A Routing Protocol for Socially Selfish Delay Tolerant Networks

Qinghua Li, Wei Gao, Sencun Zhu, Guohong Cao

*Department of Computer Science & Engineering
The Pennsylvania State University, University Park
Email: {qxl118, wxg139, 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 cope with user selfishness and provide good routing performance in an efficient way. To select an effective forwarding node, SSAR considers both users' willingness to forward and their contact opportunity, and derives a metric with mathematical modeling and machine learning techniques to measure the forwarding capability of the mobile nodes. 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 good routing performance with low transmission cost.

Keywords: DTN, Routing, Social Selfishness

1. Introduction

Delay Tolerant Networks (DTNs) [2, 3, 4] enable data transfer when mobile nodes are only intermittently connected. Due to lack of consistent connectivity, DTN routing usually follows store-carry-and-forward; i.e., after receiving some packets, a node carries them around until it contacts another node and then forwards the packets. Since DTN routing relies on mobile nodes to forward packets for each other, the routing performance (e.g., the number of packets delivered to their destinations) depends on if nodes are willing to forward for others.

In the real world, most people are *socially selfish*. As being social, they are willing to forward packets for others

with whom they have social ties¹ such as family members and friends even at the cost of their own resources. Also, they give different preferences to those with social ties, i.e., they will provide better service to those with stronger ties than to those with weaker ties, especially when there are resource constraints. As being selfish, they are unwilling to forward packets for those with whom they have no social ties in order to save their own storage and power resources. For convenience, the above social and selfish behavior will be referred to as *social selfishness*.

As far as we know, social selfishness has not been addressed before. Although many routing algorithms [6, 7, 8, 9, 10, 11] have been proposed for DTNs, most of them do not consider users' willingness and implicitly assume that a node is willing to forward packets for all others. They may not work well since some packets are forwarded to nodes unwilling to relay, and will be dropped. A few recent studies [12, 13] have considered the selfish side of users, where selfish nodes are stimulated to forward packets for all other nodes to maintain high performance. However, these schemes go to another extreme; i.e., they assume that a node is not willing to forward packets for anyone else. For convenience, such selfishness is called *individual selfishness*.

In this paper, we aim to answer the following question: *How to design a routing protocol for DTNs composed of socially selfish users?* From the network's point of view, the protocol should force every node to forward packets for all others, since this can achieve the highest network performance. However, this solution does not consider the users' willingness and users cannot behave as they are willing to. Different from it, our philosophy is "design for user", i.e., we take social selfishness as a user demand and allow nodes to be socially selfish. Following this philosophy, we propose a Social Selfishness Aware Routing (SSAR) protocol, in which a node only forwards packets for those with social ties, and it gives priority to packets received from those with stronger social ties when there are not enough resources.

Since each node only forwards packets for part of the nodes, it is important to know how this will affect the routing performance. To achieve high performance, SSAR considers both user willingness and contact opportunity when selecting relays. It combines the two factors through mathematical modeling and machine learning techniques, and obtains a new metric to measure the forwarding capability of a relay. With SSAR, a packet will most likely be forwarded to the relay that has strong willingness to forward as well as high direct or indirect/transitive contact oppor-

¹In this paper, a social tie means an interpersonal tie that falls into the *strong* or *weak* category defined by Granovetter [5].

tunity with the destination. To further improve performance, SSAR formulates the forwarding process as a Multiple Knapsack Problem with Assignment Restrictions (MKPAR). It provides a heuristic-based solution that forwards the most effective packets for social selfishness and routing performance. Extensive trace-driven simulations show that SSAR can achieve good routing performance with low transmission cost.

The remainder of this paper is structured as follows. Section 2 presents an overview of SSAR. Section 3 gives the detailed design. Section 4 introduces the trace-driven simulations and discusses the results. The last two sections present related work and conclusions, respectively.

2. SSAR Overview

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

2.1. Design for User

Existing work in mobile ad hoc networks and DTNs has focused on addressing individual selfishness using reputation-based [30], credit-based [14], or game-theory based [12] approaches to stimulate users to cooperate and forward packets for others. If the nodes cooperate with others, they will be able to get help from others; if not, they will be punished, e.g., being deprived of access to the network.

These incentive-based schemes may not be directly applied to deal with social selfishness, since they do not consider social selfishness. In these schemes, every node has to provide service to others no matter there is a social tie or not. Thus, social selfishness is not allowed. In essence, these approaches follow the philosophy of “design for network” because they sacrifice the user’s requirement for selfishness (i.e., resource saving) to achieve high performance.

We address this problem from a different point of view. We allow social selfishness but also try to maintain good routing performance under 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 new design dimension that measures the user satisfaction, similar to other traditional dimensions such as performance. Such design philosophy is referred to as “design for user”.

2.2. Network Model

In DTNs, nodes have limited bandwidth and computational capability. As in other studies [7], we assume each node has unlimited buffer space for its own packets, but limited buffer space for packets received from other nodes. We also assume each packet has a certain lifetime denoted by TTL. A packet becomes useless and should be dropped after it expires. We further assume bidirectional links, which can be provided by some MAC layer protocols, e.g., IEEE 802.11.

2.3. Willingness Table

Each node maintains a table that contains its willingness values for other nodes in the network. In this table, each item has the format $\langle nodeID, value \rangle$. 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. A node's willingness value for another node depends on the social tie between them. The stronger the social tie is, the larger the willingness is.

Each user manages its willingness table in an off-line style through some user interface provided by the mobile device (e.g., the keypad of a smart phone). Note that a user does not need to know all other users in the network. She only needs to set her willingness value via the user interface for each other user with whom she has a social tie, and set a default willingness value (e.g., 0) for all other (possibly unknown) users without any social tie. Thus, most likely, the willingness table has one item for each node with a social tie and an additional item for all other nodes. A user only needs to manually configure its willingness table when she joins the network or migrates to a new mobile device, and update the table when she has new social ties or her old social ties have changed.

Sociological studies have found that the number of social ties a human being may have is only a few hundred (150 and 290 according to Dunbar [37] and McCarty et al. [38], resp.) and social ties are usually stable over time [36]. This means that the update of willingness table is quite infrequent. Thus, only a low manual intervention from the user is required, and the usability will not be affected much. Actually, the maintenance of willingness table is comparable to the maintenance of address book in a cell phone, which can be well handled by users.

Here, one concern is whether users can quantify the strength of their social ties into the range of $[0, 1]$. This has been shown to be feasible by a recent study which requires users to quantitatively rate their friendships [15]. To be more user-friendly, the user interface can even provide several preset willingness levels for the user to choose from,

Figure 1: SSAR architecture and an example contact between node N and M. The dashed rectangles enclose the information exchanged in step 2 and step 3 (Sec. 2.5).

and convert each chosen willingness level into numerical values in the background. For example, with six preset willingness levels “very strong”, “strong”, “average”, “weak”, “very weak” and “none”, they can be converted into 1.0, 0.8, 0.6, 0.4, 0.2 and 0, respectively.

2.4. The Architecture

Figure 1 shows the architecture of SSAR. In the following we introduce the components and their functions.

Packet priority manager: A node assigns a priority between 0 and 1 to each buffered packet based on its willingness for the source node and the previous hop. The priority of a packet measures the social importance of this packet for this node. More details on priority calculation will be given in Sec. 3.1.

Buffer manager: A node 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. The second rule indicates that a new incoming packet can preempt the buffer occupied by a lower-priority packet. The buffer policy together with the priority assignment method allows nodes to be socially selfish (see Sec. 3.1).

Delivery probability estimator: It estimates a node’s *delivery probability* for a packet, which is used to measure the node’s forwarding capability for that packet. When two nodes are in contact, each packet is forwarded from the node with a lower delivery probability to the node with a higher delivery probability.

Traditionally, the quality of a relay is measured solely based on contact opportunity, which can be the relay’s direct contact opportunity to the destination node or the transitive contact opportunity provided by the relay’s contacted nodes

Figure 2: An example of willingness-aware forwarding.

or both. 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. Suppose S has a packet $m1$ to send to D , and its contact opportunity with D within the packet lifetime is low (0.3). Suppose S successively meets A , C , and B . If only contact opportunity is considered, it will forward $m1$ to A whose contact opportunity is 0.9. Unfortunately, A will drop $m1$ since it is unwilling to forward for S (the willingness is 0). SSAR will avoid such forwarding. Although C has a higher contact opportunity and it is willing to forward $m1$ for S , its willingness is so low that $m1$ may suffer a high risk of being dropped, so SSAR will also 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.

Our approach to deriving delivery probability includes the mathematical modeling of the probability that a packet will be dropped due to expiration with Markov's Inequality and the estimation of the probability that the packet will be dropped due to buffer overflow with a machine learning technique that combines two known classification algorithms from the literature. More details will be provided in Section 3.2.

Forwarding set manager: After a node determines a set of packets that should be forwarded to a better relay, existing protocols (e.g., [9]) greedily transmit them no matter the receiver has enough buffer to hold these packets or not. 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 packet to transmit by solving an MKPAR (Multiple Knapsack Problem with Assignment Restrictions). It considers the buffer constraint and transmits the packets that are the most effective for social selfishness and routing performance.

2.5. The Protocol

We use an example (Fig. 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.
2. Suppose M still buffers some other packets. Then M sends N a summary list of $\langle \text{source ID, destination ID, expiration time, priority} \rangle$ for these packets.
3. From the source ID and priority information, N calculates the new priority value for each packet in the list (Sec. 3.1). Based on the new priority, the destination ID and expiration time, N calculates its delivery probability (Sec. 3.2) and available buffer size (Sec. 3.3) for each packet, 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 (Sec. 3.3) formulation.

In Step 3, if the new priority of a packet is zero, N 's delivery probability and available buffer size for it are also zero. In this case, N does not need to go through the procedures in Sec. 3.2 and 3.3. Without loss of generality, in the last four steps we only describe how node M determines which packets to transfer to N . Node N does so in similar ways.

Sometimes 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 interacts with its neighbors one by one.

2.6. Forwarding Strategy

SSAR can work in two modes, *forwarding mode* and *replication mode*. In the forwarding mode, a node deletes its own copy after it transmits a packet to its neighbor (which has a higher delivery probability). Thus, any packet can simultaneously have at most one copy in the network. In the replication mode, however, the node keeps its own copy after transmitting the packet. Therefore, the packet may have many replicas in the network. The number of replicas depends on the mobility pattern and is non-deterministic. Generally speaking, the replication mode can deliver more packets than the forwarding mode but it also requires more resources such as buffer and bandwidth. Which mode to use should be application-specific.

3. Detailed Design

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

3.1. Packet Priority

When a node receives a packet from a previous hop, it assigns a priority to the packet. The priority determines if this node will relay the packet (i.e., the priority is positive) or not (i.e., the priority is zero). To be socially selfish, the node only forwards the packet if it is from a node with a social tie. There are two cases. First, the source of the packet has a social tie with this node, and hence forwarding the packet means helping the source. Second, the previous hop has a social tie with this node, no matter the source has a social tie or not. In this case, the previous hop has taken over the responsibility (probably from its own social tie) to deliver the packet. Thus, even if the source does not have a social tie with this node, this node should still relay the packet to help the previous hop. Actually, this is motivated by the real-world phenomenon that people usually would like to help a friend's friend. The priority should also measure the social importance of the packet to this node. For example, when other conditions are the same, packets from the node with a stronger social tie should have a higher priority.

Let p_{curr} denote the new priority of a packet in the current hop, and p_{prev} denote the old priority of the packet in its previous hop. Let ω_{src} and ω_{prev} denote the current hop's willingness for the packet source and the previous hop, respectively. Then the current hop calculates the new priority in the following ways (Note that the initial priority of a packet is set as 1 by the source node.): (1) If neither the source nor the previous hop has a social tie with the current hop, then $p_{curr} = 0$. (2) If the source has a social tie but the previous hop does not, then $p_{curr} = \omega_{src}$. (3) If the previous hop has a social tie but the source does not, then $p_{curr} = p_{prev} \cdot \omega_{prev}$. This calculation method borrows the idea of transitive trust [16] from the reputation system literature. (4) If both the source and the previous hop have a social tie with the current hop, $p_{curr} = \max\{\omega_{src}, p_{prev} \cdot \omega_{prev}\}$. The calculation method can be summarized as:

$$p_{curr} = \begin{cases} 0 & \omega_{src} = 0, \omega_{prev} = 0 \\ \omega_{src} & \omega_{src} > 0, \omega_{prev} = 0 \\ p_{prev} \cdot \omega_{prev} & \omega_{src} = 0, \omega_{prev} > 0 \\ \max\{\omega_{src}, p_{prev} \cdot \omega_{prev}\} & \omega_{src} > 0, \omega_{prev} > 0 \end{cases} \quad (1)$$

The priority assignment method and the buffer management policy can enforce social selfishness. Packets that traverse different social links will receive different forwarding service. As shown in Figure 2, although $m1$, $m2$, and $m3$ have the same priority in the source, they will receive different service at relay B after being transmitted to B . That is, $m3$ will not receive any forwarding service, and $m1$ will receive better service than $m2$.

The priority of a packet does not consider the destination of the packet due to the following reason. If a node forwards all the packets destined to its social ties no matter where these packets are from, a malicious source can exploit this vulnerability to send unsolicited packets (e.g., advertisement) via this node to its social ties, although the source has no social tie with this node or its social ties. We note that the malicious source can launch attacks (e.g., source address forgery) to achieve similar goals, but these security issues are out of the scope of this paper.

3.2. 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} ?

3.2.1. Overall Delivery Probability

We assume N can deliver m when it contacts D if it still buffers m at the time of contact. Then m will either be dropped before N contacts D or delivered when N contacts D . There are two cases of dropping. First, m expires before N contacts D and is then dropped. This dropping is due to N 's insufficient contact opportunity with D . Second, N drops m in order to allocate the buffer space originally occupied by m to other newly received packets which have a higher priority. This dropping happens 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 $P_{delivery}$. Then the probabilities of the first and second type of dropping 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 assume that the two dropping events are independent. 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. Thus, 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, $\{t_{exp} \leq t_c\}$ and $\{t_{over} \leq t_{exp}\}$. The first one means that the packet expires before N 's next contact with D , so we call it *expiration dropping*. The second one means that the packet overflows before expiration, so we call it *buffer overflow dropping*. Let P_{exp} and P_{over} denote the expiration dropping probability and buffer overflow dropping probability, respectively. Next, we describe how to estimate them.

3.2.2. Expiration Dropping Probability

To estimate P_{exp} , we adopt an approach similar to that in [12]. Let random variable X denote the inter-contact time between N and the destination D . Assume that each inter-contact time is independent, then by 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.

3.2.3. Buffer Overflow Dropping Probability

The most important factor that affects P_{over} is m 's priority value p due to the buffer policy. Two other 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 machine learning techniques and model it as a supervised learning problem. Whenever N drops or forwards

Figure 3: Heuristics used to estimate buffer overflow dropping probability P_{over} . Triangles and squares are historical samples.

a packet, it generates a record $\langle p, L_0, t_r, \beta \rangle$. With machine learning terminology, each record is called a *sample*, p , L_0 , and t_r are called *feature dimensions* and β 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 within $\mathcal{S}$ the subset of packets being dropped due to buffer overflow is $\mathcal{S}_{drop}$. Then P_{over} is estimated as:

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

Figure 3 illustrates the idea in a two-dimensional space $\langle p, L_0 \rangle$, where the historical packets in the dashed circle are the matched ones. In this example, the estimated P_{over} of $m1$ and $m2$ are $5/6 = 0.83$ and $1/4 = 0.25$, respectively.

To match m to similar packets, we choose the K-Nearest-Neighbor (KNN)[17] algorithm from the machine learning 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 [18] have been proposed to improve its online matching time, they are too complex 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 space².
- In the online phase, nodes scan the $\hat{K}$ points in the increasing order of their distance with m 's feature vector

²We refer to the original paper for details.

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. 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 that of naive KNN (i.e., $O(n\sqrt{n})$). To reduce the computation cost, 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 $D(m_1, m_2) = \sqrt{\sum_{i=1}^3 \frac{\hat{\sigma}_i^2}{\sigma_i^2} (m_1^i - m_2^i)^2}$, where σ_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 forwarding set selection, pseudo-code for M

```

1: Compute the selfish gain  $g$  for each 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

```

3.3. 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 $\mathcal{C}$, we need to find out how to determine a subset of $\mathcal{C}$ to transmit.

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:

$$L_m = L_0 + \sum_{\{k|p_k < p\}} l_k \quad (6)$$

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 (i.e., 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 to be transmitted ($X_i = 1$) or not ($X_i = 0$). According to the above two principles, the problem can be formulated as:

$$\max \sum_{i \in \mathcal{C}} g_i X_i \quad s.t. \quad \forall i \quad \sum_{j \leq i} X_j l_j \leq L_i \quad (7)$$

In a special case that all candidate packets have the same priority, Eq. 7 becomes a standard Knapsack problem. Thus, Eq. 7 is a variant of the Knapsack problem. Next we show that it can be converted into an MKPAR [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} ($j \in \{2, \dots, |\mathcal{C}|\}$) 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

Table 1: The summary of the two traces used for evaluation

Trace	Infocom05	Reality
Network type	Bluetooth	Bluetooth
Number of devices	41	97
Number of contacts	22 thousand	110 thousand
Duration	3 days	9 months
Granularity	120 seconds	300 seconds

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. 7 can be rewritten as:

$$\max \sum_{i=1}^{|\mathcal{C}|} \sum_{j=1}^{|\mathcal{C}|} g_i X_{ij} \quad s.t. \quad \forall i \sum_j X_{ij} \leq 1, \quad \forall j \sum_i X_{ij} l_i \leq \mathbb{S}_j \quad (8)$$

Since the problem is NP-hard, 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.

4. Performance Evaluations

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

4.1. Experiment Setup

We evaluate SSAR over two traces collected from real-world DTNs: Haggie Infocom05 [35] and MIT Reality [22]. Both are available from the CRAWDAD project in Dartmouth [23]. These traces record contacts among users carrying Bluetooth devices, which periodically discover peer devices and record contacts between them. The chosen traces cover a diversity of environments, from college campuses (Reality) to conference sites (Infocom05), and with experimental periods from several days (Infocom05) to several months (Reality). Table 1 summarizes their main characteristics.

Since the traces do not have the accurate social relationship information among participants (i.e., nodes), we need to construct a weighted directed social network graph upon them. In the graph, a vertex denotes a node in the trace,

Figure 4: An example of social network graph generation in the 4 basic steps.

and an edge denotes a social tie between nodes. Edge weight means the strength of the social tie, and it also means the willingness to forward. For instance, the weight of edge $\overrightarrow{NM}$ is node N 's willingness to forward packets for M . The weight of edge $\overrightarrow{NM}$ and that of $\overrightarrow{MN}$ may be different.

Several measurement studies (e.g., [25]) have empirically found that in real-world social networks node degrees follow power-law distributions. Also, in one recent study [15] participants rate the strength of their friendships nearly uniformly between 0 and 1, which is the best empirical data we can find about the distribution of social tie strength. Thus, we try to ensure that the constructed social network graph have these two basic properties.

Our construction process involves four steps as illustrated in Fig. 4. *Step 1*: Generate power-law distributed node degrees. Since the degree of a node means the number of edges emitted from this node, we denote a degree d with d arrows in Fig. 4. *Step 2*: Generate weights for these arrows (which will become edges in the finished graph) that follow the uniform distribution between 0 and 1. *Step 3*: Assign those degrees to nodes. When this step is complete, each node has a number of edges emitting from it. *Step 4*: Connect each of these edges to another node.

To better evaluate SSAR, we generate two types of social network graphs which differ in the last two steps. The first type is *contact-dependent graph*, where the contact frequency between nodes probabilistically determines how degrees are assigned to nodes and how edges are connected. It is based on the following heuristic which has been verified by sociology studies [5]. 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 . Then the last two steps for contact-dependent graph are as follows:

- *Step 3*: Repeatedly assign node degrees to nodes, 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.
- *Step 4*: For each node N , connect its edges to other nodes. First connect the edge with the highest weight 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 edges and nodes that have not been connected to N . In the end, for any ordered node pair NM that has not been connected yet, the willingness of N for M is set 0.

The second type of graph is *random graph*. To construct it, in the third and fourth step we simply assign degrees to random nodes and connect each edge of a node to another random node. Though we believe random graph is less realistic than contact-dependent graph, we still use it in order to evaluate SSAR under diversified social graphs.

The power-law coefficient used to generate node degrees is fixed at 1.76, which is based on the result of an empirical study [25]. To generate discrete node degrees, we use the Zeta distribution with exponent $s = 1.32$ which is the discrete equivalent of the power-law distribution with coefficient 1.76. Since each trace has a certain number of nodes, it is meaningless to generate node degrees equal to or larger than the number of nodes in the trace. In our simulations, we only generate node degrees which fall into range $[1, 96]$ and $[1, 40]$ for the Reality trace and Infocom05 trace, respectively. Let l and u denote the lower and upper boundary (inclusive) of the range, respectively. Then each degree $l \leq d \leq u$ will be generated with probability $\frac{1/d^s}{\sum_{a=l}^u (1/a^s)}$. In this manner, the generated node degree has finite mean and variance. One important feature of social network is the average number of social ties that each node has. In some networks, each node only has a few social ties; while in others, each node has many social ties. We tune parameter l to generate social network graphs with different average numbers of social ties per node.

In the simulation, each node generates packets to random destinations. Each packet has a certain TTL and will be removed after it expires. All packets have initial priority 1. In each run, the first 1/3 (1/10, resp.) of the Reality (Infocom05, resp.) 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 (1/10, resp.) of the Reality (Infocom05, resp.) trace. The default

Table 2: The default parameters used in the simulation

Parameter	Value
Bandwidth	2 Mbps
Buffer Size	5 MB
Packet Size	Random between [50,100] KB
Packet TTL	100 days (Reality), 48 hours (Infocom05)
Packet Generation Rate	1 pkt/node/day (Reality), 6 pkt/node/hour (Infocom05)
Avg. Number of Social Ties per Node	25 (Reality), 8 (Infocom05)

parameters used in the simulation are given in Table 2.

4.2. Routing Algorithms

We compare SSAR with two other benchmark algorithms, PROPHET [6] and SimBet [8]. PROPHET is a standard non-oblivious benchmark that has been used to compare against several previous works [10]. 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 [6]. SimBet has also been used as a benchmark in several works [11]. It calculates a simbet metric using two social measures (similarity and betweenness). A packet is forwarded to a node if that node has a higher simbet metric than the current one. We use the same parameters as in [8].

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 [9]. 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 so that they have some basic selfishness awareness. 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

Figure 5: Comparison of algorithms in the forwarding mode. The Reality trace and contact-dependent graph are used.

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-2*, and label the basic one *PROPHET-1*.

We evaluate SSAR and these algorithms under the forwarding mode and replication mode introduced in Sec. 2.6.

4.3. Metrics

We use the following metrics to evaluate these algorithms: 1) **Packet delivery ratio:** The proportion of packets that are delivered to their destinations out of the total unique packets generated. 2) **Packet delivery cost:** The total number transmissions divided by the number of unique packets generated. 3) **Packet delivery delay:** From the time a packet is generated to the time the packet is delivered. In the results, we plot the average delay of all delivered packets. 4) **Selfishness satisfaction (SS):** A node's SS is defined as the ratio of the average priority of the packets it forwards or delivers over the average priority of the packets it drops. SS reflects how much the user is satisfied with the network, because a larger SS indicates more important messages are served. In the results, we plot the average SS of all nodes.

4.4. Results

The Effects of TTL We first evaluate the algorithms when the forwarding mode (see Sec. 2.6) is used, i.e., each packet has only one copy in the network, so that the effect of packet replication can be separated. Figure 5 shows the results over the Reality trace when the contact-dependent graph is used.

Figure 5(a) compares the algorithms in packet delivery ratio. When the TTL increases all algorithms deliver more packets to the destinations. However, as the TTL becomes large the increment in packet delivery ratio becomes marginal, because at that time the forwarding capacity of the network becomes the performance bottleneck. Among all algorithms, SSAR has the highest packet delivery ratio. For example, when the TTL is 100 days it outperforms

(a) Performance

(b) Cost

(c) Delay

Figure 6: Comparison of algorithms in the forwarding mode. The Reality trace and random graph are used.

(a) Performance

(b) Cost

(c) Delay

Figure 7: Comparison of algorithms in the forwarding mode. The Infocom05 trace and contact-dependent graph are used.

PROPHET-2, SimBet and PROPHET-1 by 80%, 120% and 300%, respectively. This is because SSAR incorporates user willingness, buffer constraint, and contact opportunity into relay selection. It avoids low-willingness nodes or overloaded hot spots when selecting relays, and has much less packet dropping caused by these nodes. In contrast, PROPHET and SimBet cannot avoid those nodes since they only consider contact opportunity when selecting relays. Another reason is that the MKPAR formulation of SSAR does not forward a packet to the next hop whose buffer is insufficient to hold this packet, but PROPHET and SimBet may make such transmissions and result in packet dropping. PROPHET-1 performs much worse than PROPHET-2 because PROPHET-1 is not selfishness-aware. It forwards many packets to those who are unwilling to forward, and those packets are dropped.

Figure 5(b) compares the algorithms in packet delivery cost. As the TTL increases, all algorithms have more transmissions, because packets stay longer in the network and have more opportunities to be transmitted. Among the algorithms SSAR has the smallest cost, since it makes very cautious forwarding decisions, i.e., it forwards a packet only when the next hop has both a good contact opportunity and high willingness. PROPHET-1 has much lower cost than PROPHET-2, because most packets are dropped early.

SSAR's gain in packet delivery ratio and cost does not come for free. As shown in Figure 5(c), its packet delivery delay is longer than that of the other three algorithms, because it does not forward packets to the relays that have a good contact opportunity (which usually means a shorter delivery delay) but low willingness. However, the packets forwarded to these relays also have a high risk of being dropped due to the low willingness. Thus, there is a tradeoff between packet delivery ratio and delay.

Similar results are also found in the Infocom05 trace as shown in Fig. 7.

Figure 6 shows the results on the Reality trace when the random graph is used. Similar as under the contact-dependent graph and for similar reasons, SSAR delivers more packets than the other algorithms (see Fig. 6(a)) but also has longer packet delivery delays (see Fig. 6(c)). The cost of SSAR is lower than that of PROPHET-2 and SimBet but higher than that of PROPHET-1 (see Fig. 6(b)). Here, PROPHET-1 has the lowest cost since most packets are dropped quite early, which can be seen from its extremely low packet delivery ratio.

Comparing the results under the two types of social network graph in the Reality trace (see Fig. 5 and 6), we found that SimBet delivers around 30% more packets under the random graph than it does under the contact-dependent

Figure 8: Comparison of algorithms in the replication mode. The Reality trace and contact-dependent graph are used.

graph. This is because in the contact-dependent graph a socially popular node is very likely to be a hot spot node in contact, while in the random graph such probability is much lower. Since SimBet tends to forward packets to socially popular nodes, it overloads more hot spot nodes under the contact-dependent graph. PROPHET-1 also delivers much less packets under the random graph. The reason is that PROPHET-1 forwards packets out no matter the contacted nodes are socially tied or not, and in the random graph the contacted nodes are more likely to be the ones without social ties, resulting in more packet dropping. PROPHET-2 delivers 25%-35% less packets in the random graph, since less contacts that happen between social ties can be used for packet forwarding. SSAR delivers similar amounts of packets under the two types of social network graph. The cost of all algorithms is lower in the random graph since fewer contacts happen between socially tied nodes. Packet delivery delay does not change much in the two graphs.

Next we evaluate the algorithms when the replication mode (see Sec. 2.6) is used over the Reality trace. The packet generation rate is 0.1 packets per node per day. As shown in Fig. 8(a), all algorithms have a similar packet delivery ratio; however, SSAR achieves such performance at much lower cost (see Fig. 8(b)). For example, when TTL is 75 days, the cost in SSAR is only 8%-13% of that in the other three algorithms. Again, this is because SSAR makes very cautious but effective transmissions. If a packet has a high probability to be dropped by the next hop, most likely SSAR will not replicate the packet. In contrast, the other three algorithms are more greedy in replication. Many replicas only play a marginal role in packet delivery, and very likely such transmissions are wasted. In addition, the MKPAR formulation of SSAR also avoids many useless transmissions. As shown in Fig. 8(c), all algorithms have similar packet delivery delays.

The Effects of Buffer Size To evaluate how SSAR performs when nodes have different storage resources, we

Figure 9: Comparison of algorithms on the Reality trace when nodes have different buffer sizes. The forwarding mode and contact-dependent graph are used.

Figure 10: Comparison of algorithms on the Reality trace when nodes have different average numbers of social ties. The replication mode and contact-dependent graph are used.

change the buffer size of each node from 1MB to 10MB over the Reality trace and show the results in Fig. 9. When the buffer size increases, all the three metrics (packet delivery ratio, cost and delay) also increase. SSAR delivers much more packets than the other three algorithms at lower cost but it also has a longer packet delivery delay for aforementioned reasons.

The Effects of the Average Number of Social Ties per Node Fig. 10 shows the results on the Reality trace when each node generates 0.1 packets per day and the TTL is 50 days. When nodes have more social ties the packet delivery ratio of all algorithms increases. The packet delivery cost of SSAR, SimBet and PROPHET-2 also increases. Obviously, the increment in packet delivery ratio and cost is because more nodes can be used for relays and packets have more chances to be transmitted. SSAR achieves similar performance with other algorithms at much lower cost, and these algorithms have very similar delays that do not change much with the average number of social ties.

The Effects of Willingness-aware Forwarding Fig. 11 compares SSAR with its variant where willingness is not

Figure 11: The effects of willingness awareness in SSAR. The forwarding mode and contact-dependent graph are used over the Reality trace.

Figure 12: The accuracy of our KNN-plus-Kcenter algorithm in estimating the buffer overflow dropping probability of the packets that fall into 10 priority intervals with the i^{th} interval being $[0.1 \cdot (i - 1), 0.1 \cdot i]$. The forwarding mode and contact-dependent graph are used over the Reality trace.

considered in relay selection (i.e., only contact opportunity is considered). The Reality trace is used and the buffer size is 1MB. As shown in Fig. 11(a) and 11(b), SSAR achieves a higher packet delivery ratio at lower cost than its variant that does not consider willingness, and their difference is more significant when the workload is higher. This is because willingness significantly affects the probability that a packet will be dropped by a node, especially when the workload is high. For instance, when each node generates only 0.25 packets per day, SSAR outperforms its variant by 10% in packet delivery ratio with 7% lower cost. However, when each node generates 2 packets per day, SSAR outperforms its variant by 31% in packet delivery ratio at 23% lower cost. When delay is considered (see Fig. 11(c)), SSAR and its variant has a small difference which is only 4%-7%. Thus, willingness-aware forwarding is effective.

Estimation Accuracy Now we evaluate the accuracy of our KNN-plus-Kcenter algorithm in estimating the probability that a packet will be dropped due to buffer overflow. Since packet priority is the major factor that affects this

probability, we divide the packets into 10 groups whose priority falls into 10 intervals evenly spanned in $[0, 1]$, i.e., the i^{th} priority interval is $[0.1 \cdot (i - 1), 0.1 \cdot i]$ ($1 \leq i \leq 10$). We compare the average estimated dropping probability of each group of packets with the real dropping rate of the same group in Fig. 12, where SSAR works in the forwarding mode over the Reality trace and the buffer size is 1MB. As shown in Fig. 12, the estimation result of our algorithm is close to the real dropping rate, and our algorithm can correctly predict that low-priority packets are more likely to be dropped than high-priority packets. Thus, it helps direct packets to those relays with high willingness to forward.

Allowed Selfishness To compare SSAR with other algorithms on how much selfishness is allowed, we plot the SS metric in Fig. 13 and 14. The buffer size is 1MB and 5MB in the Reality and Infocom05 trace, respectively. The two figures show that SSAR allows better selfishness than the other three algorithms. This is because SSAR's buffer management policy satisfies social selfishness. Also, due to the selfish gain metric and the MKPAR formulation, high-priority packets are more likely to be forwarded than low-priority ones. In contrast, the other three algorithms manage buffers without selfishness information and forward packets purely based on contact opportunity, so they perform much worse. We noted that the SS in SSAR drops as the packet generation rate increases. This is because when the workload is higher more packets are dropped. Thus, the average priority of dropped packets increases and SS decreases. In the extreme, if the workload is so high that nearly all packets are dropped, the SS will be close to 1.

5. Related Work

Many algorithms have been proposed for routing and data dissemination in DTNs, and they vary in what information is used to evaluate a node's forwarding capability and make forwarding decisions. Earlier works [6, 7] evaluate the forwarding capability of a node by the historic contact information, while more recent studies [8, 10, 28, 29] employ the social property (e.g., betweenness and centrality) of nodes or transient contact patterns [32] for relay selection. Gao and Cao [33] also propose a social centrality metric that considers the social contact patterns and interests of mobile users simultaneously and achieves effective relay selection. However, these approaches do not consider the social willingness of users to forward packets for others, and implicitly assume nodes are fully willing to forward packets for each other, which may not be always true in reality. Algorithms [26, 27] have also been proposed for finding the right relays for data forwarding in vehicular ad hoc networks, but they consider a different scenario from this work.

Hui et al. [34] study the impact of altruistic behavior on the communication throughput of mobile social networks.

(a) Contact-dependent Graph

(b) Random Graph

Figure 13: Comparison of algorithms in selfishness satisfaction over the Reality trace. The forwarding mode is used.

(a) Contact-dependent Graph

(b) Random Graph

Figure 14: Comparison of algorithms in selfishness satisfaction over the Infocom05 trace. The forwarding mode is used.

The “altruism” concept in their work is similar to the willingness concept in this work in that altruism also describes if a node will forward packets for other nodes. However, there is important difference between the two. Altruism is used in an absolute sense and it denotes the probability that a node will forward a received packet. On the contrast, willingness is used in a relative sense, and it takes effect only when multiple packets contend for shared limited resource (e.g., buffer and bandwidth). More importantly, their focus is the impact of altruism on opportunistic communication but our focus is to design an effective and efficient routing protocol that considers user willingness. Thus, this work and their work are complementary to each other.

Most existing routing algorithms explicitly or implicitly assume unlimited buffer (e.g., delegation forwarding [11]) which is unrealistic. Though some algorithms [9, 7] consider the buffer constraint in design, their attention is limited to which packets to drop when buffer overflows. However, SSAR integrates the buffer constraint into relay selection and takes it as a factor of delivery probability. Some work (e.g., RAPID [9]) has addressed in which order candidate packets should be transmitted to a better relay during a contact, but existing approaches ignore that the forwarded

packets may be immediately dropped by the relay due to the buffer constraint. SSAR uses MKPAR to determine both which subset of candidate packets to forward and in what order to forward them. The formulation is different from the Knapsack formulation in [29] and [28]. Recently, Thompson *et al* [31] propose a congestion control scheme for DTNs which uses the local dropping information to control packet replications. Their scheme is different from SSAR since it controls a node's overall amount of replications but does not determine which node is a better relay.

Individual selfishness has been widely studied in mobile ad hoc networks [14, 30] and even in DTNs [13, 12]. The solutions proposed so far fall into three categories, credit-based approaches (e.g., [14, 13]), reputation-based approaches (e.g., [30]) and gaming-based approaches (e.g., [12]). The principle idea is to stimulate users to forward packets for others. As discussed in Section 2.1, they cannot be directly applied to the social selfishness problem.

6. Conclusions

This paper introduces the social selfishness problem into DTNs and proposes a routing algorithm SSAR following the philosophy of *design for user*. SSAR allows users to behave in the socially selfish way and improves performance by considering user willingness, resource constraints, and contact opportunity when selecting relays. Extensive simulations on the MIT Reality trace and Infocom05 trace show that SSAR can maintain social selfishness and achieve very good routing performance in an efficient way.

Acknowledgments

The authors would like to thank the anonymous reviewers whose insightful comments helped us improve the presentation of the paper. A preliminary version [1] of the paper appeared in IEEE INFOCOM'2010. This work was supported in part by Network Science CTA under grant W911NF-09-2-0053.

References

- [1] Q. Li, S. Zhu, G. Cao, Routing in socially selfish delay tolerant networks, Proc. IEEE INFOCOM, 2010.
- [2] K. Fall, A delay-tolerant network architecture for challenged internets, Proc. SIGCOMM, 2003.
- [3] M. Motani, V. Srinivasan, P. Nuggehalli, PeopleNet: engineering a wireless virtual social network, Proc. Mobi-Com, 2005.

- [4] P. Hui, A. Chaintreau, J. Scott, R. Gass, J. Crowcroft, C. Diot, Pocket switched networks and human mobility in conference environments, SIGCOMM Workshops, 2005.
- [5] M. Granovetter, The strength of weak ties, *The American Journal of Sociology* 78 (6) (1973) 1360–1380.
- [6] A. Lindgren, A. Doria, O. Schelen, Probabilistic routing in intermittently connected networks, *ACM SIGMOBILE CCR* 7 (3) (2003) 19–20.
- [7] J. Burgess, B. Gallagher, D. Jensen, B. Levine, Maxprop: Routing for vehicle-based disruption-tolerant networks, *Proc. INFOCOM*, 2006.
- [8] E. Daly, M. Haahr, Social network analysis for routing in disconnected delay-tolerant MANETs, *Proc. MobiHoc*, 2007, 32–40.
- [9] A. Balasubramanian, B. N. Levine, A. Venkataramani, Dtn routing as a resource allocation problem, *Proc. ACM SIGCOMM*, 2007.
- [10] P. Hui, J. Crowcroft, E. Yoneki, Bubble rap: social-based forwarding in delay tolerant networks, *Proc. MobiHoc*, 2008.
- [11] V. Erramilli, A. Chaintreau, M. Crovella, C. Diot, Delegation Forwarding, *Proc. MobiHoc*, 2008.
- [12] U. Shevade, H. Song, L. Qiu, Y. Zhang, Incentive-aware routing in dtns, *IEEE ICNP*, 2008.
- [13] B. Chen, C. Choon, MobiCent: A Credit-Based Incentive System for Disruption Tolerant Network, *Proc. IEEE INFOCOM*, 2010.
- [14] S. Zhong, J. Chen, Y. R. Yang, Sprite: A simple, cheat-proof, credit-based system for mobile ad-hoc networks, *Proc. IEEE INFOCOM*, 2003.
- [15] E. Gilbert, K. Karahalios, Predicting tie strength with social media, *Proc. CHI*, 2009.
- [16] A. Josang, S. Pope, Semantic constraints for trust transitivity, *Proceedings of the Asia-Pacific Conference of Conceptual Modelling (APCCM)* (Volume 43 of *Conferences in Research and Practice in Information Technology*), 2005.

- [17] G. McLachlan, *Discriminant Analysis and Statistical Pattern Recognition*, Wiley, 1992.
- [18] J. H. Friedman, J. L. Bentler, R. A. Finkel, An algorithm for finding best matches in logarithmic expected time, *ACM Transactions on Mathematical Software* 3 (3) (1977) 209–226.
- [19] T. Gonzalez, Clustering to minimize the maximum inter-cluster distance, *Theoret. Comput. Sci.* 38 (1985) 293–306.
- [20] D. . Loftsgaarden, C. P. Quesenberry, A nonparametric estimate of a multivariate density function, *Ann. Math. Statist.* 1965.
- [21] M. Dawande, J. Kalagnanam, P. Keskinocak, R. Ravi, F. Salman, Approximation algorithms for the multiple knapsack problem with assignment restrictions, *Journal of Combinatorial Optimization* 4 (2000) 171–186.
- [22] N. Eagle, A. Pentland, Reality mining: sensing complex social systems, *Personal and Ubiquitous Computing* 10 (4) (2006) 255–268.
- [23] A. community resource for archiving wireless data at Dartmouth, <http://crawdad.cs.dartmouth.edu/data.php>.
- [24] A. Chaintreau, A. Mtibaa, L. Massoulie, C. Diot, The diameter of opportunistic mobile networks, *Proc. ACM CoNEXT*, 2007.
- [25] A. Mislove, M. Marcon, K. P. Gummadi, B. Druschel, Measurement and analysis of online social networks, *Proc. IMC*, 2007.
- [26] J. Zhao, G. Cao, Vadd: Vehicle-assisted data delivery in vehicular ad hoc networks, *Proc. IEEE INFOCOM*, 2006.
- [27] Y. Zhang, J. Zhao, 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, A. Passarella, Contentplace: Social-aware data dissemination in opportunistic networks, *Proc. MSWiM*, 2008.

- [29] W. Gao, Q. Li, B. Zhao, G. Cao, Multicasting in delay tolerant networks: A social network perspective, Proc. ACM MobiHoc, 2009.
- [30] J. J. Jaramillo, R. Srikant, Darwin: Distributed and adaptive reputation mechanism for wireless ad-hoc networks, Proc. ACM MobiCom, 2007.
- [31] N. Thompson, S. C. Nelson, M. Bakht, T. Abdelzaher, R. Kravets, Retiring Replicants: Congestion Control for Intermittently-Connected Networks, Proc. IEEE INFOCOM, 2010.
- [32] W. Gao, G. Cao, On Exploiting Transient Contact Patterns for Data Forwarding in Delay Tolerant Networks, Proc. IEEE ICNP, 2010.
- [33] W. Gao, G. Cao, User-Centric Data Dissemination in Disruption Tolerant Networks, Proc. IEEE INFOCOM, 2011.
- [34] P. Hui, K. Xu, V. O.K. Li, J. Crowcroft, V. Latora, P. Lio, Selfishness, Altruism and Message Spreading in Mobile Social Networks, Proceeding of First IEEE International Workshop on Network Science For Communication Networks (NetSciCom09), 2009.
- [35] A. Chaintreau, P. Hui, J. Crowcroft, C. Diot, R. Gass, J. Scott, Impact of Human Mobility on Opportunistic Forwarding Algorithms, IEEE Trans. on Mobile Computing 6 (6)(2007), 606-620.
- [36] S.M. Degirmencioglu, K.A. Urberg, J.M. Tolson, P. Richard, Adolescent Friendship Networks: Continuity and Change Over the School Year, Merrill-Palmer Quarterly 44 (3) (1998), 313-37.
- [37] R.I.M. Dunbar, Neocortex size as a constraint on group size in primates, Journal of Human Evolution 22 (6)(1992), 469-493.
- [38] C. McCarty, P.D. Killworth, H.R. Bernard, E. Johnsen, G. Shelley, Comparing Two Methods for Estimating Network Size, Human Organization 60 (1992), 28-39.