bluetooth and 802.11.

The propagation was simulated using the Two-Ray Ground propagation model. This was selected because the solution with which we compare the performance of our system, heavily relies on this model. The values used for RXThresh_ to define the range of the antenna were calculated using the tool *threshold*, available in the NS2 suite.

To generate the movement patterns we use the previously mentioned model with the following properties:

- *Skipped seconds from the beginning of the simulation:* 3600. This value was used since it is BonnMotion's default value.
- *Width and Height:* 450m, except for the density tests.
- *Number of nodes:* 1000.
- *Speed change probability:* 20%.
- *Minimum speed:* 0.5m/s and *Mean speed:* 1.5m/s. This values were chosen because according to [12].
- *Pause probability:* 0%.
- *Turn probability:* 50%.
- *Number of blocks per dimension:* This value was chosen for each test to create blocks of 20m thus not allowing nodes on different streets to communicate with each other.

The goal of this configuration was to model a realistic scenario while allowing simulations to be executed in an efficient fashion.

Two measures were the focus of our tests, the reliability and the amount of transmitted data. We define reliability as the percentage of published events of a topic that subscribers of that same topic effectively receive. Presented results are an aggregation of 10 individual experiments using different movement patterns.

[3]http://www.isi.edu/nsnam/ns/

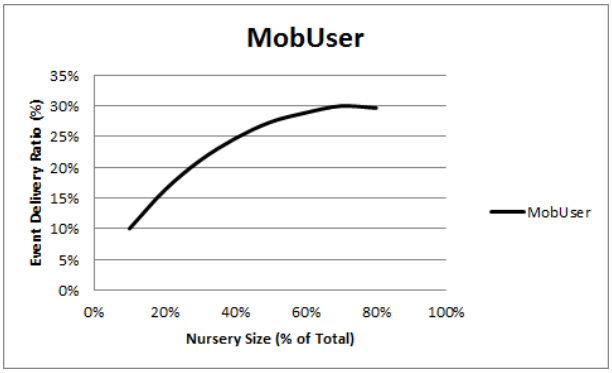


Figure 1. Reliability across different recent events list sizes

In all tests the time interval for the periodic neighbor discovery procedure was configured to 300 seconds. For Mobility Friendly Publish-Subscribe (MFPS)[2] that interval varies between 200 to 400 seconds being that, as MFPS uses the velocity of the nodes around it to calculate it's interval, it's mean value is of 300 seconds.

We start by presenting the experimental result that allowed us to choose the best configuration for our system. We follow with results for experiments that analyze the behavior of the systems in three distinct dimensions: Time, Subscription Probability and Population Density.

1) System Configuration: Another configuration that had to be set was the size of each list of the MobUser, and also a maximum size for the memory for all solutions. We have set a maximum memory of 20 events for all solutions, and devised an experiment to find the best value for the size of each list in the MobUser service. We had already established that the irrelevant list would only occupy 10% of the maximum memory space in order to maintain a very low overhead to the system. With that in mind, we only had to find the best compromise between the recent and popular events lists. To find the value we simulated a squared area with 100 meters in each side and with 100 nodes. We then increased the size of the recent events list, while consequently decreasing the popular events list. The results can be found in Figure 1. In the Figure the nursery size corresponds to the size, in terms of percentage, of the recent events list and the delivery ratio is the reliability with which the simulation ended (simulations were executed during 1 hour of simulation time). We can see that a bigger recent events list translates into a higher reliability until the size reaches 70% of the maximum memory. Sizes above this lead to a decrease in the reliability of the system. For this reason all simulations from this point forward were performed with a recent events list occupying 70% of the maximum memory, the popular events list occupies 20% and the irrelevant events list stays, as defined, with 10%.

V. Performance over time

We began by evaluating the reliability evolution over time. To achieve this goal, we simulated a squared area of

450 meters per side and established that all nodes would subscribe to the same topic. Of the subscribing nodes about 10% of those nodes would publish an event. The simulator was set to simulate 24 hours of execution.

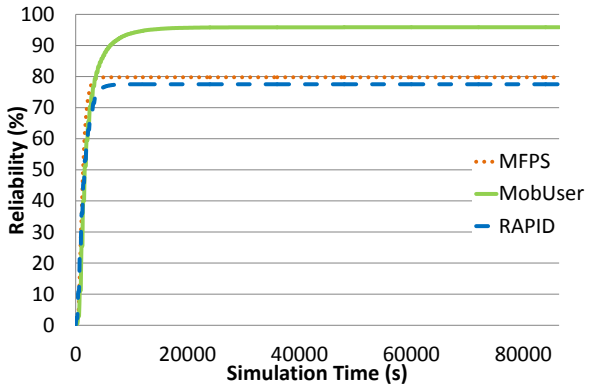


Figure 2. Reliability over time in a 24h period

Figure 2 presents the results of the reliability of the three solutions over time. In it we can see that most of the dissemination is done in the first hour or so, the increase in terms of reliability is very slim past that first hour. We can also note that both MFPS and RAPID have very similar reliabilities while MobUser presents a much higher reliability.

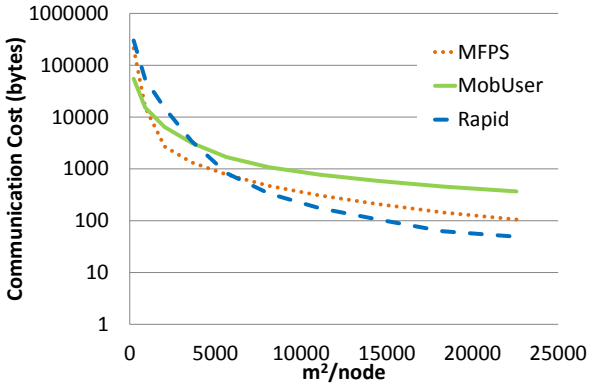
Due to the fact that the increase of reliability is very slim past the first hour, we have chosen to perform simulations of 4 hours for the remainder of the evaluation (with exception of the simulation performed with multiple interest areas) to have faster simulations but yet maintaining some margin for simulations in which the dissemination is slower.

VI. Performance with different densities

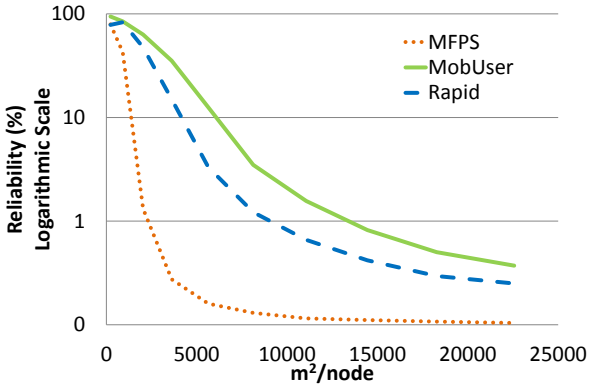
Evaluating the performance of the solutions for variable population densities is also important since mobile nodes can exist in several types of environment with different population densities. The goal is for our system to operate correctly in any population density.

To evaluate the performance in several densities we maintained the 1000 nodes in every simulation, but increased the squared simulation area's side from 450m to 4500m, with steps of 450m. That means that we simulate from a density in which a neighbor can be found approximately every 8 meters, to a density in which a nodes finds a neighbor at approximately 80 meters from its current location. In this simulation all nodes also subscribe to the same topic, and about 10% publish an event. The presented results are for simulations covering an interval of 4 hours.

Figure 3(b) clearly shows that MobUser is more robust to density changes than the remaining simulated solutions. The Figure is presented with a logarithmic reliability scale to be



(a) Communication cost

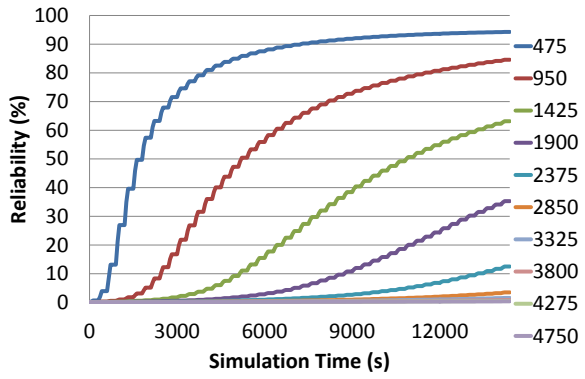


(b) Reliability

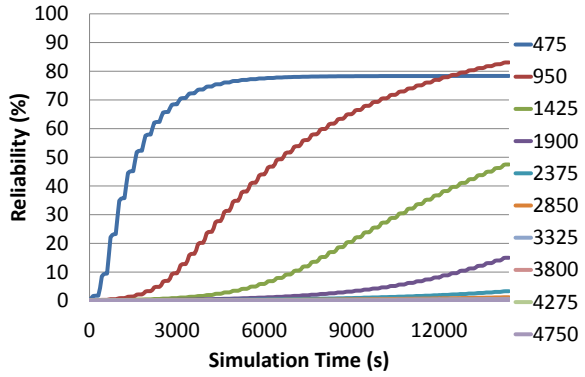
Figure 3. Behavior of the systems across several densities

easier to verify the differences between each tested solution, since all of them present a very large decrease in reliability with lower densities. This, however, is normal since lower densities translate to less opportunities to exchange data through pairwise interaction. But even with this decreasing tendency we can note that MobUser always presents higher reliability when compared with the remaining solutions. Examining Figure 3(a) one can argue that MobUser has higher communication cost than the other solutions at lower densities, but if we analyze the reliability it is also higher which explains the higher need for communication among devices, since each node has more data available to exchange with its peers.

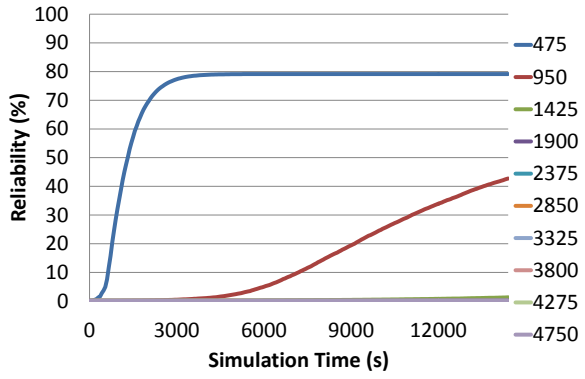
Another important aspect to retain is that if we analyze the evolution of reliability over time in the different densities, which can be seen in Figure 4, we can note that MobUser is much faster than MFPS, and is also faster than RAPID especially at lower densities. Which, once again, explains



(a) MobUser



(b) RAPID



(c) MFPS

Figure 4. Evolution of reliability over time across several densities

the higher communication cost of MobUser, since the nodes have information available sooner and for that reason have more information to share with other devices. It is also worth noting that RAPID has a lower communication cost because it does not exchange the profile of the user (because it is a protocol that is not aware of user interests) and for that reason, on lower densities, the weight of that profile exchange in the overall communication is much larger than on higher densities.

VII. Performance with different subscription probabilities

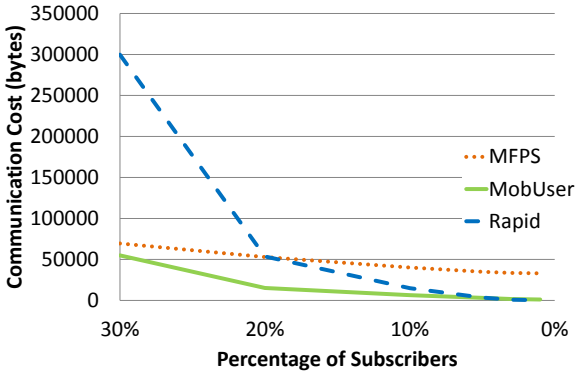
After evaluating the performance of MobUser in idealistic (potentially more academic) environments, where only a single topic existed and every node subscribed to it, we moved to evaluate the performance of the systems in the scenario for which our solution was created. This means a scenario where topics with low subscription co-exist with topics with high subscription.

To evaluate this feature we chose a squared simulation area with 450 meters in each side. The simulation was executed to cover 4 hours of execution. In this simulation co-exist 6 different topics. Each topic has a probability associated with it. This probability encodes the probability with which an individual node subscribes to that particular topic. Therefore, topics with a low probability value will become less popular when compared with topics with higher probability values associated. If a node did not subscribe a particular topic, then it subscribes to a default (popular) topic. Every node that subscribed to the topic associated with the lowest subscription probability publishes a single event for that topic.

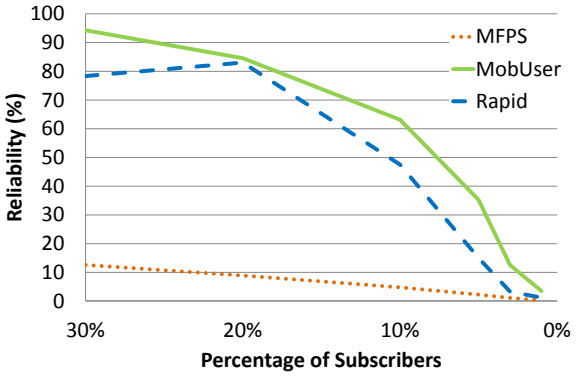
Figure 5 summarizes results obtained for different probabilities associated with the less popular topic and clearly demonstrates that MobUser achieves its goal. MobUser achieves the best of both worlds, maintaining a low communication cost on environments with a large amount of information, but yet maintaining a reliability that is higher than even a protocol that is blind to topics and, as such, uses the entire population's memory to disseminate topics, wasting a very large amount of energy exchanging events that are irrelevant to the user. It is important to note that with a subscription probability of 30% MobUser achieves higher reliability than RAPID with 6 times less communication cost.

It is important to remind the reader that MobUser only uses 10% of the device memory allocated to store events from MobUser to store events associated with topics that were not subscribed by the user, and still manages to achieve reliability values that are higher than a system of pure gossip dissemination (approximating a flood protocol), that uses the entire allocated memory of each device to disseminate any topic.

MFPS in this situation, since (like most proposed publish-subscribe systems) it does not implement a mechanism to help the dissemination of low subscribed topics, has a very low reliability with even higher communication cost than MobUser. This scenario perfectly reflects the need for

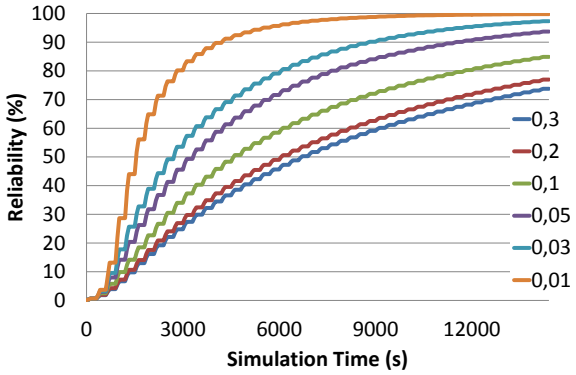


(a) Communication cost

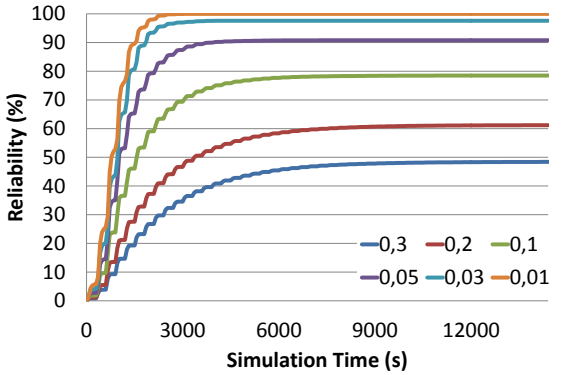


(b) Reliability

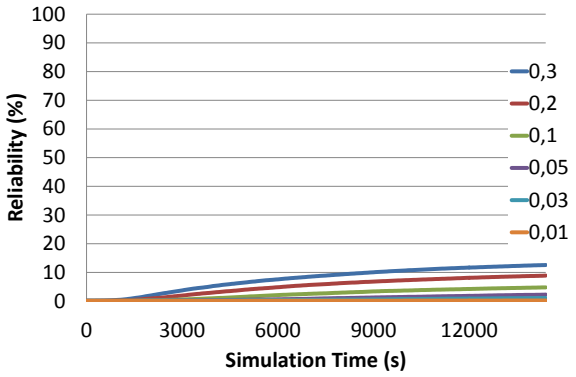
Figure 5. Behavior of the systems across several subscriber probabilities



(a) MobUser



(b) RAPID



(c) MFPS

Figure 6. Evolution of reliability over time across several subscriber probabilities

mechanisms to help the dissemination of topics that have less subscribers.

Figure 6 shows that RAPID as a lower latency in the delivery of messages on lower subscription probabilities, delivering every event in about 2000 seconds, while MobUser only delivers every event in about 9000 seconds. However, in higher subscription probabilities MobUser is not only more reliable but faster than RAPID. The reason of RAPID's low latency is the fact that every node stores all the events in their cache (until it is full), while in MobUser we bound the consumed storage on topics that are not locally subscribed.

VIII. Performance on a scenario with multiple areas of interest

Finally, we devised a test that demonstrates the performance of our solution in a realistic environment considering the initial motivation for the design of MobUser. Figure 7 represents the setting of this test. We consider six nodes, A to F. Squares labeled in the figure represent their frequent

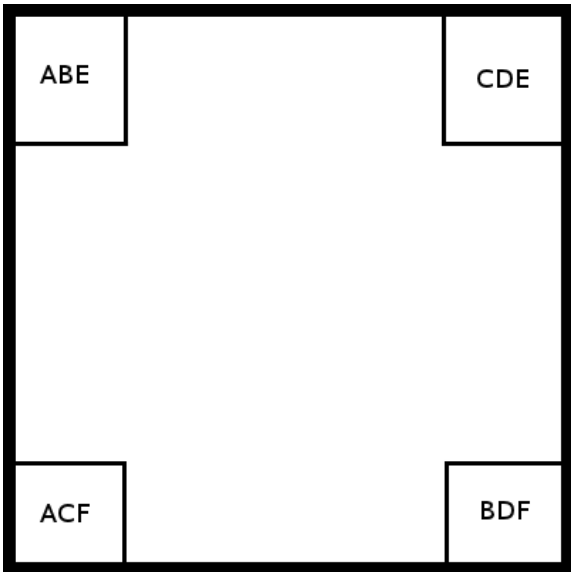


Figure 7. Test setting

locations (i.e. the locations to which they subscribe). Each node subscribes to these two areas and publishes an event for that area in a given topic. Besides these nodes we consider four additional nodes, that move in a Random Waypoint fashion over the entire simulation area. Nodes A to F present a movement pattern in which they move in a Random Waypoint fashion over the simulation area for some time. After this they move inside one of their frequent areas (chosen at random) for an equal amount of time.

To create the movement patterns for this test we used BonnMotion and extended it with our version of the Random Waypoint model. As previous tests we performed 10 independent executions of each experience and averaged the results. The square presented in Figure 7 has 120 meters in each side, and the simulation was performed for 1 hour.

In this test both MobUser and RAPID achieved an average reliability of 100%, while MFPS achieved a lower reliability of 85%. Note that in this test only a small number of nodes is considered, which resulted in the number of published events not being enough to fill the memory of the devices. If there were enough events to fill the available memory of nodes it would be expectable that both MFPS's and RAPID's reliability would be significantly hampered as shown by the experimental results presented earlier in this chapter. By contrast, MobUser is expected to maintain higher reliability compared to MFPS since it also relies on devices that did not subscribe to the information to help disseminate events between nodes (as discussed previously), and is also expected to maintain higher reliability compared with RAPID since, as previously noted, in environments with higher number of events RAPID quickly fills the cache of the devices, while MobUser uses the storage in a way that maximizes usefulness of the information.

Although this test is small scale we can clearly verify the difference between the pure publish-subscribe solution and MobUser, since our solution achieved 100% of reliability while MFPS, even in small scale, was not able to achieve such values.

This scenario is important to demonstrate the importance of solutions that, like ours, do not rely solely on exchanging information with devices that share interests. Consider that in the scenario portrayed by Figure 7 that node E has an event that is very important to D. In pure publish-subscribe solutions D would either have to communicate directly with E, or it would have to communicate with C after it has communicated with E. Those are the only two paths that the information can take to reach D. In our solution the information can flow through many different paths, such as E could communicate with C, C with F and F would relay the event to D. Or every other path that goes from E to D, since every node can, potentially, have an event from any other topic (we would like to remember the reader that locations of interest are also considered topics in our solution).

IX. Case Study

In this section we present a case study application developed as proof of concept of our work. The application was developed for Android version 4 and tested using two devices. Since Android devices present some constrains to the usage of wireless technology we had to create some workarounds to allow us to develop the application in such a restrictive setting.

The first constrain found was the fact that android devices do not allow applications to put Bluetooth in Discovery Mode without the interaction of the user, and even with interaction of the user the Discovery Mode can only be made active for a maximum period of time (even though the Android documentation refers to the maximum period as 3600 seconds we have found that in all tested devices this was in reality 300 seconds). Since it would not be convenient for the user to be forced to interact with it's device every 300 seconds we have built a workaround. We ask for user interaction as soon as the application starts and through Java's reflection we call the setDiscoverableTimeout method and set the timeout to -1, in fact, disabling the timeout and maintaining the device discoverable as long as the Bluetooth device is enabled.

Since our application was meant to support both Bluetooth and 802.11, we decided to use the Wi-Fi Direct API of the Android systems. This was, however, made difficult by the fact that Android does not allow applications to enable Wi-Fi Direct without the interaction of the user. Once again we have followed the Android code and found that through reflection we could call the ENABLEP2P method of the Wi-Fi Direct service and enable it without interaction of the user.

With these two workarounds we could ask the user to enable Bluetooth discoverability only once during the lifetime of the application and we did not need any user interaction for using the Wi-Fi communication, and thus the

application would be viable to be deployed for the regular user.

To evaluate our prototype two simple tests were performed. The first evaluated the battery consumption of our solution on the previously mentioned mobile devices. In this test we fully charged the devices batteries and then recorded the battery health at 1 hour intervals for 3 hours, we then fully charged the devices once more and activated our application and again recorded the battery health at 1 hour intervals for 3 hours.

The results have shown that the difference in battery consumption between the average device consumption and the consumption with our service exchanging events every five minutes was of 1.5%. We consider that such small overhead is acceptably low to allow users to continually execute our service without significantly impairing the lifetime of the device's battery. Difference of battery consumption between the 2 solutions was not significant.

The second test was executed to evaluate the amount of time required by devices to exchange events between them. Note that considering a highly mobile environment, devices should only have (on average) tens of seconds to perform an exchange between them. For this experience two devices were put in range of each other and the application exchanged a single event from each device and recorded the elapsed time between the moment at which the second device was found to the moment when event storage terminates. We executed this 10 times.

In the experiments performed, the exchange operation had a duration between 0.5 and 2 seconds. This is an acceptable delay since the time window in which two devices are in range is expected to be larger (a few tens of seconds).

The reader should note that for the aforementioned tests we employed a partial wake lock over the Android system to enable the devices to exchange messages even when the device entered sleep mode. Our deployment has shown that MobUser can be implemented in current systems and that it can efficiently exchange information between devices in real world applications. We believe this prototype demonstrates the applicability of MobUser to support decentralized applications in currently available, and common, devices.

X. Conclusion

This paper has presented an event dissemination solution for long lasting events that follows a topic-based publish-subscribe interaction style. This algorithm was designed for ad-hoc networks and relies only on pure pair-wise communication being therefore, completely decentralized and inherently scalable. Our solution not only supports mobility but embraces it enabling events to be disseminated to nodes located in remote areas in relation to the local where the event was originally produced.

We have also contributed with an event storage mechanism that limits the memory usage and also enables events from topics with low subscription rates to be eventually disseminated. According to our results, our solution has better performance and consumes less device resources when compared with a state-of-the-art publish-subscribe solution. We have also demonstrated that our solution is able to have the advantage of filtering most irrelevant information yet maintain reliability values that match gossip dissemination protocols. We have also demonstrated that, using our system, with a high probability, any event published will reach all the interested devices.

We have also presented a prototype that clearly shows that MobUser can be deployed on currently available devices and that it can serve as a support for decentralized applications, namely, social-inspired applications that are user-centric and avoid the use of centralized components, such as the application that we described in the paper which serves as a motivation to this work.

Acknowledgments

This work was partially supported by FCT - Fundação para a Ciência e Tecnologia under the project PEstOE/EEI/LA0021/2011.

References

[1] P. T. Eugster, P. A. Felber, R. Guerraoui, and A.-M. Kermarrec, "The many faces of publish/subscribe," *ACM Comput. Surv.*, vol. 35, no. 2, pp. 114–131, Jun. 2003.

[2] S. Baehni, C. S. Chhabra, and R. Guerraoui, "Mobility Friendly Publish/Subscribe," Tech. Rep., 2004.

[3] R. Friedman, D. Gavidia, L. Rodrigues, A. C. Viana, and S. Voulgaris, "Gossiping on manets: the beauty and the beast," *SIGOPS Oper. Syst. Rev.*, vol. 41, no. 5, pp. 67–74, Oct. 2007.

[4] A. Datta, S. Quarteroni, and K. Aberer, "Autonomous gossiping: A self-organizing epidemic algorithm for selective information dissemination in wireless mobile ad-hoc networks," in *Semantics of a Networked World*, ser. Lecture Notes in Computer Science, M. Bouzeghoub, C. Goble, V. Kashyap, and S. Spaccapietra, Eds. Springer Berlin / Heidelberg, 2004, vol. 3226, pp. 126–143, 10.1007/978-3-540-30145-5_8.

[5] Y.-C. Tseng, S.-Y. Ni, Y.-S. Chen, and J.-P. Sheu, "The broadcast storm problem in a mobile ad hoc network," *Wireless Networks*, vol. 8, pp. 153–167, 2002, 10.1023/A:1013763825347.

[6] G. Cugola, E. Di Nitto, and A. Fuggetta, "The jedi event-based infrastructure and its application to the development of the opss wfms," *Software Engineering, IEEE Transactions on*, vol. 27, no. 9, pp. 827 –850, sep 2001.

[7] R. Baldoni, R. Beraldi, G. Cugola, M. Migliavacca, and L. Querzoni, "Structure-less content-based routing in mobile ad hoc networks," in *Pervasive Services, 2005. ICPS '05. Proceedings. International Conference on*, july 2005, pp. 37 – 46.

[8] G. Chockler, R. Melamed, Y. Tock, and R. Vitenberg, "Spidercast: a scalable interest-aware overlay for topic-based pub/sub communication," in *Proceedings of the 2007 inaugural international conference on Distributed event-based systems*, ser. DEBS '07. New York, NY, USA: ACM, 2007, pp. 14–25.

[9] S. Voulgaris, E. Rivière, A.-M. Kermarrec, and M. Van Steen, "Sub-2-Sub: Self-Organizing Content-Based Publish and Subscribe for Dynamic and Large Scale Collaborative Networks," INRIA, Rapport de recherche RR-5772, 2005.

[10] L. Fiege, F. C. Gärtner, O. Kasten, and A. Zeidler, "Supporting mobility in content-based publish/subscribe middleware," in *Proceedings of the ACM/IFIP/USENIX 2003 International Conference on Middleware*, ser. Middleware '03. New York, NY, USA: Springer-Verlag New York, Inc., 2003, pp. 103–122.

[11] N. Aschenbruck, R. Ernst, E. Gerhards-Padilla, and M. Schwamborn, "Bonnmotion: a mobility scenario generation and analysis tool," in *Proceedings of the 3rd International ICST Conference on Simulation Tools and Techniques*, ser. SIMUTools '10. ICST, Brussels, Belgium, Belgium: ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2010, pp. 51:1–51:10.

[12] M. S. Tarawneh, "Evaluation of pedestrian speed in jordan with investigation of some contributing factors," *Journal of Safety Research*, vol. 32, no. 2, pp. 229 – 236, 2001.