SPRINT-SELF: Social-Based Routing and Selfish Node Detection in Opportunistic Networks

Radu Ioan Ciobanu[1], Ciprian Dobre*[1], Valentin Cristea[1], Florin Pop[1] and Fatos Xhafa[2]

[1]Faculty of Automatic Control and Computers, University Politehnica of Bucharest, Romania
[2]Universitat Politecnica de Catalunya, Barcelona, Spain

October 3, 2013

Abstract

Since mobile devices nowadays have become ubiquitous, several types of networks formed over such devices have been proposed. One such approach is represented by opportunistic networking, which is based on a store-carry-and-forward paradigm, where nodes store data and carry it until they reach a suitable node for forwarding. The problem in such networks is how to decide what the next hop will be, since nodes do not have a global view of the network. We propose using the social network information of a node when performing routing, since a node is more likely to encounters members of its own social community than other nodes. In addition, we approximate a node's contacts as a Poisson distribution and show that we can predict its future behavior based on the contact history. Furthermore, since opportunistic network nodes may be selfish, we improve our solution by adding a selfish node detection and avoidance mechanism, which can help reduce the number of unnecessary messages sent in the network, and thus avoid congestion and decrease battery consumption. We show that our algorithm outperforms existing solutions such as BUBBLE Rap and Epidemic in terms of delivery cost and hit rate, as well as the rate of congestion introduced in the network, by testing in various realistic scenarios.

Keywords: opportunistic; social; prediction; selfishness

*Corresponding author, E-mail: ciprian.dobre@cs.pub.ro

1 Introduction

The ubiquitousness of mobile devices has led to the advent of opportunistic networks (ONs), which consist mainly of human-carried mobile devices (e.g. smartphones, tablets, etc.) that are unaware of any network infrastructure and interact with each other based on a store-carry-and-forward paradigm. Nodes communicate opportunistically when they are within wireless range of each other. A node stores data in the form of messages, which are carried around until a node with a higher chance of delivering them to the destination is met. Then, the messages are forwarded to the encountered node. Routes between nodes are dynamically created, and nodes can be opportunistically used as a next hop for bringing each message closer to the destination.

The nodes of an opportunistic network are mobile devices usually carried by people, which are organized into communities according to common professions, workplaces, interests, etc. Generally, members of the same community interact with each other more often than with members of outside communities, so a good opportunistic network routing algorithm should take community organization into consideration. We show here that adding knowledge about social links between ON nodes to routing and dissemination algorithms greatly improves their effect. We strongly believe that social network connections are a better approximation of human relationships than existing community detection algorithms. However, we show that only using social information about the nodes in the network is not enough to obtain satisfactory results, therefore we attempt to predict the future behavior of a node by using the history of encounters and information about its social community. We approximate a node's contact history as a Poisson distribution and use the result in creating a routing algorithm entitled SPRINT.

However, some nodes in the opportunistic network may not be willing to participate in the routing process at all times. Thus, a node may be selfish towards another node, for various reasons (e.g. it might be low on resources such as battery life, memory, CPU, network or bandwidth, or it might lack interest in helping nodes outside its own social community). The existence of such selfish nodes in an ON might lead to messages having high delays or never being delivered at all, so these nodes must be acknowledged and avoided when possible. Therefore, we propose a gossip-based improvement to the SPRINT algorithm, entitled SPRINT-SELF, which is able to detect selfish nodes and avoid forwarding data to them. The results are compared to existing ON routing solutions in various realistic scenarios, and we show that our solution performs better when selfish nodes are populating the network.

Preliminary versions of our work were previously published in [9] and [11]. This article proposes an improvement to the SPRINT algorithm that detects and avoids selfish nodes, and also adds more extensive work, through a new set of experiments in realistic scenarios where nodes have a limited contact duration when they are capable of exchanging data, as well as decreasing battery life.

The rest of the article is structured as follows. Section 2 presents related work in the field of routing, dissemination and selfish node detection in opportunistic networks. In Sect. 3 we describe SPRINT and propose an addition that allows it to detect and avoid selfish nodes. Section 4 describes the experiments performed and shows the results obtained when comparing SPRINT to BUBBLE Rap and Epidemic. Finally, Section 5 concludes the article.

2 Related Work

2.1 Data Routing and Forwarding

Since opportunistic networks have become more and more popular over the past years, partly due to the ubiquitousness of mobile devices, several authors have treated this research area in great detail. A review of opportunistic networking can be found in [12], where functions such as message forwarding, security, data dissemination and mobility models are analyzed. Several opportunistic forwarding algorithms are also reviewed, among them being BUBBLE Rap [14], PROPICMAN [22] or HIBOp [5]. We propose a taxonomy for data dissemination algorithms in [7], where we split such algorithms into four main categories. The first category refers to the infrastructure of the network, i.e. the way the network is organized into an overlay. The dissemination techniques are also split according to their node properties, such as state or interaction. The third category of the taxonomy is represented by content characteristics, i.e. the way content is organized and analyzed, and finally the last category is social awareness.

Several other authors propose dissemination algorithms for opportunistic networking. For example, the Epidemic routing protocol [25] is an algorithm where two encountering nodes exchange all their messages between each other. This way, barring data transfer restrictions, the maximum hit rate of a network can be obtained. The Socio-Aware Overlay algorithm [26] creates an overlay for an opportunistic network with publish/subscribe communication, composed of nodes with high values of centrality. Another dissemination algorithm, Wireless Ad Hoc Podcasting [21], has the purpose of wireless ad hoc delivery of content among mobile nodes and enables the distribution of content using opportunistic contacts whenever podcasting devices are in wireless range. ContentPlace [6] facilitates data dissemination in resource-constrained opportunistic networks by making content available in regions where interested users are present. In order to optimize content availability, it exploits information about users' social relationships to decide where to place user data. Nodes from ContentPlace use a utility function in order for each node to associate a value to any data object. When a node encounters a peer, it computes the utility values of all the data objects stored in the local and in the peer's cache and then it selects the set of data objects that maximizes the local utility of its cache.

The addition of social network information to opportunistic routing has been studied in [3], where the authors show that using Facebook information instead of community detection algorithms decreases the delivery cost and produces comparable delivery ratio. In [19], an analytical model for the expected hop count and latency of messages delivered in a socially-aware opportunistic routing algorithm is proposed, where the forwarding process is modelled as a semi-Markov process. A socially-aware middleware that learns information about the nodes in the network and then uses it to predict their future movement is proposed in [4]. The middleware was integrated with the Haggle architecture and was used for content sharing, yielding up to 200% improvement in terms of hit rate and 99% reduction in resource consumption in terms of traffic in the network.

The problem of predicting the future behavior of nodes in delay-tolerant networks (DTNs) is also treated in several papers. In [18], a framework for evaluating routing algorithms for DTNs is proposed and the performance of several such algorithms is analyzed in terms of the amount of knowledge about the network that they require. In [24], the authors analyze the predictability of human behavior and mobility on user traces obtained from mobile carriers. In [17], the behavior of a time series is modelled as a Poisson process model and then is modulated using a hidden Markov process. The authors show that using a Poisson model is significantly more accurate at detecting future behavior and known events than a traditional threshold-based technique. Since contact information in an opportunistic network is also a time series, we believe that it can also be approximated as a Poisson distribution, which we prove in [8].

2.2 Selfish Node Detection

Although it has been shown that opportunistic networks are robust towards altruism distribution [15], detecting and avoiding selfish nodes has the potential of lowering the unnecessary loss of resources or the delays that may appear. Therefore, several methods for the detection of selfish nodes in DTNs have been proposed in the past.

The selfish node detection mechanism for Mobile Ad Hoc Networks and DTNs described in [13] uses a collaborative watchdog approach to detect selfish nodes and spread this information in the network. In such an approach, if one node has previously detected a selfish node, it transmits this information to encountered nodes. However, this method has the main disadvantage that it assumes that a node can either be fully altruistic or fully selfish. Therefore, the perceived state of a node can fluctuate heavily if contradictory information comes from different sources.

Our approach (described in Sect. 3.2) uses fuzzy values for a node's altruism and computes perceived altruism values based on both context (social knowledge, battery level) and content (computations are performed per message). Our approach is somewhat similar to [20], where gossiping is used by nodes to spread their interpretation of the monitoring level in order to have a faster detection of selfish nodes in the network. Another proposed method [27] splits selfish nodes into free riders, black holes and novas, and uses message path analysis to separate them from other nodes.

However, simply detecting selfish nodes may not be enough to improve the performance of a network. An incentive mechanism may, for example, not accept messages from nodes considered selfish, thus forcing them to participate if they want their messages delivered. Such a mechanism is IRONMAN [2], which uses pre-existing social network information to detect and punish selfish nodes, incentivising them to participate in the network. Each node stores a perceived altruism (or trust) value for other nodes, that is initialized based on the social network layout: if the nodes are socially connected, this value is higher than for regular nodes. When a node A meets a node B, it checks its encounter history to see if B has ever created a message for A that has been relayed to another node C. If this is the case, and A has encountered C after B had given it the message but A didn't receive the message, then C is considered selfish, and A's trust in C is decreased. Whenever a node A receives a message from a node B which is not the source of the message, A's trust in B is increased. Apart from detecting selfish nodes, IRONMAN also uses incentives to make nodes behave better. Therefore, whenever a node B is considered selfish by A (its trust score is below a given threshold), it is notified, and A won't send it any messages. Moreover, it won't accept any messages from B either, so a selfish node might end up not being able to send its messages, unless it becomes altruistic.

3 Routing and Selfish Node Detection in ONs

This section presents SPRINT, a socially-aware and prediction-based routing algorithm for opportunistic networks, and shows how its performance can be improved by adding selfish node detection and avoidance techniques.

3.1 Routing in Opportunistic Networks

The SPRINT (Social PRedIction-based routing in opportunistic NeTworks) algorithm is based on two important assumptions. First of all, knowing that most of the nodes in an opportunistic network are devices carried by people, we proved in [10] that a node is more likely to interact with its own social community that with unrelated nodes. As a consequence, the most popular nodes in terms of social relationships have more contacts than the other ON participants. We also showed that adding existing social information (such as Facebook-provided data) to existing ON routing algorithms, instead of detecting social communities on-the-fly using algorithms such as k-CLIQUE [16], also leads to an increase in performance. The SPRINT algorithm takes advantage of this social community knowledge when making routing decisions.

The second main aspect of SPRINT is represented by node prediction. We proposed in [8] a way to predict a node's behavior by analyzing its history in terms of a node's past encounters and approximating the time series obtained as a Poisson distribution. We analyzed the predictability of ON nodes' mobility considering contact distribution over time under several real-world mobility traces, which contain data collected in environments where we expected to see evidence that users tend to follow certain patterns. We showed that, by using contact history, we are able to predict how many contacts would a node have in a given one-hour interval, by using a Poisson distribution.

Based on these two assumptions, SPRINT combines socially-aware routing with future node behavior prediction. Its main goal is to achieve a better hit rate than existing algorithms do, while reducing bandwidth consumption and network congestion as well.

SPRINT nodes have a data memory and a cache memory. The actual messages are stored in the data memory, while the cache memory is used to store contact history information. When two nodes are within wireless range of each other, they exchange information about the messages they carry. This information includes a hash of the message's content (which acts as a unique ID), the source and destination together with the communities they belong to, the generation time of the messages, and the number of hops it has traversed so far. Based on this information, each node computes utilities for the messages carried by itself, as well as the neighboring node. It then chooses those with the highest utility values (limited by the size of the data memory) by sending a request to the encountered node with hash values of the required messages. The neighbor sends the messages until the two nodes are no longer in range or all the required messages have reached their target.

The main contribution of the SPRINT algorithm is its utility function, which is based on the prediction of the future encounters of a node by using its contact history and social network information. SPRINT computes the utility u of a message M at node A as:

$$u(M, A) = w_1 * U_1(M, A) + w_2 * U_2(M, A)$$

In the formula, w_1 and w_2 are weight values which follow the conditions that $w_1 + w_2 = 1$ and $w_1 > w_2$, while U_1 and U_2 are utility components computed according to the following formulas:

$$U_1(M, A) = freshness(M) + p(M, A) * (1 - \frac{enc(M, A)}{24})$$

$$U_2(M, A) = c_e(M, A) * \frac{s_n(M) + hop(M) + pop(A) + time(M, A)}{4}$$

For U_1, the $freshness$ value can either be 0 if message M is old, or 0.5 if the message has been generated recently. We use it in order for newly-generated messages to start traveling through the network faster. This leads to a higher chance of delivering them to their destinations, since they reach more nodes. $p(M, A)$ is the probability of node A being able to deliver message M. Its computation starts with the analysis of the cache memory, counting how many times node A encountered each of the other nodes. If a node has been previously met in the same day of the week or in the same two-hour interval as the current time, the total encounters value is increased by 1. For the nodes encountered in the past that are in the same social community as node A, the total number of contacts is doubled. The reason 1 is added to the total number of encounters for nodes that have been met in the same day of the week or in the same two-hour interval when computing $p(M, A)$ is that there is a certain regularity in the activity of nodes in an opportunistic network. Therefore, there should be a higher probability of encountering nodes that have been seen in the same intervals. There is a similar reason for doubling the total number of contacts when the nodes met in the past are in the same community as node A, since a node tends to interact more with other members of its own social community. Having the number of encounters for each node, we can compute the probabilities of encountering them by performing a ratio between the number of encounters per node and the total number of encounters.

The next step consists of computing the number of encounters N that node A will have for each of the next 24 hours by using the Poisson probabilities and choosing the value with the highest probability as N. We pick only the first N nodes as potential future contacts for each of the next 24 hours (sorted by probability), and for the rest of them $p(M, A)$ is set to 0. The second component of the product uses $enc(M, A)$, which is the hour of the day when the destination of message M will be met according to the probabilities previously computed. If the destination will not be encountered, then $enc(M, A)$ is set to 24. We multiply $p(M, A)$ by $1 - \frac{enc(M,A)}{24}$ because the sooner a good target for a message is met, the sooner the node can delete the message from its memory and have room for others.

The second component of the utility value is U_2. Here, $c_e(M, A)$ is set to 1 if node A is in the same community as the destination of message M or if node A will ever encounter a node that has a social relationship with M, and 0 otherwise. The information computed for U_1 is used to analyze the potential future encounters of a node. The $s_n(M)$ component is 1 if the source and destination of M are not socially connected, because if a message doesn't have the source and destination in the same community, the chance of it being delivered by the source is low since it will mostly meet messages in its community. Therefore, the messages should be given to a different node that has the chance of reaching the destination community. $hop(M)$ is the normalized number of nodes that M has visited, $pop(A)$ is the normalized popularity value of A according to its social network information (i.e. number of Facebook friends in the opportunistic network), and finally $time(M, A)$ is the total time spent by node A in contact with M's destination. The *hop* and *time* values are used because nodes should travel as little as possible before reaching their destination.

We have put the condition $w_1 > w_2$ because messages destined for nodes that will be met in the future are more important than the others, since we know (with a certain probability) that they will be delivered eventually. While it may seem complicated to choose appropriate values for the two weights, in our experiments and tests we empirically observed that the best results were obtained when $w_1 = 0.7$ and $w_2 = 0.3$. However, we are also investigating possible methods to predict the most suitable weights according to the behavior of the network.

3.2 Selfish Node Detection and Avoidance

Since opportunistic network nodes are limited by the duration of a contact in terms of the amount of data they can transfer, reducing the number of pointless data transfers is paramount to obtaining good performance. By pointless transfers we refer to data that is sent to nodes that have no interest in delivering it to the destination, and which may end up dropping it shortly after receiving it. Therefore, a method of detecting these so-called “selfish” nodes and avoiding them is needed in order to have a more efficient algorithm. Thus, we proposed SENSE [11], a collaborative selfish node detection and incentive mechanism for opportunistic networks, and in this section we show how some of its elements can be integrated into SPRINT in order to improve its performance. We will refer to the SENSE-improved version of the SPRINT algorithm as SPRINT-SELF from now on.

Aside from its own unique ID, a SPRINT-SELF node stores social information, i.e. the relationships it has with other participants in the network, which is also helpful in designing a selfishness detection algorithm. Furthermore, as we have shown that nodes tend to be less selfish towards their own communities, having knowledge about a node’s social connections might help in deciding whether it was being selfish or if it couldn’t deliver a message due to other reasons (such as insufficient space in the data memory). Therefore, a node contains information about its own community as a list of nodes it has social relationships with. For the sake of simplicity, we do not take into consideration the strength of a social relationship, so two nodes are either socially connected or not. The community can either be taken from various social networks such as Facebook or Google+, but when this information is not available, we use a distributed community detection algorithm such as k-CLIQUE [16]. Also, the battery level is another contextual information that a node has access to at an encounter, and that it may use in the altruism computation process.

Aside from information regarding its ID, community, and battery level, each SPRINT-SELF node has its data and cache memories split into two sections each. The role of these sections is to control the number of messages stored for forwarding, as well as the amount of contact history each node is aware of. The data memory is therefore split into messages generated by the node itself (G) and messages that the node stores, carries and then finally forwards (or drops, if the memory is full) for other nodes (C). We decided to split the data memory in two because messages generated by the node should be kept for a longer period of time than other messages, since they are more important from the node’s standpoint. The cache memory is split into two sections as well: a list of past forwards (O) and a list of past receives (I). O contains information regarding past message forward operations performed either by the current node, or by other nodes. Therefore, the following information is stored: ID and community of both nodes that participated in the forward (sender and receiver), time of the encounter, encountered node’s battery level when the contact occurred, and metadata about the message that has been exchanged between the sender and the receiver (source and destination node IDs, priority, etc.). I contains information regarding past message receive operations performed either by the current node or by other nodes, and the stored data is similar to the one from O. Both I, as well as O, are updated whenever a new data exchange takes place (e.g. if node A sends a message to node B, an entry will be added in A’s O and one in B's I).

When two nodes A and B running SPRINT-SELF meet, they perform a series of steps. Firstly, each node checks if its battery level is above a certain threshold, and if it's not, then that node will not participate in the data exchange. Secondly, the two nodes exchange the I and O lists of past data transfers. When a node receives one of these lists, it updates its own list with the newly received information. This way, a node can have a more informed view of the behavior of various nodes in the network, through gossiping. We consider this to be an improvement over other existing selfishness detection solutions, since node A isn't simply told that node C has a certain degree of altruism based on the computations performed by B, but it is allowed to make the decision itself based on information gathered from encountered nodes. Since the sizes of O and I are limited, there may not be enough room for the entire history of contacts to be stored; therefore a node only keeps the most recent information in its own contact history memory.

After the nodes finish exchanging knowledge about past encounters, each of them advertises its own specific information, such as battery level and metadata about the messages it carries (which includes source and destination IDs). After computing utilities for all the messages as shown in Sect. 3.1, each node requests the messages with the highest utilities from the encountered neighbor. However, the SENSE-based improvement comes into play here, by allowing a node to refuse forwarding a certain message to another node. For this purpose, each node computes a perceived altruism value for every requested message from its own data memory, with regard to the encountered node. In other words, it computes how willing the encountered node is to forward a certain type of message. If this value is within certain limits, the message is sent. If not, it is skipped and the algorithm continues with the next message.

The formula for computing perceived altruism values for a node N and a message M based on the lists of past forwards (O) and receives (I) is:

$$altruism(N, M) = \sum_{o \in O, i \in I, o.m = i.m}^{N.id = o.d, N.id = i.s} type(M, o.m) * thr(o.b)$$

In the previous formula, a past encounter x has a field $x.m$ which specifies the message that was sent or received, $x.s$ is the source of the transfer, $x.d$ is the destination and $x.b$ is the battery level of the source. *type* is a function that returns 1 if the types of the two messages received as parameters are the same (in terms of communities, priorities, etc.), and 0 otherwise, while *thr* returns 1 if the value received as parameter is higher than a preset threshold, and 0 if it's not. Basically, the altruism computation function counts how many messages of the same type as M have been forwarded with the help of node N, when N's battery was at an acceptable level. In other words, we exclude the cases in which a node N didn't forward a message when it had (for example) 2% battery left, since we wouldn't expect it to do that anyway. The result of the formula is normalized with the total number of message exchanges, so the final value of the perceived altruism is between 0 and 1.

4 Experimental Setup and Results

This section presents an analysis of the SPRINT and SPRINT-SELF algorithms presented in Sect. 3 in terms of network performance. We highlight the improvements they bring compared to existing solutions such as distributed BUBBLE Rap [14] and Epidemic [25]. We compare with BUBBLE Rap because it is one of the most efficient opportunistic routing algorithms in terms of hit rate and delivery latency, and with Epidemic because we want to highlight that it doesn't behave as well as expected in realistic scenarios, and that using an algorithm such as SPRINT may yield better results.

4.1 Setup

Our tests were performed on several existing mobility traces, by analyzing four performance metrics that highlight the throughput, congestion and battery consumption of the ON devices.

4.1.1 Mobility Traces

Testing was performed on the MobEmu emulator [10], which parses human mobility traces and then applies an opportunistic routing and/or selfish node detection algorithm at every encounter between two nodes. We used four mobility traces, publicly available in the CRAWDAD archives[1]: UPB 2011 [10], St. Andrews [3], Intel and Cambridge [23]. UPB 2011 is a trace taken in an academic environment at the University Politehnica of Bucharest, where the participants were students and teachers carrying Android smartphones. It includes data collected for a period of 25 days by 22 participants. St. Andrews is a real-world mobility trace taken on the premises of the University of St. Andrews and its surroundings. The trace lasted for 79 days and involved 27 participants that used T-mote Invent devices. The main advantage of these two traces is that they also include social information about participating nodes, in the form of a graph of social connections. As shown in Sect. 3, this information is necessary to SPRINT. The other two traces (Intel and Cambridge) belong to a common collection of traces of Bluetooth sightings by groups of users carrying iMote devices in various situations [23]. The Intel trace was recorded for six days in the Intel Research Cambridge Laboratory, having nine participants: a stationary node and eight iMotes. The Cambridge trace was also taken for six days, at the Computer Lab of the University of Cambridge, having as participants 19 graduate students from the System Research Group. The main characteristics of the four traces are shown in Table 1. It can be seen that the traces we used vary in terms of number of participants, duration, total number of contacts and average contact duration, since they were chosen to highlight the capabilities of our solution in various different scenarios.

[1]`http://crawdad.cs.dartmouth.edu/`

Trace	Participants	Duration	Contacts	Cont. duration
UPB 2011	22	25 days	339	1,174 s
St. Andrews	27	79 days	112,264	3 s
Intel	9	6 days	1,364	658 s
Cambridge	19	6 days	4,229	308 s

Table 1: Mobility traces characteristics.

4.1.2 Performance Metrics

As stated before, there are four metrics that we use for analyzing the simulation data. The *hit rate* is the ratio between successfully delivered messages and the total number of generated messages, and signifies the efficiency of the ON algorithm, together with the *delivery latency*, which is defined as the time passed between the generation of a message and its eventual delivery to the destination. The *delivery cost*, defined as the ratio between the total number of messages exchanged and the number of generated messages, shows the congestion of the network. The *hop count* is the number of nodes that carried a message until it reached the destination on the shortest path, and should also be as low as possible in order to avoid node congestion. Improving the last two metrics may also lead to a decrease in battery consumption, since the number of data transfers would be reduced.

4.1.3 Testing Parameters

When testing, we assumed we were in an environment with devices that have limited resources, therefore the data memory of a node was limited. We tested with multiple values, ranging from 20 to 4500. The data memory size applies not only to SPRINT, but also to BUBBLE Rap, and signifies the number of messages a node can carry. For Epidemic, it is assumed that the data memory is unlimited. The cache size represents the amount of encountered nodes that the algorithm remembers, and after empirical testing we decided to set this value to 100 for each of the two sections (I and O, as described in Sect. 3).

For all experiments, we tried to model the generation of messages so as to resemble a real-life environment. Thus, every weekday, each node from the trace generates 30 messages with destinations chosen based on its social relationships using a Zipf distribution with an exponent of 1. Therefore, a node has a higher chance of sending a message to another member of its community than to other nodes. Inside the community, the destinations are chosen randomly. The time of the day when the messages are sent is randomly chosen inside the two-hour interval when the most contacts occur for each trace. We chose a Zipf distribution of messages because it has been shown that data requests and sends follow power law distributions [1]. For the UPB 2011 and St. Andrews traces, we used the social networking information available in the trace files as input to SPRINT. However, for Cambridge and Intel, where such information wasn't available, k-CLIQUE was employed in order to dynamically detect social communities on the fly. The contact and community thresholds for k-CLIQUE were thus set to 1200 seconds and 6, respectively.

For the battery-aware scenarios we simulated (presented in Sect. 4.2), we assumed that each node starts the simulation with a random amount of battery left, which decreases every second and lasts a maximum of 24 hours, after which the node has to recharge its battery, making it unable to participate in the network for one hour. When the battery level is under 20%, a node doesn't accept to receive messages from other nodes (but it still forwards its own messages). The SPRINT-SELF battery threshold (under which it doesn't consider a node as being selfish) was chosen to be 25%.

In real-life, ON nodes are not only limited by battery when applying a routing algorithm, but also by the contact duration. A node won't necessarily have time to deliver all its intended messages to another node, so this is why we added this limitation to our simulations as well. Therefore, we assumed that a node is able to deliver a message in 3 seconds.

4.2 Results

We present here several scenarios that we tested with, that highlight the benefits of SPRINT and SPRINT-SELF.

4.2.1 Influence of Battery Usage and Transfer Speed on Hit Rate

The first scenario we performed has the role of highlighting the influence of choosing realistic testing parameters on the outcome of the tests. Generally, when routing or dissemination in ONs is analyzed, it is assumed that devices can transfer any amount of data to each other instantly, and that the participating nodes have unlimited battery power, thus being available for communication at any point in time. However, in real life we are far from this ideal situation. Devices have batteries that drain in time, and even more so when communication is performed. Furthermore, the number of messages two nodes can exchange when they are in contact is limited by the encounter duration, so a node may only receive a small part of the data it wants.

This is why we show in Fig. 1 the way that hit rate decreases when the two limitations are applied on the UPB 2011 trace, where nodes have a data memory of 4500. First of all, it can be seen that for the unrestricted run, the hit rate is around 47% for all three algorithms we tested (BUBBLE Rap, SPRINT and Epidemic). In such a situation, the Epidemic algorithm yields the best possible hit rate, since nodes exchange all the messages in their data memories when they are in contact. The fact that BUBBLE Rap and SPRINT achieve a hit rate very close to the maximum possible one shows that the algorithms behave well for the unrestricted scenario.

However, when limiting the amount of data nodes can transfer per time unit (as shown in Sect. 4.1), the hit rate drops for all three algorithms. The drop is higher for BUBBLE Rap than for Epidemic and SPRINT, which means that it doesn't behave so well in such situations. Epidemic's hit rate decreases because

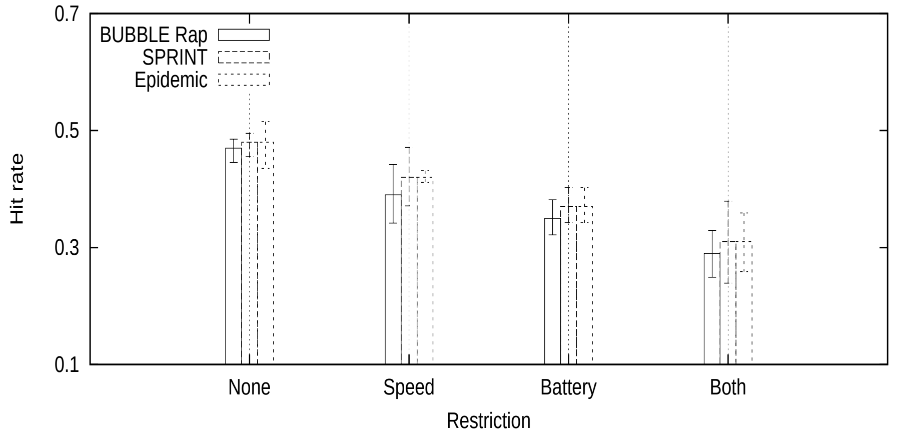


Figure 1: Influence of battery usage and transfer speed on hit rate.

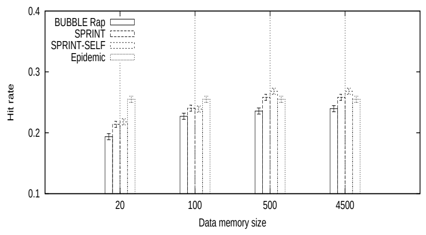
the algorithm transfers messages from a node to another one in a given order. If contact durations are low, then it may never have enough time to transfer a certain message if it is located at the end of the data memory. SPRINT's performance is worse because it doesn't take into consideration the history of a node's transfers when deciding what messages should be exchanged, which may lead to transferring the wrong messages and then not having any more time to transfer the correct ones as well. The same conclusions can be drawn when looking at the influence of draining battery life on the three algorithms.

Finally, when adding both variable battery life and limited transfer speed to the UPB 2011 trace, all three algorithms obtain hit rates that are up to 40% lower than on the unrestricted scenario, so having more realistic testing parameters heavily influences the outcome of a routing algorithm. This is why all the scenarios performed in the remainder of this section assume an approach where a node's battery lasts for 24 hours and takes one hour to recharge, and a node can transfer a message in 3 seconds.

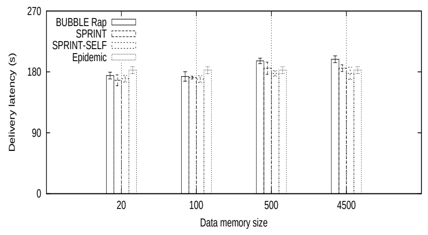
4.2.2 UPB 2011

The following subsections (including this one) present the results of running BUBBLE Rap, SPRINT, Epidemic and SPRINT-SELF on each of the four traces described in Sect. 4.1.

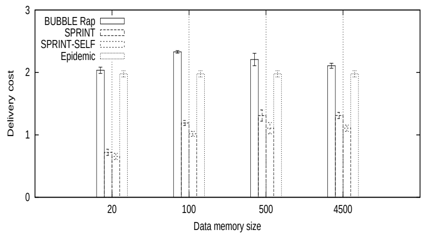
Figure 2(a) presents the hit rate of BUBBLE Rap, SPRINT, SPRINT-SELF and Epidemic for the UPB 2011 trace, when the data memory size ranges from 20 to 4500 messages. It can be seen that, regardless of the size of the data memory, BUBBLE Rap performs the worst out of all four algorithms. The reason for this behavior is that BUBBLE Rap relies on nodes with high centrality values to carry the load of routing a large part of the total messages, which leads to two



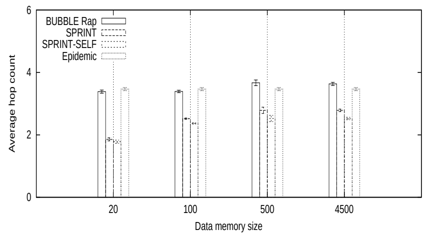
(a) Hit rate



(b) Delivery latency



(c) Delivery cost



(d) Average hop count

Figure 2: ON routing results for the UPB 2011 trace.

potential issues. Firstly, the high-centrality nodes can easily get congested this way and start getting buffer overflows, which in turn leads to them having to drop older messages. If other copies of said messages don't exist in the network anymore, they never get to reach their intended destinations. Secondly, since high-centrality nodes have to perform many data transmissions, naturally their battery gets depleted quicker and they become inactive (for charging) more often. While being inactive, they may miss contacts with some nodes they carry messages for, and which they may never encounter again for the remainder of the trace.

As shown in the previous scenario, SPRINT yields similar results to Epidemic in terms of hit rate, especially for higher data memory sizes. When the data memory can contain fewer messages, the routing algorithm has to discard some of them whenever there is a buffer overflow. In addition to this, Epidemic assumes that it has unlimited data memory, so it's not limited by space, only by the duration of the contact. Figure 2(a) also shows that, for high enough data memory sizes, SPRINT-SELF outperforms even Epidemic in terms of hit rate. This happens mainly because of a severe limitation of Epidemic, namely that when a contact between a node A and a node B occurs, it transfers all messages from A to B in order. Thus, when there are only short contacts between A and B, some messages located at the far end of their data memories may never have the chance of being successfully delivered. The advantage of SPRINT-SELF over SPRINT is that it doesn't forward messages to selfish nodes, so it maximizes the duration of a contact by only transferring messages that the receiving node has a high chance of successfully delivering.

Figure 2(b) presents the delivery latencies for the four algorithms. Again, SPRINT-SELF performs best, regardless of the data memory size. The latency is reduced with up to 10%, even when SPRINT-SELF's hit rate is higher. So our algorithm not only delivers data to more nodes in the network, but also does it quicker. Although opportunistic networks are DTNs, increasing the delivery latency leads to a better user experience for the participants in the network.

Aside from improving hit rate and delivery cost, the SPRINT algorithms also decrease the overall node and network congestion, thus performing fewer transfers and saving battery power in the process. This is shown in Fig. 2(c) and Fig. 2(d). A lower delivery cost means that fewer messages are sent in the network, and this value is improved by SPRINT and SPRINT-SELF by up to 70% for some situations (i.e. 70% less messages are exchanged by nodes in the ON, although the hit rate and delivery latency are improved). Finally, the average hop count is reduced by almost 50% for our algorithms, which shows that nodes travel to 50% less other nodes before eventually reaching their destinations. For both these situations, SPRINT and SPRINT-SELF perform the best, with the selfish node detection mechanism helping SPRINT-SELF achieve the best results out of all four algorithms we tested with.

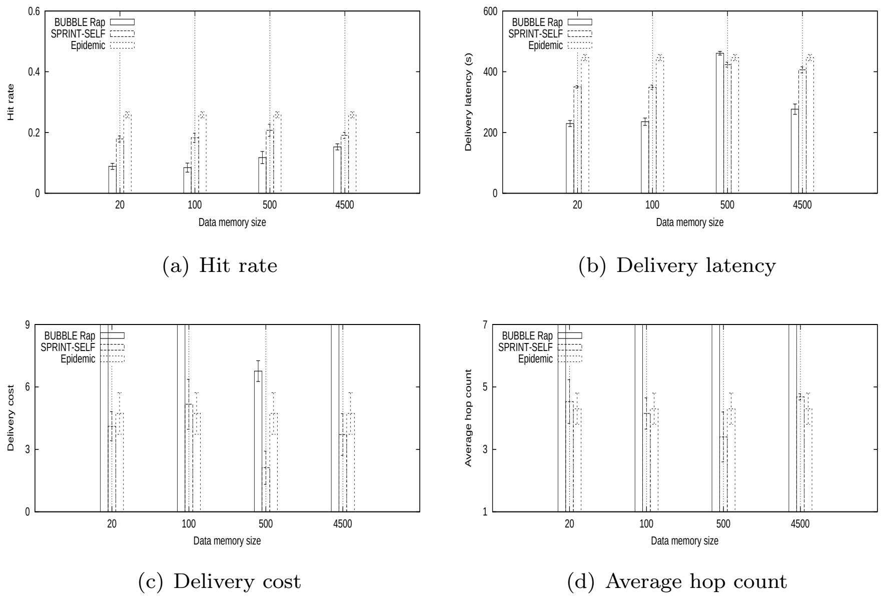
Since we have shown that SPRINT-SELF performs better than SPRINT regardless of the metric, in the following sections we only show results obtained for BUBBLE Rap, Epidemic and SPRINT-SELF.

4.2.3 St. Andrews

The main difference between the UPB 2011 trace and St. Andrews is that the latter is taken not only in an enclosed academic environment, but also in and around the town of St. Andrews. This leads, as seen in Table 1, to a much larger number of contacts (about 300 times more than for UPB 2011) and to a smaller average contact duration (3 seconds as opposed to 1,174).

The hit rates obtained by BUBBLE Rap, SPRINT-SELF and Epidemic on the St. Andrews trace are shown in Fig.3(a). The results show that, this time, Epidemic obtains the best results out of all the routing algorithms tested. The reason it outperforms SPRINT-SELF and BUBBLE Rap is that, since there are much more contacts here that in the previous trace, a larger data memory would be required in order to store useful messages from all the encountered nodes. As stated before, Epidemic assumes that a node's data memory is unlimited, so it has enough room for an infinite amount of messages. However, SPRINT-SELF still performs up to 10% better than BUBBLE Rap. Increasing SPRINT-SELF's data memory size leads to it eventually performing better than Epidemic.

Regarding delivery latency (seen in Fig. 3(b)), the situation is opposite to hit rates: BUBBLE Rap has the best latencies, whereas Epidemic performs the worst. However, the results shown here should be corroborated with the ones in Fig. 3(a): the algorithm with the highest hit rate has the lowest latency and vice versa. This is natural, since Epidemic is able to reach a greater number of



(a) Hit rate (b) Delivery latency

(c) Delivery cost (d) Average hop count

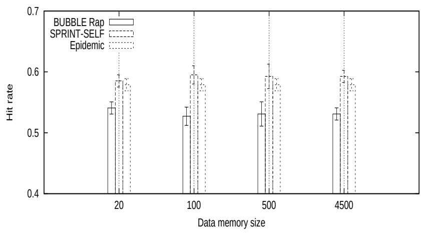
Figure 3: ON routing results for the St. Andrews trace.

nodes in the ON, but some of these nodes may have very few contacts and may be met after a long time, which affects the average delay of the algorithm.

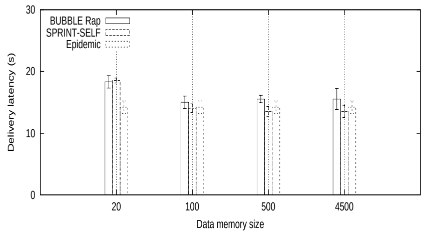
Finally, Fig.3(c) and Fig.3(d) show the delivery cost and average hop count. For most of the situations, SPRINT-SELF exhibits the best values, which means a reduction of network and node congestion. Furthermore, fewer messages exchanged in the ON lead to a lower overall battery consumption. Although Epidemic has a better hit rate and delivery latency that SPRINT-SELF, our algorithm still behaves better in terms of delivery cost and average hop count. Another very important conclusion that can be drawn from Fig.3(c) and Fig.3(d) is that BUBBLE Rap has a very poor performance regarding these two metrics analyzed here. The charts have been scaled down in order to see the values for SPRINT-SELF and Epidemic, but (except for a data memory of 500), the delivery cost and hop count of BUBBLE Rap are very high. This means that BUBBLE Rap has a tendency of passing a message from a node to another back and forth, when the two nodes have very close centrality values and the highest one varies from one node to the other. SPRINT-SELF does not exhibit this problem because it also analyzes a node's social connections, in addition to performing a prediction of its future encounters.

4.2.4 Intel

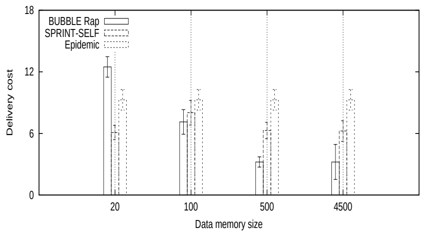
The Intel trace results (in Fig.4) show that SPRINT-SELF performs the best out of the three algorithms tested for all four metrics analyzed. The hit rate



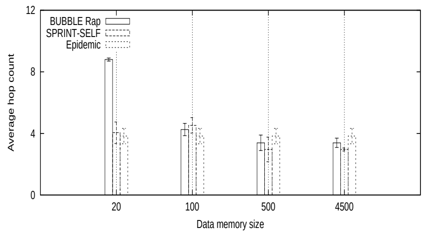
(a) Hit rate



(b) Delivery latency



(c) Delivery cost



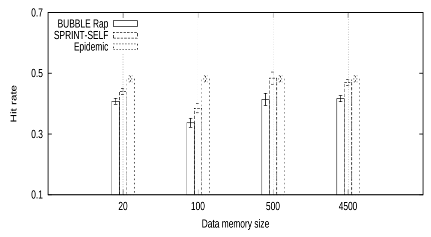
(d) Average hop count

Figure 4: ON routing results for the Intel trace.

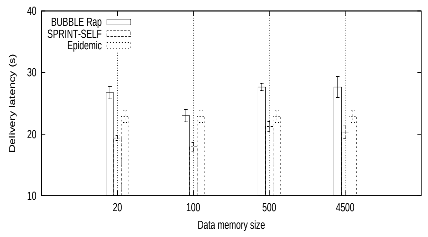
is much higher than BUBBLE Rap's (an improvement of up to 7%) and also better than the one obtained by Epidemic, whereas the latency is lower than for the other algorithms for higher data memory sizes. Not only is SPRINT-SELF's network performance better, but it also offers lower congestion and better overall battery saving. The delivery cost for our solution is the lowest out of all three algorithms for a data memory of 20, and lower than Epidemic but higher than BUBBLE Rap for the rest of the experiments. However, BUBBLE Rap has a better delivery cost because it delivers messages to a lower number of nodes. The situation is similar for the average hop count, where, for high enough data memory sizes (500 and 4500), SPRINT-SELF has the best values out of all three algorithms.

4.2.5 Cambridge

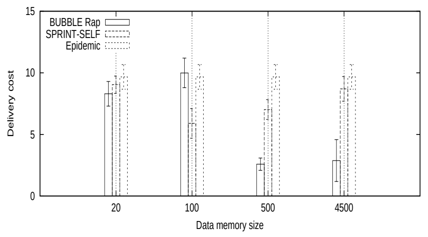
Finally, the Cambridge trace shows similar results to Intel (see Fig. 5). The hit rate at SPRINT-SELF is higher than BUBBLE Rap's and, for higher data memory sizes, very close to Epidemic's results. In the meantime, the delivery latency values obtained by our algorithm are the lowest (and thus the best) out of all three solutions tested. The delivery cost and average hop count are also better for SPRINT-SELF than for Epidemic, but worse than for BUBBLE Rap. However, as stated before, BUBBLE Rap manages to deliver data to fewer nodes, so it doesn't have to exchange as many messages as SPRINT-SELF of Epidemic.



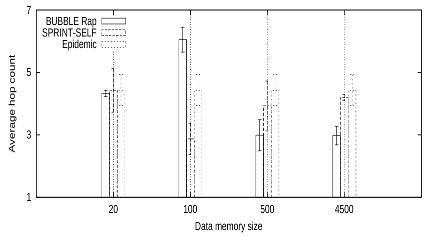
(a) Hit rate



(b) Delivery latency



(c) Delivery cost



(d) Average hop count

Figure 5: ON routing results for the Cambridge trace.

5 Conclusion

We have presented in this article an algorithm entitled SPRINT that makes use of information about the social relationships between owners of the mobile devices from an opportunistic network when routing data. Moreover, SPRINT can correctly predict the number of contacts a node will have in a given time interval. We also added a selfish node detection and avoidance mechanism and created an improved version of SPRINT entitled SPRINT-SELF, which is able to outperform existing solutions such as BUBBLE Rap and Epidemic on realistic scenarios where the transfer speed of a node is limited and the battery level decreases in time. Our solution behaves better in terms of network performance (hit rate and delivery latency) as well as node and network congestion (delivery cost and hop count). Moreover, reducing the latter two metrics helps decrease the overall battery consumption in the ON, since nodes have to perform fewer data transfers. The experiments that prove our algorithm's effectiveness were performed on existing mobility traces in various real-life situations.

Acknowledgment

This work was partially supported by the project "ERRIC - Empowering Romanian Research on Intelligent Information Technologies/FP7-REGPOT-2010-1", ID: 264207. The work has been cofounded by the Sectoral Operational Programme Human Resources Development 2007-2013 of the Romanian Ministry of Labour, Family and Social Protection through the Financial Agreement POSDRU/89/1.5/S/62557.

References

[1] Lada A. Adamic and Bernardo A. Huberman. Power-law distribution of the world wide web. *Science*, 287(5461):2115, March 2000.

[2] G. Bigwood and T. Henderson. Ironman: Using social networks to add incentives and reputation to opportunistic networks. In *Privacy, security, risk and trust (passat), 2011 ieee third international conference on and 2011 ieee third international conference on social computing (socialcom)*, pages 65 –72, oct. 2011.

[3] Greg Bigwood, Devan Rehunathan, Martin Bateman, Tristan Henderson, and Saleem Bhatti. Exploiting self-reported social networks for routing in ubiquitous computing environments. In *Proceedings of the 2008 IEEE International Conference on Wireless & Mobile Computing, Networking & Communication*, pages 484–489, Washington, DC, USA, 2008. IEEE Computer Society.

[4] Chiara Boldrini, Marco Conti, Franca Delmastro, and Andrea Passarella. Context- and social-aware middleware for opportunistic networks. *J. Netw. Comput. Appl.*, 33(5):525–541, 2010.

[5] Chiara Boldrini, Marco Conti, Jacopo Jacopini, and Andrea Passarella. HiBOp: a History Based Routing Protocol for Opportunistic Networks. In *World of Wireless, Mobile and Multimedia Networks, 2007. WoWMoM 2007. IEEE Int. Symp. on a*, pages 1–12, 2007.

[6] Chiara Boldrini, Marco Conti, and Andrea Passarella. Exploiting users' social relations to forward data in opportunistic networks: The HiBOp solution. *Pervasive Mob. Comput.*, 4:633–657, 2008.

[7] Radu Ciobanu and Ciprian Dobre. Data dissemination in opportunistic networks. In *18th Int. Conf. on Control Systems and Computer Science*, CSCS-18, pages 529–536, 2011.

[8] Radu Ioan Ciobanu and Ciprian Dobre. Predicting encounters in opportunistic networks. In *Proceedings of the 1st ACM workshop on High performance mobile opportunistic systems*, HP-MOSys '12, pages 9–14, New York, NY, USA, 2012. ACM.

[9] Radu-Ioan Ciobanu, Ciprian Dobre, and Valentin Cristea. Reducing congestion for routing algorithms in opportunistic networks with socially-aware node behavior prediction. In *AINA*, pages 554–561, 2013.

[10] Radu Ioan Ciobanu, Ciprian Dobre, Valentin Cristea, and Dhiya Al-Jumeily. Social opportunistic networking. In *Proceedings of the 11th International Symposium on Parallel and Distributed Computing*, ISPDC 2012, 2012.

[11] Radu-Ioan Ciobanu, Ciprian Dobre, Mihai Dascalu, Stefan Trausan-Matu, and Valentin Cristea. Collaborative selfish node detection with an incentive mechanism for opportunistic networks. In *Proc. of 5th IFIP/IEEE International Workshop on Management of the Future Internet*, IFIP/IEEE ManFI 2013, pages 1161–1166, 2013.

[12] Marco Conti, Silvia Giordano, Martin May, and Andrea Passarella. From opportunistic networks to opportunistic computing. *Comm. Mag.*, 48:126–139, 2010.

[13] Enrique Hernández-Orallo, Manuel D. Serrat Olmos, Juan-Carlos Cano, Carlos T. Calafate, and Pietro Manzoni. Evaluation of collaborative selfish node detection in manets and dtns. In *Proceedings of the 15th ACM international conference on Modeling, analysis and simulation of wireless and mobile systems*, MSWiM '12, pages 159–166, New York, NY, USA, 2012. ACM.

[14] Pan Hui, Jon Crowcroft, and Eiko Yoneki. BUBBLE Rap: social-based forwarding in delay tolerant networks. In *Proc. of the 9th ACM int. symp. on Mobile ad hoc networking and computing*, MobiHoc '08, pages 241–250, New York, USA, 2008. ACM.

[15] Pan Hui, Kuang Xu, V.O.K. Li, J. Crowcroft, V. Latora, and P. Lio. Selfishness, altruism and message spreading in mobile social networks. In *INFOCOM Workshops 2009, IEEE*, pages 1–6, april 2009.

[16] Pan Hui, Eiko Yoneki, Shu Yan Chan, and Jon Crowcroft. Distributed community detection in delay tolerant networks. In *Proc. of 2nd ACM/IEEE inter. workshop on Mobility in the evolving internet architecture*, MobiArch '07, pages 7:1–7:8, New York, NY, USA, 2007. ACM.

[17] Alexander Ihler, Jon Hutchins, and Padhraic Smyth. Learning to detect events with markov-modulated poisson processes. *ACM Trans. Knowl. Discov. Data*, 1(3), December 2007.

[18] Sushant Jain, Kevin Fall, and Rabin Patra. Routing in a delay tolerant network. *SIGCOMM Comput. Commun. Rev.*, 34(4):145–158, August 2004.

[19] Dmytro Karamshuk, Chiara Boldrini, Marco Conti, and Andrea Passarella. Human mobility models for opportunistic networks. *IEEE Comm. Magazine*, 49(12):157–165, 2011.

[20] A. Lavinia, C. Dobre, F. Pop, and V. Cristea. A failure detection system for large scale distributed systems. In *Complex, Intelligent and Software*

Intensive Systems (CISIS), 2010 International Conference on, pages 482 –489, feb. 2010.

[21] Vincent Lenders, Martin May, Gunnar Karlsson, and Clemens Wacha. Wireless ad hoc podcasting. *SIGMOBILE Mob. Comput. Commun. Rev.*, 12:65–67, January 2008.

[22] Hoang Anh Nguyen, Silvia Giordano, and Alessandro Puiatti. Probabilistic Routing Protocol for Intermittently Connected Mobile Ad hoc Network (PROPICMAN). *2007 IEEE Int. Symp. on a World of Wireless Mobile and Multimedia Networks*, pages 1–6, 2007.

[23] J. Scott, R. Gass, J. Crowcroft, P. Hui, C. Diot, and A. Chaintreau. CRAWDAD data set Cambridge/Haggle (v. 2009-05-29). Downloaded from http://crawdad.cs.dartmouth.edu/cambridge/haggle, May 2009.

[24] Chaoming Song, Yehui Qu, Nicholas Blumm, and Albert-László Barabási. Limits of Predictability in Human Mobility. *Science*, 327:1018–1021, 2010.

[25] A. Vahdat and D. Becker. Epidemic Routing for Partially Connected Ad Hoc Networks, 2000.

[26] Eiko Yoneki, Pan Hui, ShuYan Chan, and Jon Crowcroft. A socio-aware overlay for publish/subscribe communication in delay tolerant networks. In *Proceedings of the 10th ACM Symposium on Modeling, analysis, and simulation of wireless and mobile systems*, MSWiM '07, pages 225–234, New York, NY, USA, 2007. ACM.

[27] Qiang Zhou, Jing Ying, and Minghui Wu. A detection method for uncooperative nodes in opportunistic networks. In *Network Infrastructure and Digital Content, 2010 2nd IEEE International Conference on*, pages 835 –838, sept. 2010.