# Routing in Socially Selfish Delay Tolerant Networks

Qinghua Li, Sencun Zhu, Guohong Cao
Department of Computer Science & Engineering
The Pennsylvania State University, University Park
Email: {qxl118,szhu,gcao}@cse.psu.edu

*Abstract*—**Existing routing algorithms for Delay Tolerant Networks (DTNs) assume that nodes are willing to forward packets for others. In the real world, however, most people are *socially selfish*; i.e., they are willing to forward packets for nodes with whom they have social ties but not others, and such willingness varies with the strength of the social tie. Following the philosophy of *design for user*, we propose a Social Selfishness Aware Routing (SSAR) algorithm to allow user selfishness and provide better routing performance in an efficient way. To select a forwarding node, SSAR considers both users' willingness to forward and their contact opportunity, resulting in a better forwarding strategy than purely contact-based approaches. Moreover, SSAR formulates the data forwarding process as a Multiple Knapsack Problem with Assignment Restrictions (MKPAR) to satisfy user demands for selfishness and performance. Trace-driven simulations show that SSAR allows users to maintain selfishness and achieves better routing performance with low transmission cost.**

## I. INTRODUCTION

Delay Tolerant Networks (DTNs) [1] have the unique feature of intermittent connectivity, which makes routing quite different from other wireless networks. For example, since an end-to-end connection is hard to setup, store-carry-and-forward is used to deliver the packets to the destination. Although many routing algorithms [2]–[7] have been proposed to increase data delivery reliability, they are purely based on contact opportunity; i.e., they are designed without considering users' willingness and implicitly assume that all nodes are willing to forward packets for others.

In the real world, most people are selfish. As a result, in civilian DTNs such as PeopleNet [8] and Pocket Switched Network [9], a node may not be willing to forward packets for others. Then, previous algorithms may not work well since some packets are forwarded to nodes unwilling to relay, and will be dropped. Although many researchers have designed incentive schemes to stimulate selfish nodes to forward packets in mobile ad hoc networks [10], [11], they go to another extreme; i.e., they believe that users are selfish and are not willing to forward packets for anyone else.

To capture user selfishness in a more realistic manner, we have two observations from the social perspective. First, a selfish user is usually willing to help others with whom he has social ties (e.g., friends, coworkers, roommates), because he got help from them in the past or will probably get help from them in the future. In this paper, a social tie means an interpersonal tie that falls into the *strong* or *weak* category defined in [12]. Second, for those with social ties, a selfish user may give *different* preferences. That is, he is willing to provide better service to those with stronger ties than those with weaker ties, especially when there are resource constraints. For easier presentation and comparison, such refined selfishness model will be referred to as *social selfishness* and the previously well studied simple model is called *individual selfishness*. Social selfishness is not totally contradictory to individual selfishness, but is a generic extension to it. When a node has no social tie to the outside world, his social selfishness becomes individual selfishness. However, in most cases, social selfishness conveys more meaning.

Social selfishness will affect node behaviors. As a forwarding service provider, a node will not forward packets received from those with whom it has no social ties, and it gives preference to packets received from nodes with stronger ties when the resource is limited. Thus, a DTN routing algorithm should take the social selfishness into consideration.

In this paper, different from existing incentive based schemes which stimulate individually selfish nodes to forward for all other nodes, we follow a new philosophy of "design for user". We will take social selfishness as a user demand and allow socially selfish nodes to behave in the aforementioned ways to satisfy such demand. With this in mind, we need to address the problem of *how to enforce users' social selfishness in routing*. This is not easy since the routing performance (e.g., the number of packets delivered to their destinations) may be affected when social selfishness is considered. For example, when a packet is forwarded to a node that is unwilling to forward, it will most likely be dropped. As such, we also need to address *how to maintain acceptable routing performance when social selfishness is maintained*.

We propose a Social Selfishness Aware Routing (SSAR) algorithm to address these challenges. To maintain social selfishness, SSAR allocates resources such as buffers and bandwidth based on packet priority which is related to the social relationship among nodes. To maintain the routing performance, SSAR quantifies the relay's willingness to evaluate its forwarding capability and thus reduces the packet dropping rate. Furthermore, SSAR formulates the forwarding process as a Multiple Knapsack Problem with Assignment Restrictions (MKPAR). It forwards the most effective packets for social selfishness and routing performance. Our trace-driven simulations show that SSAR achieves very good selfishness and

routing performance with low transmission cost.

We make the following contributions:

- We introduce social selfishness into DTN routing.
- We present a routing algorithm SSAR for DTNs, which follows the philosophy of *design for user*.
- We incorporate user willingness into relay selection. Combined with contact opportunity, it results in a better metric to evaluate a node's forwarding capability.
- We formulate the forwarding process as an MKPAR and provide a heuristic based solution.

The remainder of this paper is structured as follows. Section II presents an overview of SSAR. Section III gives the detailed design. Section IV introduces the trace-driven simulations and discusses the results. The last three sections present discussions, related work, and conclusions, respectively.

## II. SSAR Overview

In this section, we first introduce our design philosophy, then discuss our models and assumptions, and finally give an overview of SSAR and explain how it works.

### A. Philosophy: Design for User

The existing literature has focused on addressing individual selfishness such as using reputation-based, credit-based, or game-theory based approaches to stimulate users to cooperate and forward packets for others. If the nodes cooperate with others, they will get help from others as a return; if not they will be punished; e.g., being deprived of access to the network.

However, these incentive schemes may not be directly applied to deal with social selfishness, since the incentive schemes do not consider social selfishness. By using incentives, every node will have to provide service to others no matter there is a social tie or not. As a result, social selfishness is violated.

We address this problem from a different point of view. We allow users to behave as what their social selfishness requires, but try to improve the routing performance under the social selfish behavior. Our underlying philosophy is that social selfishness is a kind of user demand that should be satisfied. It should be treated as a design metric to measure the user satisfaction, similar to other traditional performance metrics such as data delivery ratio and delay. We call such design philosophy "design for user".

### B. Models and Assumptions

*Network Graph* We model the socially selfish network as a fully-connected weighted directed graph, where the vertex set $\mathcal{V}$ consists of all the nodes and the edge set $\mathcal{E}$ consists of the social links between nodes. The weight of edge $\overrightarrow{AB}$ is $A$'s *willingness* to forward packets for $B$. The weight of edge $\overrightarrow{AB}$ and that of $\overrightarrow{BA}$ may be different. The value of willingness is a real number within $[0, 1]$, where 0 means unwilling to forward and 1 means the most willing to forward. The social willingness between two nodes depends on the social tie between them. The stronger the social tie is, the larger the social willingness is.
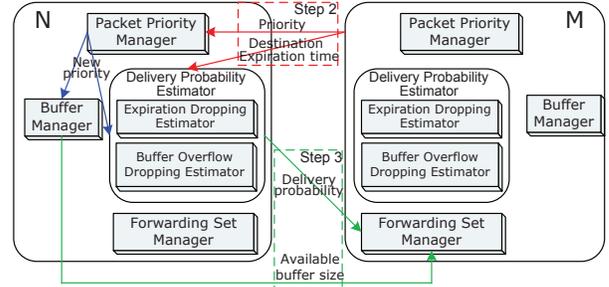


Fig. 1. SSAR overview where node N meets node M. The dashed rectangles enclose the information exchanged in step 2 and step 3 (Section II-D).

We assume that the willingness information is available. This should be achievable since each node only needs to know its willingness to forward for others. From the system perspective, this can be done as one system configuration step, in which the user assigns its willingness values for people he knows via some user interface in the mobile device. Actually, one recent study already shows that people can quantitatively rate their friendships [13]. In addition, the user can set a default value (e.g., 0) for strangers. To be more user-friendly, the interface can provide several willingness levels (e.g., "most", "much", "average", "poor", "none") for the user to choose from. Such levels can be easily mapped into numerical values. The willingness information is configured when the user joins the network or migrates to a new mobile device, and is updated when his social ties change. However, such update is quite infrequent since social ties are usually stable.

*Network Model* In DTNs, nodes have limited bandwidth and computational capability. As in other studies [4], we assume each node has unlimited buffer for its own packets, but limited buffer for others. As for data traffic, we only consider unicast, and assume each packet has a certain lifetime (i.e., TTL). We further assume bidirectional links, which can be provided by some MAC layer protocols, e.g., IEEE 802.11.

*Trust Model* We assume the source of a packet is *anonymous* to intermediate nodes. For example, the source ID can be encrypted in a way so that only the destination can decrypt. Then intermediate nodes provide data forwarding service only based on the previous hop information. This assumption is not essential to SSAR, and we add it just to simplify the routing model. We also assume that some authentication service is available so that one node can not impersonate another. Otherwise, a node may claim to be someone else to obtain forwarding services from that node's social ties. How to provide such authentication service has been well studied (e.g., [14]) and is out of the scope of this paper.

*Adversary Model* In this paper, we only consider socially selfish behaviors. Malicious attacks (e.g., DoS, wormhole, blackhole) and free-riding behaviors are not our focus. This is not because we do not think they are important, but because we believe they deserve separate studies. Thus, we leave them for future work, and discuss more in Section V.

### C. Architecture

Figure 1 shows the architecture of SSAR, which has the following four components.

*Packet priority manager*  It calculates a priority for each buffered packet based on the willingness between nodes that the packet has traversed. This priority of a packet measures the social importance of the packet to the node.

*Buffer manager*  It manages buffers based on packet priority: (i) packets with priority 0 will not be buffered; (ii) when buffer overflows, packets of low priority are dropped first. That is, a new incoming packet can preempt the buffer occupied by lower-priority packets. This policy exactly follows the philosophy of "design for user".

*Delivery probability estimator*  It estimates a node's "delivery probability" of a packet, which is used to quantify the node's forwarding capability for that packet. A node forwards the packet to the neighbor with a higher delivery probability.

Traditionally, the quality of a relay is measured solely based on its contact opportunity to the destination node. SSAR measures the delivery probability of a node based on both of its contact opportunity to the destination and its willingness to forward. It is straightforward that a node with a low contact opportunity should not be a relay. Interestingly, a node with a high contact opportunity but low willingness should not be a relay either. This is illustrated in Figure 2(a). Suppose $S$ has a packet $m1$ to send to $D$, and it successively meets $A$, $C$, and $B$. If only contact opportunity is considered, it will forward $m1$ to $A$. Unfortunately, $A$ will drop $m1$ since it is unwilling to forward for $S$ (the edge weight is 0). SSAR will avoid such forwarding. Though $C$ is willing to forward $m1$, its willingness is so low that $m1$ may suffer high risk of being dropped, so SSAR will avoid such forwarding. As a result, $B$ is the optimal forwarder for $m1$ in this scenario, since it has high willingness to forward and a high contact opportunity.

*Forwarding set manager*  After a node determines a set of packets that should be forwarded to a better relay, existing routing protocols greedily transmit them no matter the receiver has enough buffers to hold these packets or not [5]. Obviously, bandwidth will be wasted if the transmitted packets are dropped due to buffer overflow. To address this issue, the forwarding set manager decides which packets to transmit by solving an MKPAR formulation. It considers the buffer constraint and transmits the packets that are most effective for social selfishness and routing performance.

### D. The Protocol

We use an example (Figure 1) to illustrate how SSAR works in the following five steps.

1) After neighbor discovery, node $N$ and $M$ deliver packets destined to each other in the decreasing order of priority. During packet delivery, they also exchange information related to their willingness to forward.
2) If $N$'s willingness for $M$ is positive, $M$ sends $N$ a summary list of ⟨destination ID, expiration time, priority⟩ for its buffered packets.
3) From the priority information, $N$ calculates the new priority value for each packet (Section III-A). Based on the new priority and other information in the summary list, $N$ calculates its delivery probability (Section III-B)
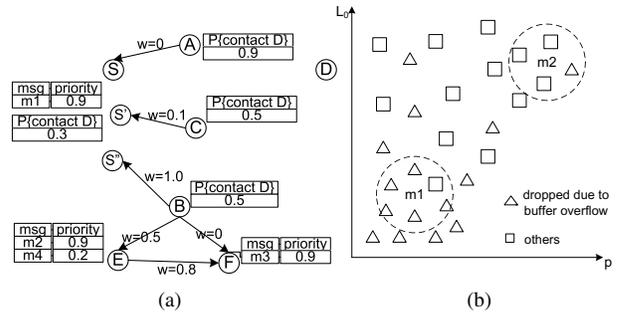


Fig. 2.  (a) Examples of willingness-aware forwarding. (b) Heuristics used to estimate $P_{over}$. Triangles and squares are historical samples.

and available buffer size (Section III-C) for each packet in the list, and returns them to $M$.
4) $M$ determines a candidate set of packets for which $N$ has higher delivery probabilities.
5) Considering the available buffer size information, $M$ further decides which candidates to transmit by solving the MKPAR (Section III-C) formulation. Packets will be deleted after being forwarded, so there is only one copy for each packet.

Without loss of generality, in the last four steps we only describe how node $M$ determines which packets to transfer to node $N$. Node $N$ does so in similar ways.

Though not very frequent in opportunistic DTNs, a node may be in contact with multiple neighbors at the same time. Then it would be very difficult to extend the MKPAR formulation to the whole neighborhood. As a simple solution, the node contacts neighbors one by one.

### III. DETAILED DESIGN

This section describes the detailed design of the packet priority calculation, the delivery probability estimation, and the forwarding set optimization.

### A. Packet Priority

When a node receives and buffers a packet, it assigns a priority $p$ to the packet. We borrow the idea of transitive trust [15] from the literature on reputation system and calculate packet priority in a chained way. Formally:

$$p_i = p_{i-1} \cdot \omega \quad i \geq 1 \tag{1}$$

where $p_i$ is the packet's priority in its $i^{th}$ hop and $\omega$ is the $i^{th}$ hop's willingness to forward the packets from the $(i-1)^{th}$ hop. The initial priority $p_0$ is set by the source. Since source anonymity is assumed, the packet source is not considered by intermediate hops.

The priority assignment method and the buffer management policy are used to enforce social selfishness. First, packets that traverse stronger social edges tend to have higher priorities. As shown in Figure 2(a), although $m1$, $m2$, and $m3$ have the same priority in the previous hop, they will receive different services in $B$ after traversing different links. $m3$ will not receive any forwarding service; $m1$ will receive better service than $m2$. Second, packets from the same upstream node are also differentiated. In this example, $m2$ receives better service than $m4$ in $B$ although they come from the same node.

Since priority is updated hop by hop, it may improve cooperation in some cases. In Figure 2(a), if $m3$ from $F$ arrives at $B$ via $E$, its priority becomes 0.36 (instead of 0 through direct transmission) and will receive $B$'s service.

### B. Delivery Probability Estimation

Suppose each packet has some expiration time, the question is: at a given time $t$, how to estimate node $N$'s probability of delivering packet $m$ to its destination $D$ before its expiration time $t_{exp}$?

*1) Overall Delivery Probability:* As a starting point, we first look at the complement of delivery probability, i.e., dropping probability, which is the probability that $m$ will be removed from $N$'s buffer before being delivered to $D$. Assume that $N$ can deliver $m$ when it meets $D$, then there are two cases of dropping. First, $m$ expires before $N$ reaches $D$ due to insufficient contact opportunity. Second, $N$ drops $m$ before $N$ reaches $D$ because $m$'s priority is too low and $N$ does not have sufficient buffers for it.

Suppose the next contact between $N$ and $D$ happens at time $t_c$, and $N$ has to drop $m$ due to buffer overflow at time $t_{over}$. Further denote the overall delivery probability by $P_{delivery}$. By definition the first and second dropping probability are given by $P\{t_{exp} \leq t_c\}$ and $P\{t_{over} \leq t_c\}$, respectively. Note that the temporal order of $t_c$ and $t_{exp}$ is determined by system parameters and the mobility pattern of $N$ and $D$, while the time of buffer overflow depends on $N$'s traffic load. Thus we can assume that the two dropping events are independent. Then we integrate them to get the delivery probability:

$$P_{delivery} = (1 - P\{t_{exp} \leq t_c\})(1 - P\{t_{over} \leq t_c\}) \quad (2)$$

In DTNs with unpredictable connectivity, when $N$ makes such estimation it is impossible to know the exact $t_c$, and thus it is impossible to compute the r.h.s of Eq. 2. So we have to make some approximations. When $t_{exp} > t_c$, $P\{t_{over} \leq t_c\} \leq P\{t_{over} \leq t_{exp}\}$ because the probability density function of $t_{over}$ is nonnegative. After inserting this inequation into Eq. 2, we get a conservative estimation:

$$P_{delivery} \geq (1 - P\{t_{exp} \leq t_c\})(1 - P\{t_{over} \leq t_{exp}\}) \quad (3)$$

The above estimation of $P_{delivery}$ can be seen as determined by two independent droppings, $P\{t_{exp} \leq t_c\}$ and $P\{t_{over} \leq t_{exp}\}$. The first one means that the packet expires before $N$'s next contact with $D$, so we call it *expiration dropping probability* and denote it by $P_{exp}$. The second one means that the packet overflows before expiration, so we call it *buffer overflow dropping probability* and denote it by $P_{over}$. Next, we discuss how to estimate them individually.

*2) Expiration Dropping Probability:* To estimate $P_{exp}$, we adopt an approach similar to that in [16]. Let random variable $X$ denote the inter-contact time between $N$ and the destination $D$. Assume that each inter-contact time is independent, then according to Markov's Inequality:

$$P_{exp} = P\{X > t_{exp} - \hat{t}\} \leq E(X)/(t_{exp} - \hat{t}) \quad (4)$$

where $E(X)$ is the mean of $X$ and $\hat{t}$ is the most recent contact time between $N$ and $D$ before the estimation time $t$. $E(X)$ can be approximated by the average of historical inter-contact times. The value of $P_{exp}$ should be bounded by 1. Eq. 4 intuitively means that nodes with a lower average inter-contact time (i.e., a higher contact frequency) with the destination have a lower expiration dropping probability.

*3) Buffer Overflow Dropping Probability:* The most important factor that affects $P_{over}$ is $m$'s priority value $p$ due to the buffer policy. Other two minor factors are the current empty buffer size $L_0$ and the residual time $t_r = t_{exp} - t$ before expiration. $L_0$ is positively related to how long $m$ can stay before being removed. But $t_r$ is negatively related: the longer $t_r$ is, the more likely it will be dropped due to buffer overflow.

Without clear knowledge of how these factors interact, it is extremely hard to theoretically model $P_{over}$. Therefore we turn to data mining techniques and model it as a supervised classification problem. Whenever $N$ drops or forwards a packet, it generates a record $< p, L_0, t_r, \beta >$. With data mining terminology, each record is called a *sample*, $p$, $L_0$, and $t_r$ are called *feature dimensions* and $\beta$ is called *class label*. $\beta = 1$ if $N$ drops the packet due to buffer overflow and $\beta = 0$ if $N$ does not drop it or drops it due to expiration.

Our basic heuristic is that the probability that $m$ will be dropped is similar to some historical packets which have similar feature values when they enter $N$'s buffer. Suppose we match $m$ to a set $\mathcal{S}$ of similar packets, and its dropped subset is $\mathcal{S}_{drop}$, then $P_{over}$ is estimated as:

$$P_{over} = |\mathcal{S}_{drop}|/|\mathcal{S}| \quad (5)$$

Figure 2(b) illustrates the idea in a two-dimensional space $< p, L_0 >$, where the historical packets in the dashed circle are the matched ones. In this example, the estimated $P_{over}$ of $m1$ and $m2$ are 0.83 and 0.25, respectively.

To match $m$ to similar packets, we choose the K-Nearest-Neighbor (KNN) [17] algorithm from the data mining literature, which identifies the $K$ packets that have the shortest distance to $m$ in the feature space. However, KNN traverses all samples during matching, which induces high online computation cost, and leaves less contact duration time for data transmission. Although some techniques have been proposed to improve its online matching time, they are too complex [18] to be applied to DTN nodes. To address this problem, we combine KNN with the Kcenter algorithm [19] to propose a two-phase solution:

- In the offline phase (when not in contact with others), nodes use the Kcenter algorithm to cluster samples into $\hat{K}$ clusters around $\hat{K}$ points in the feature space.
- In the online phase, nodes scan the $\hat{K}$ points in the increasing order of their distances with $m$'s feature vector until $K$ samples are included in the scanned clusters.

Based on previous work in this area [20], we set $K = \sqrt{n}$, where $n$ is the number of samples. We also set $\hat{K} = \sqrt{n}$. Simulations show that they perform well. Then the time and space complexity of offline clustering is $O(n\sqrt{n})$ and $O(n)$. The time complexity of online matching is $O(\sqrt{n}\log n)$, which is much smaller than $O(n\sqrt{n})$, the time complexity of naive KNN. To reduce the computation overhead, the offline

phase does not have to be run whenever a packet is dropped or forwarded; it can be run when a certain number of packets have been dropped or forwarded since the last run.

Both the online and offline phase need to compute the distance between two feature vectors. When doing so, $L_0$ is normalized to $[0, 1]$ based on the total buffer size of $N$, and $t_r$ is normalized to $[0, 1]$ based on the packet TTL. Moreover, since Euclidean distance performs poorly when samples are sparse in the feature space, we propose a distance metric by assigning different weights to different feature dimensions. We observe that if dropped samples spread narrowly in one feature dimension, this dimension is sensitive and should be highly weighted and vice versa. Suppose $m_1$ and $m_2$ are two feature vectors. Then their distance is:

$$D(m_1, m_2) = \sqrt{\sum_{i=1}^{3} \frac{\hat{\sigma_i}^2}{\sigma_i^2}(m_1^i - m_2^i)^2} \qquad (6)$$

where $\sigma_i^2$ and $\hat{\sigma_i}^2$ denote the variance of feature $i$ in dropped samples and all samples, respectively.

---

**Algorithm 1** : Greedy Algorithm for MKPAR, pseudo-code for $M$

---
1: Compute the selfish gain $g$ for any packet in $\mathcal{C}$
2: Sort $\mathcal{C}$ in the decreasing order of $g/l$ (Let $i$ denote the $i^{th}$ packet in $\mathcal{C}$)
3: **for** Packet $i$ from 1 to $|\mathcal{C}|$ **do**
4:    **if** $N$ is not in contact with $M$ anymore **then**
5:       **break**
6:    **end if**
7:    **if** $L_i \geq l_i$ **then**
8:       Forward $i$ to $N$
9:       **for** Packet $j$ from $i+1$ to $|\mathcal{C}|$ **do**
10:          $L_j - = l_i$
11:       **end for**
12:    **else**
13:       **continue**
14:    **end if**
15: **end for**

---

### C. Forwarding Set Optimization

In this subsection, we solve the following problem: suppose a node $M$ contacts $N$, and $M$ has determined a candidate packet set $\mathcal{C}$ for which $N$ has higher delivery probabilities. Since $N$'s buffer may be inadequate to accept all packets in $\mathcal{C}$, and the contact duration may be too short to transmit all these packets, how to determine a subset of $\mathcal{C}$ to transmit and in what order?

We follow two principles. First, $M$ will not forward a packet to $N$ if $N$ does not have sufficient buffers for that packet. According to the buffer management rule, $N$'s available buffer size $L_m$ for $m$ is:

$$L_m = L_0 + \sum_{\{k|p_k<p\}} l_k \qquad (7)$$

where $L_0$ denotes $N$'s empty buffer size, $\{k|p_k < p\}$ denotes the packets in $N$'s buffer whose priority is smaller than that of $m$ ($p$), and $l_k$ denotes the size of packet $k$.

Second, $M$ tries to maximize its selfish gain through this contact, which is defined as follows.

**Definition 1 (Selfish Gain)**    The selfish gain $g$ that $M$ achieves by forwarding $m$ to $N$ is the product of $m$'s priority $p$ in $M$ and the increment of delivery probability, i.e., $g = p \cdot \Delta P_{delivery}$.

Both factors in the definition are related to selfishness. $p$ means how socially important the packet is. The larger $p$ is, the more selfishness is gained. $\Delta P_{delivery}$ means how much this forwarding can increase the packet's probability to be delivered. The larger $\Delta P_{delivery}$ is, the more help is provided. So their product is a natural representation of the gained selfishness.

Suppose all the packets in $\mathcal{C}$ are sorted by priority in the increasing order, then we can simply use $i$ to denote the $i^{th}$ packet. Let $X_i$ denote if packet $i$ is selected by the to be transmitted subset ($X_i = 1$) or not ($X_i = 0$). According to the above two principles, the problem can be formulated as:

$$max \quad \sum_{i \in \mathcal{C}} g_i X_i \quad s.t. \quad \forall i \quad \sum_{j \leq i} X_j l_j \leq L_i \qquad (8)$$

Next we convert it into an MKPAR formulation [21], where each item can only be assigned to a subset of the knapsacks. Suppose the original buffer is divided into $|\mathcal{C}| + 1$ knapsacks such that the first knapsack has size $\mathbb{S}_1 = L_1$, the $j^{th}$ ($j \in \{2, ..., |\mathcal{C}|\}$) one has size $\mathbb{S}_j = L_j - L_{j-1}$, and the $(|\mathcal{C}| + 1)^{th}$ one consists of buffers that cannot be preempted by any packet in $\mathcal{C}$. Then packet $i$ can only be packed into knapsacks indexed smaller than or equal to $i$. Let $X_{ij}$ denote if packet $i$ is packed into knapsack $j$ ($X_{ij} = 1$) or not ($X_{ij} = 0$), then $X_{ij} = 0$ when $i < j$. Eq. 8 can be rewritten as an MKPAR:

$$
\begin{aligned}
max \quad & \sum_{i=1}^{|\mathcal{C}|}\sum_{j=1}^{|\mathcal{C}|} g_i X_{ij} \\
s.t. \ \forall i \ & \sum_j X_{ij} \leq 1, \quad \forall j \ \sum_i X_{ij}l_i \leq \mathbb{S}_j
\end{aligned}
\qquad (9)
$$

Since a simpler variation of MKPAR has been proved by Dawande et al. [21] to be NP-hard, MKPAR is also NP-hard. Thus, we give a greedy algorithm, which ranks the packets in the decreasing order of selfish gain weighted by packet size, and packs them one by one until no more packets can be packed. The details are shown in Algorithm 1. The time complexity of this algorithm is $O(|\mathcal{C}|^2)$, which is acceptable because most handsets have such computing capability.

## IV. PERFORMANCE EVALUATIONS

In this section, we evaluate the performance of SSAR and compare it to other existing routing algorithms.

### A. Experiment Setup

The evaluation is based on the MIT Reality trace [22], [23], on which a recent work [24] has validated the existence of the so-called "small-world" phenomenon, a well-known phenomenon in social networks. In this trace, 97 Nokia 6600 smart phones were carried by students and staff at MIT over nine months. These phones run Bluetooth device discovery every five minutes and log about 110 thousand contacts with

each other. Each logged contact includes the two contact parties, the start-time, and the duration. We use the contacts to generate trace-driven simulations.

Since the trace does not have the accurate social relationship information among participants, we need to construct a weighted directed social graph upon them. To better study SSAR, we evaluate it on two types of social graphs.

The first type of social graph is probabilistically contact-dependent. It can be built based on the following heuristic, which has been verified by many sociology studies [12]. The stronger tie two individuals have, the more likely they contact frequently. Individuals with more social ties are more likely to meet other people. Let $f_*$ denote the overall contact frequency of the whole trace, $f_N$ denote node $N$'s overall contact frequency, and $f_{NM}$ denote the contact frequency between $N$ and $M$. The graph is constructed in four steps:

1) We generate power-law distributed node degrees based on several measurement studies [25].
2) We repeatedly assign those degrees to nodes in the trace, i.e., assign the largest degree to a node in such a way that node $N$'s probability to be selected is $f_N/f_*$, and repeat this for the remaining degrees and nodes.
3) We generate weights for the social ties (edges) of each node. The best empirical data we can find about social tie strength is from one recent study in which participants rate their friendship nearly uniformly between 0 and 1 [13]. Thus, we generate weights for each node's social ties that are uniformly distributed within [0,1].
4) For each node $N$, we connect its ties to other nodes. We connect the strongest tie to another node in a way that node $M$'s probability to be connected is $f_{NM}/f_N$, and repeat this for the other ties and not-connected nodes. In the end, for any ordered node pair $NM$ that has not been connected yet, the weight of edge $\overrightarrow{NM}$ is set 0.

The second type is random social graph that is also constructed in four steps. The first and third steps are the same, but in the second and fourth steps we assign degrees to random nodes, assign weights to random social ties, and connect social ties to random nodes.

One important feature of a social network is the average number of social ties per node; i.e., the number of nodes with a social tie strength larger than 0. In some networks, each node only has a few social ties; while in others, each node has many social ties. To generate social graphs with different average numbers of social ties per node, we fix the power-law coefficient at 1.76 [25] when generating node degrees, but change the minimum acceptable degree.

In the simulation, each node generates one packet every day to random destinations. The packet size is uniformly distributed from 50 KB to 100 KB. Each packet has a certain TTL and will be removed after the TTL expires. All packets have initial priority 1. Each node has buffer size 5MB and bandwidth 2Mbps. In each run, the first 1/3 of the trace is used for warm-up, and the results are collected from the remaining part. To avoid end-effects, no packet is generated in the last 1/3 of each trace. All results are averaged over 10 runs.

## B. Routing Algorithms and Metrics

*1) Routing Algorithms:* We compare SSAR with two other benchmark algorithms, PROPHET [2] and SimBet [7]. PROPHET is a standard non-oblivious benchmark that has been used to compare against several previous works [6]. It calculates a metric, delivery predictability, based on contact histories, and relays a packet to a node with higher delivery predictability. We use the same parameters as in [2], and age the delivery predictability upon every contact as done in [6]. SimBet has also been used as a benchmark in several works [3]. It calculates a simbet metric using two social measures (similarity and betweenness). A packet is forwarded to a node if that node has higher simbet metric than the current one. We use the same parameters as in [7].

Since the original algorithms do not define the order of packets to be transmitted during a contact, we adopt the transmission order used in RAPID [5]. Because this order has been shown to be the most effective, we believe such refinement does not favor SSAR in comparison. Since the original algorithm either assumes infinite buffer (SimBet) or assumes finite buffer but does not specify the packet dropping policy (PROPHET), we apply three policies (drop-tail, random drop, and minimum-utility-drop-first) in simulation, and only present the results of the best policy here, i.e., minimum-utility-drop-first. Since it is impossible to traverse all dropping policies and choose the optimal one, we tried our best to impose the minimum influence on the original algorithms.

PROPHET and SimBet are designed without considering social selfishness. For fair comparison, we modified them to be selfishness-aware. That is, nodes do not forward packets to others who are not willing to forward for them, and avoid immediate droppings caused by selfishness. However, when nodes forward packets to others who are willing to forward for them, they still follow the aforementioned transmission order and buffer policy. To show the effect of such selfishness-awareness, we also include the basic PROPHET in our simulations. For convenience, we label the selfishness-aware PROPHET *PROPHET-1*, and label the basic one *PROPHET-2*.

*2) Metrics:* We use the following metrics to evaluate these algorithms: packet delivery ratio, the total number of transmissions, and selfishness satisfaction (SS). Packet delivery ratio is defined as the proportion of packets that are delivered to their destinations out of the total unique packets generated. The total number of transmissions can be used as a cost factor [7], and fewer transmissions mean lower cost. SS is defined as the ratio of the average priority of all forwarded or delivered packets over the average priority of all dropped packets. SS reflects how much users are satisfied with the network, because a larger SS indicates more important messages are served. SSAR, PROPHET, and SimBet are expected to have similar cost since all of them have only one replica of a packet.

## C. Results

*1) Performance and Cost:* **The Effects of TTL** We change the packet TTL from 0 to 125 days to see its effects on the packet delivery ratio and the cost. Each node has 25 social ties
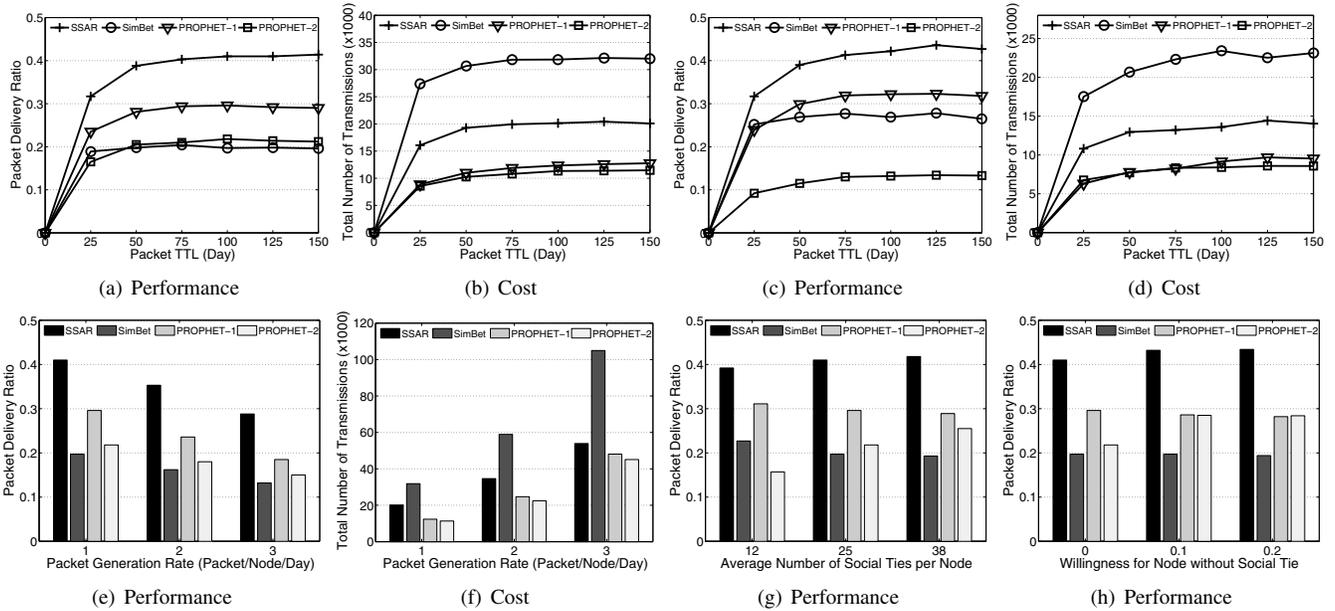
Fig. 3. Comparison of performance and cost. For (e)-(h), the packet TTL is 100 days. (a)(b) Results under the contact-dependent social graph. (c)(d) Results under the random social graph. (e)(f) Results under different workloads. (g) Performance v.s. the number of social ties. (h) Performance v.s. willingness for those without social ties.

on average. Figure 3(a) shows the packet delivery ratio under the contact-dependent social graph. As the TTL increases, all algorithms can deliver more packets to the destinations. However, the delivery ratios will not increase after the TTL reaches a certain value, e.g., 25 days in SimBet and 50 days for the other three algorithms. This is because, after the TTL reaches some value, the forwarding capacity of the network becomes the performance bottleneck.

Among all algorithms, SSAR has the highest packet delivery ratio. It outperforms PROPHET-1 by 35%-40%, and outperforms SimBet and PROPHET-2 by 100%. This is because SSAR incorporates several factors such as user willingness, buffer constraint, and contact opportunity into relay selection. It avoids low-willingness nodes or overloaded hot spots, and avoids being dropped by these nodes. But in SimBet, since it tends to swarm packets to hot spot nodes (with higher centrality), it has the worst delivery ratio due to congestion. In SSAR, its MKPAR formulation does not forward a packet to the receiver whose buffer is insufficient. However, the other three algorithms cannot achieve this, resulting in many packet drops due to buffer overflow. As expected, PROPHET-1 significantly outperforms PROPHET-2 by about 50%, because PROPHET-2 forwards many packets to those who are unwilling to forward, and those packets are dropped.

Figure 3(b) shows the cost under the contact-dependent social graph. As the TTL increases, all algorithms have more transmissions, because packets stay longer in the network and have more opportunities to be transmitted. The total number of transmissions does not increase too much after the TTL reaches some value (about 75 days), when the number of transmissions is limited by the contact opportunities of the trace. Among the algorithms, SimBet has the most number of transmissions, which is about 60% more than that SSAR and 150% more than PROPHET-1 and PROPHET-2. Interest-

ingly, PROPHET-2 has similar cost to PROPHET-1. Though PROPHET-2 uses more contacts to forward packets, its packets traverse fewer hops than PROPHET-1 before being dropped.

Figure 3(c) and Figure 3(d) show the results under the random social graph. Similar with the results under the contact-dependent graph, SSAR delivers the most number of packets to the destinations. It outperforms PROPHET-2, SimBet, and PROPHET-1 by 200%, 60%, and 30%, respectively. SimBet has the most number of transmissions, about 60% more than SSAR and 150% more than PROPHET-1 and PROPHET-2.

Based on the results under the two types of social graphs, we found that SimBet delivers 35% more packets under the random social graph than that under the contact-dependent graph, and PROPHET-2 delivers 35% less packets. This is because in the latter graph a socially popular node is very likely to be a hot spot node in contact, while in the former graph such probability is much lower. Since SimBet tends to forward packets to socially popular nodes, it overloads more hot sport nodes in the contact-dependent social graph. PROPHET-2 forwards packets out no matter if the contacted node is socially tied or not. But in the random social graph the contacted node is more likely to be the ones without social ties, resulting in low packet delivery ratio. Both SSAR and PROPHET-1 deliver similar amounts of packets under the two types of social graphs. In the random social graph, the three selfishness-aware algorithms have fewer packet transmissions because fewer contacts occur between two socially tied nodes.

Since SSAR has similar performances under both social graphs, in the following, we only present the results under the contact-dependent social graph.

**The Effects of Workload** To evaluate the performance of SSAR under higher workloads, we change the packet generation rate from 1 packet per node per day to 3 packets per node per day. Each node on average has 25 social ties. Figure

3(e) and Figure 3(f) show the results. When the workload increases, all algorithms have lower packet delivery ratios and more transmissions. However, they change at different rates, especially SSAR and PROPHET-1. When the packet generation rate is 1 packet per node per day, SSAR delivers 40% more packets than PROPHET-1 with 60% more transmissions. When the rate increases to 3 packets per node per day, SSAR delivers 60% more packets than the latter with only 10% more transmissions. This means that SSAR is more efficient under high workloads.

**The Effects of the Average Number of Social Ties per Node** The packet delivery ratios of the algorithms are shown in Figure 3(g). As the average number of social ties per node increases from 12 to 38, PROPHET-2's packet delivery ratio increases from 16% to 26%, because fewer packets are forwarded to nodes without social ties. Such significant increase does not exist in the other three selfishness-aware algorithms. The packet delivery ratio of SimBet and PROPHET-1 even drops a little bit. The reason is as follows. With the contact-dependent social model, social ties are more likely to be added to nodes with frequent contacts first, and then to nodes with less frequent contacts. As a result, most later-added nodes contact each other less frequently, and the network's contact opportunity does not increase too much. Moreover, the extra data traffic due to the new social ties may overload the existing hot spots, affecting the packet delivery ratio negatively. Despite all these issues, SSAR still manages to deliver some more packets, because it makes more balanced use of social ties considering their contact opportunity, willingness, and buffer constraint. In summary, SSAR outperforms the other three algorithms when nodes have various numbers of social ties.

**The Effects of Willingness to Forward for Nodes without Social Ties** In previous simulations, a node's willingness to forward for others without social ties has been set as 0. In some networks, a generous user may be willing to forward packets for those who have no social tie with him, though the willingness is lower than that for those with social ties. To evaluate SSAR under such environments, we set a small weight for nodes without social ties to forward for others, and generate higher weights for nodes with social ties. In this case, each node can be seen as having a social tie with every other node, and PROPHET-2 becomes identical to PROPHET-1. The results are shown in Figure 3(h). As the willingness for nodes without social ties changes from 0 to 0.1 and 0.2, the three selfishness-aware algorithms do not change much in packet delivery ratio, but PROPHET-2 delivers more packets.

*2) Allowed Selfishness:* One key feature of SSAR is that it allows users to be socially selfish. To compare SSAR with other algorithms on how much selfishness is allowed, we plot the SS metric in Figure 4. The packet TTL is 25 days, and each node on average has 25 social ties.

SSAR allows better selfishness than the other three algorithms. Specifically, SS in SSAR is one magnitude larger than that of the other three algorithms. SSAR allows more selfishness for two reasons. First, its buffer management policy satisfies social selfishness. Second, because of the MKPAR
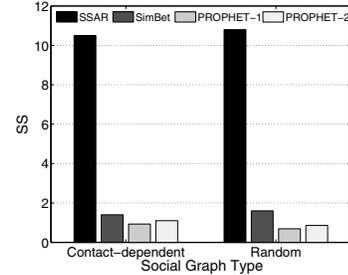


Fig. 4. Comparison of allowed selfishness.

formulation, high-priority packets are more likely to be forwarded than low-priority ones, as determined by the selfish gain metric defined in Section III-C. In contrast, the other three algorithms manage buffers without selfishness information and forward packets purely based on contact opportunity, so they perform much worse.

## V. DISCUSSION

Though the current design of SSAR does not consider malicious behaviors, it can tolerate many of them. Suppose a malicious attacker joins the network and launches flooding-based DoS attacks. If other network members do not trust this attacker and assign 0 or very low social tie strength to the attacker, the flooding traffic will not be allocated any buffer/bandwidth resource or only allocated very few resources to mitigate the DoS attack. If the attacker launches a blackhole attack (i.e., drop all received packets), SSAR can still tolerate it with the least modification that a node never forwards packets to those without social ties from its point of view. Therefore, the blackhole attack also fails. In summary, SSAR can deal with outside attackers.

For an inside attacker which is already known by some existing members, it may successfully launch flooding-based DoS or blackhole attacks in the short run. However, the effect of such attacks is limited by the number of social ties the attacker has, and will be weakened beyond one hop. Furthermore, when such behaviors are detected, the attacker may loose social ties with others, rendering its future attacks impossible. Certainly, this does not mean SSAR should not take care of such behaviors, but we believe such insider attacks deserve a separate study. Issues like how to determine legitimate or malicious packet droppings in DTN nodes are different from those in mobile ad hoc networks where people often make such decisions based on wireless channel reliability. We leave it as our future work.

We note that misreporting behaviors are possible in SSAR. For example, a node may set higher priority values for its buffered packets than they should have to make them more competitive in the next hop. To prevent such behaviors, we can apply ideas similar to rate limiting. For example, a node may limit the number of high-priority packets or the total priority from a neighbor; when the actual received number is over the limit, it will randomly degrade some of them. We will look into this problem in our future work.

## VI. Related Work

### A. DTN Routing

We review existing works along three lines.

*Forwarding decision* When a node contacts another, it needs to determine which packets to forward. Existing algorithms vary in what information is used to evaluate a node's forwarding capability and make forwarding decisions. Earlier works [2], [4] evaluate the forwarding capability of a node by the historic contact information. Algorithms [26], [27] have also been proposed for finding the right relays for data forwarding in vehicular ad hoc networks. Recently, several algorithms [6], [7], [28], [29] use social metrics calculated from contacts. These approaches evaluate the forwarding capability of a node purely based on its contact opportunity. However, we consider contact opportunity, social willingness and buffer constraint in an integrated way.

*Forwarding set and order* Since a contact may be too short to transmit all packets, it is important to determine in what order to forward the packets. Existing algorithms usually decide a greedy transmission sequence by weighting some metric (like RAPID [5]), and transmit the packets one by one until the contact ends or no packet is left. However, these approaches ignore that the forwarded packets may be immediately dropped by the receiver due to buffer constraint. So attention should also be given to what to forward. Note that deciding the forwarding packet set is different from deciding if a neighbor is a better relay for a packet. SSAR uses MKPAR to determine both the forwarding set and the order, which is different from the MKP formulation in [28].

*Buffer management* Balasubramanian et al [5] give a good survey on this topic. Most routing protocols assume unlimited buffer (e.g., delegation forwarding [3]), which is unrealistic especially when the traffic load is high and replication is used. Others consider buffer constraints [4], [5]. However, existing works only discuss the packet drop policy (e.g., [4]). SSAR integrates the buffer constraint into forwarding decision and forwarding set optimization.

### B. Individual Selfishness

Individual selfishness has been widely studied in mobile ad hoc networks [10], [11] and even in DTNs [16], [30]. The solutions proposed so far fall into two categories, credit-based approaches (e.g., [11]) and reputation-based approaches (e.g., [10]). The principle idea is to stimulate users to forward packets for others. As discussed in Section II-A, they cannot be directly applied to the social selfishness problem.

## VII. Conclusion

This paper introduces the social selfishness problem into DTNs and proposes a routing algorithm SSAR following the philosophy of *design for user*. SSAR allows user selfishness and improves performance by considering user willingness, resource constraints, and contact opportunity when selecting relays. Extensive simulations on the MIT Reality trace show that SSAR can maintain social selfishness and achieve a very good routing performance in an efficient way.

## References

[1] K. Fall, "A delay-tolerant network architecture for challenged internets," *Proc. SIGCOMM*, pp. 27–34, 2003.

[2] A. Lindgren, A. Doria, and O. Schelen, "Probabilistic routing in intermittently connected networks," *ACM SIGMOBILE CCR*, vol. 7, no. 3, pp. 19–20, 2003.

[3] V. Erramilli, A. Chaintreau, M. Crovella, and C. Diot, "Delegation Forwarding," *Proc. MobiHoc*, 2008.

[4] J. Burgess, B. Gallagher, D. Jensen, and B. Levine, "Maxprop: Routing for vehicle-based disruption-tolerant networks," *Proc. INFOCOM*, 2006.

[5] A. Balasubramanian, B. N. Levine, and A. Venkataramani, "Dtn routing as a resource allocation problem," *Proc. ACM SIGCOMM*, 2007.

[6] P. Hui, J. Crowcroft, and E. Yoneki, "Bubble rap: social-based forwarding in delay tolerant networks," *Proc. MobiHoc*, pp. 241–250, 2008.

[7] E. Daly and M. Haahr, "Social network analysis for routing in disconnected delay-tolerant MANETs," *Proc. MobiHoc*, pp. 32–40, 2007.

[8] M. Motani, V. Srinivasan, and P. Nuggehalli, "PeopleNet: engineering a wireless virtual social network," *Proc. MobiCom*, pp. 243–257, 2005.

[9] P. Hui, A. Chaintreau, J. Scott, R. Gass, J. Crowcroft, and C. Diot, "Pocket switched networks and human mobility in conference environments," *SIGCOMM Workshops*, 2005.

[10] J. J. Jaramillo and R. Srikant, "Darwin: Distributed and adaptive reputation mechanism for wireless ad-hoc networks," *Proc. MobiCom*, 2007.

[11] S. Zhong, J. Chen, and Y. R. Yang, "Sprite: A simple, cheat-proof, credit-based system for mobile ad-hoc networks," *Proc. IEEE INFOCOM*, vol. 3, pp. 1987–1997, 2003.

[12] M. Granovetter, "The strength of weak ties," *The American Journal of Sociology*, vol. 78, no. 6, 1973.

[13] E. Gilbert and K. Karahalios, "Predicting tie strength with social media," *Proc. CHI*, 2009.

[14] A. Seth and S. Keshav, "Practical security for disconnected nodes," *First IEEE ICNP Workshop on Secure Network Protocols (NPSec)*, 2005.

[15] A. Josang and S. Pope, "Semantic constraints for trust tansitivity," *Proceedings of the Asia-Pacific Conference of Conceptual Modelling (APCCM) (Volume 43 of Conferences in Research and Practice in Information Technology)*, 2005.

[16] U. Shevade, H. Song, L. Qiu, and Y. Zhang, "Incentive-aware routing in dtns," *IEEE ICNP*, 2008.

[17] G. Mclachlan, *Discriminant Analysis and Statistical Pattern Recognition*. Wiley, 1992.

[18] J. H. Friedman, J. L. Bentler, and R. A. Finkel, "An algorithm for finding best matches in logrithmic expected time," *ACM Transactions on Mathimatical Software*, vol. 3, no. 3, pp. 209–226, 1977.

[19] T. Gonzalez, "Clustering to minimize the maximum inter-cluster distance," *Theoret. Comput. Sci.*, vol. 38, pp. 293–306, 1985.

[20] D. . Loftsgaarden and C. P. Quesenberry, "A nonparametric estimate of a multivariate density function," *Ann. Math. Statist.*, vol. 36, 1965.

[21] M. Dawande, J. Kalagnanam, P. Keskinocak, R. Ravi, and F. Salman, "Approximation algorithms for the multiple knapsack problem with assignment restrictions," *Journal of Combinatorial Optimization*, vol. 4, pp. 171–186, 2000.

[22] N. Eagle and A. Pentland, "Reality mining: sensing complex social systems," *Personal and Ubiquitous Computing*, vol. 10, no. 4, pp. 255–268, 2006.

[23] A. community resource for archiving wireless data at Dartmouth, "http://crawdad.cs.dartmouth.edu/data.php."

[24] A. Chaintreau, A. Mtibaa, L. Massoulie, and C. Diot, "The diameter of opportunistic mobile networks," *Proc. ACM CoNEXT*, 2007.

[25] A. Mislove, M. Marcon, K. P. Gummadi, Druschel, and Bhattacharjee, "Measurement and analysis of online social networks," *Proc. IMC*, 2007.

[26] J. Zhao and G. Cao, "Vadd: Vehicle-assisted data delivery in vehicular ad hoc networks," *Proc. IEEE INFOCOM*, 2006.

[27] Y. Zhang, J. Zhao, and G. Cao, "Roadcast: A popularity aware content sharing scheme in vanets," *IEEE International Conference on Distributed Computing Systems (ICDCS)*, 2009.

[28] C. Boldrini, M. Conti, and A. Passarella, "Contentplace: Social-aware data dissemination in opportunistic networks," *Proc. MSWiM*, 2008.

[29] W. Gao, Q. Li, B. Zhao, and G. Cao, "Multicasting in delay tolerant networks: A social network perspective," *Proc. ACM MobiHoc*, 2009.

[30] F. Li, A. Srinivasan, and J. Wu, "Thwarting blackhole attacks in distruption-tolerant networks using encounter tickets," *Proc. IEEE INFOCOM*, 2009.