RELICS: In-network Realization of Incentives to Combat Selfishness in DTNs

Md Yusuf Sarwar Uddin, Brighten Godfrey, Tarek Abdelzaher
Department of Computer Science
University of Illinois at Urbana-Champaign, IL 61801, USA
Email: {mduddin2, pbg, zaher}@illinois.edu

***Abstract*—In this paper, we develop a cooperative mechanism, RELICS, to combat selfishness in DTNs. In DTNs, nodes belong to self-interested individuals. A node may be selfish in expending resources, such as energy, on forwarding messages from others, unless offered incentives. We devise a rewarding scheme that provides incentives to nodes in a physically realizable way in that the rewards are reflected into network operation. We call it *in-network realization* of incentives. We introduce explicit ranking of nodes depending on their transit behavior, and translate those ranks into message priority. Selfishness drives each node to set its energy depletion rate as low as possible while maintaining its own delivery ratio above some threshold. We show that our cooperative mechanism compels nodes to cooperate and also achieves higher energy-economy compared to other previous results.**

I. Introduction

In this paper, we design a cooperative mechanism for DTNs that enables the network to achieve network-wide good performance despite *all* nodes being *selfish*. In DTNs, end-to-end connectivity between nodes rarely exists. The disruption in connectivity may be due to the frequent failure of links or due to the high mobility of nodes in a wireless environment. In traditional wireless networks, mobility is seen as an inconvenience that may cause partitions. But in DTNs, movement of nodes is exploited to transfer messages in store-carry-and-forward fashion. In a typical DTN deployment, people use their battery operated hand-held devices (e.g., cell phones, PDAs). These devices usually have short range radios that may not allow them to connect all other devices therein in the form of a multi-hop ad hoc network. People/vehicles need to physically carry messages from one location to another before relaying to another node. When two such devices come closer, they establish a connection between them and exchange messages. In contrast to usual latency experience, message delivery latency, in DTNs, is very long (even, hours).

Essentially each of these devices (which we refer to as "nodes") belongs to an individual human. These devices are usually resource-limited (e.g., powered by non-rechargeable batteries), which may cause their users to be selfish. Selfish users would try to operate their devices in a manner that maximizes their own benefits, being reluctant to assist others. With prevalent disconnectivity, it is quite unlikely that a message would be able to be delivered to its destination unless it has been carried and relayed by another node except the source itself. Sometimes, multiple such relays may be required to make a successful delivery. Therefore, a network wide cooperation among nodes is required.

A subtle question is as to how the network can enforce nodes to cooperate. Nodes may not cooperate, if not properly incentivized. There have been considerable works on incentive protocols for ad hoc networks [1], [2], [3], [4], for p2p networks [5], [6], [7]. But, these schemes are not suitable for DTNs because of some unique properties of DTNs, such as intermittent connectivity, lack of contemporary paths and delayed end-to-end feedbacks. Only a few works focus on DTN selfishness. A recent work, MobiCent [8], describes a payment-based system where nodes are paid in "cents" for their relay services. In [9], [10], utility-based schemes are proposed that allow nodes to exchange messages between themselves so that they can individually maximize their respective utilities. An issue regarding to these schemes is that in most cases these payments or utilities represent some abstract quantity that seldom has any physical existence or realization. Sometimes, the realization of payments requires an out-of-band trusted third party, which is unlikely to be available in DTNs. In contrast, we propose a novel scheme that does not pay nodes virtually, but rewards them for their cooperation in such a way that the reward is physically realizable and can immediately be reflected into service differentiation among nodes. We refer to this as *in-network realization* of incentives.

Our cooperative mechanism relies on the following simple principle—*a node's message is forwarded only if it has forwarded messages originated from others.* This is called *reciprocity of service*. The scheme works as follows. Every node is given an explicit *rank* based on its transit behavior (i.e., forwarding messages originated from others). The rank of a node is a kind of reward that is accumulated when the node participates in relaying messages. Based on the rank of the source node, messages originated from high-ranked nodes are given priority over low-ranked ones. This is how—through message prioritization—the scheme converts an abstract quantity, rank, into network operation and physically realizes the difference among nodes originated due to their varying level of cooperation.

In addition to the message prioritization based on ranking, our proposed scheme allows nodes to limit their resource usage commensurate with the achieved service from the network. We treat energy to be the main resource, and the service is expressed in terms of success rate of message delivery, called delivery ratio. A node needs to expend more energy if it wants to raise its delivery ratio. Yet the node, from a selfish perspective, tries to limit its energy expenditure as low as possible. In our scheme, each node is allowed to set a delivery threshold and to adapt its energy depletion rate based on its rank such that the rate is merely enough to achieve the desired delivery ratio. We consider energy to be the core rationale behind selfishness, and in that sense, we are the first to propose an energy-aware incentive mechanism for DTNs.

The selfishness of a node, in our model, originates from the intention of not willing to expend energy for others. We assume that the strategies available to nodes involve turning on or off their radios and arbitrary reordering or holding of messages while transferring to the peer upon a contact; however, nodes will not deviate from the prescribed protocol. This is a reasonable assumption when, for example, the protocol is implemented in the firmware of mobile devices which users are unlikely to modify. We leave the study of more general models to future work.

To this end, we propose RELICS (in-network REaLization of Incentives to Combat Selfishness), a mechanism to combat selfishness in energy-constrained DTNs. RELICS is not itself a routing protocol; rather it can be mounted on top of any routing protocol that is able to provide a few standard callback routines. The components of RELICS exchange information with the control plane of the routing substrate to manipulate their states (e.g., rank table) and thus implement energy- and incentive-aware message forwarding.

The paper is organized as follows. Section II describes existing literature on incentive techniques in wireless ad hoc networks, p2p systems and DTNs. We explain our proposed cooperative mechanism, RELICS, in Section III. We simulate our protocol and show the evaluation results in Section IV. Section V concludes the paper with some future research directions.

II. Related work

To combat selfishness, the prominent strategy is to offer incentives to nodes for their cooperation or penalize them if they do not cooperate. There are mainly two methods of providing incentives: reputation based and payment based. Reputation based incentive system assesses the cooperation level of nodes and provides services to nodes whose reputation index is high. In [11], authors propose a reputation based system where each node keeps track of what fraction of packets that it has forwarded to a particular neighbor are ultimately forwarded by that neighbor. Each node does this by overhearing the channel after forwarding a packet to a neighbor. The scheme is not be useful in DTNs because of the large spatial separation between the successive message forwarding. It is, therefore, significantly difficult to verify whether a given packet has been further forwarded or not. Another scheme, called SPRITE [12], uses credits to provide incentives to selfish nodes. Whenever a node receives/forwards a packet, it keeps a receipt of the packet, and then submits the receipt to a trusted entity called CCS (Credit Clearance Service). CCS maintains credit records of all nodes and charges or credits nodes depending upon the receipts submitted. The availability of the CCS, however, raises a big concern. In DTNs, the situation could be even worse due to poor connectivity.

There are schemes that model the cooperation as a game and analyze the problem with game theoretic tools. Most game theoretic studies are based on VCG (Vickery-Clarke-Groves) mechanism. VCG mechanism considers the second best option along with the best one to decide payment for participants. VCG payment, sometimes, seems impractical due to *overcharging*, which causes "payment" to exceed "cost". The papers [13] and [14] describe the incentive-compatible opportunistic routing protocols for wireless networks. Zhong et al. [13] show that there exists no forwarding-dominant protocol for ad-hoc games. It also shows that the simple VCG payment does not guarantee the existence of a dominant-subaction solution in routing in the presence some nodes cheating about link costs. They proposed a cryptographic solution to prevent such cheating. In [14], Wu et al. present a payment mechanism that compels each node to share their true packet loss rate while receiving payments for forwarding packets. The subtlety of these payments is that these payments have hardly any physical reality. Sometimes, they are treated as transfer of credits, which requires an out-of-band entity to realize or enforce the credit.

There have been considerable amount of works on incentive mechanisms in P2P networks. P2P system defines selfishness as a problem of *free-riding*, where nodes try to download items from the system without uploading any content. A few incentive strategies are usually applied: direct reciprocity, reputation and currency. BitTorrent [5], one of the most popular p2p systems, uses direct reciprocity strategy, popularly known as Tit-For-Tat (TFT). TFT is based on the basic principle "I'll do for you as much as you did for me". In BitTorrent, a user's download rate is proportional to its upload rate. BitTyrant [6], a modified client of BitTorrent, works around TFT and demonstrates that TFT does not provide enough robustness to BitTorrent. KaZaA, another popular content sharing system, uses reputation to encourage nodes to upload content. MojoNation and KARMA [7] use virtual currency, where nodes earn currency and use those currency when they "purchase" services from peers. These incentive systems are, however, highly communication intensive and are hardly usable in an intermittent environment posed by DTNs.

There are some works on incentive based systems for DTNs. MobiCent [8] proposes a credit-based incentive system where nodes participate in a path revelation game. In this game, nodes are paid for forwarding packets and the destination makes the payment decision. The payment is made in such a way that none gets incentive to tamper the path (i.e., the list of forwarders) that they report to the destination. They handle the edge insertion and edge deletion attacks. Another work [9] proposes a barter-based cooperation scheme. It's a kind of trade model where two nodes, when in contact, exchange messages between them in an exact count so that the utility of an individual node is maximized for those exchanged messages. The mechanism is applied to scenarios where nodes carry a bunch of information of different interests and want to trade them for information, possibly of another type, contained in other nodes. Xie et al. [10] extend the barter idea for source-destination like communication paradigms and replace the notion of utility with a metric, called *appraisal*. A relay node claims reward from the destination node when it successfully delivers a message to the destination. The appraisal of a message is simply the probability that the node will be able to meet the destination multiplied by a constant reward value dependent on the message.

Incentive-aware routing [15] describes an application of the popular incentive mechanism, pair-wise TFT (Tit-For-Tat), for DTNs. TFT is an effective strategy for service reciprocation between two *end hosts*. Since nodes, in DTNs, function mostly as relays (as well as end hosts), pair-wise TFT does not seem to be a viable form of service reciprocation. It is not quite rational that a relay node reciprocates message transfer, in exact byte count, to another relay node, whereas neither of them has any real utility of those messages to themselves. Incentive-aware routing assigns utility to messages when they are delivered to destinations. In contrast, our approach does not look into pair-wise reciprocation; instead, a network-wide assistance is solicited and then reciprocated.

III. The Cooperative Mechanism

In this section, we describe our proposed cooperative mechanism, RELICS. The mechanism attempts to keep overall performance of the network high despite individual node hoarding their resources. The main resource that each node cares most is its energy. As long as nodes experience their desired services from the network, they do not necessarily care about others. It is the responsibility of the cooperative mechanism to ensure that an individual node cannot get its expected service from the network unless it cooperates.

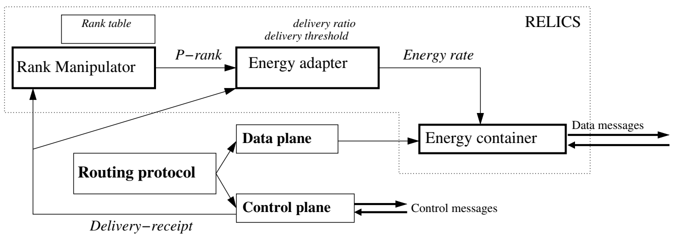


Fig. 1. RELICS framework

A. RELICS Framework

Figure 1 presents the RELICS framework. RELICS can convert a DTN routing protocol into an incentive-aware protocol, if properly instrumented. There are indeed a good number of protocols that can be instrumented with RELICS, provided that they can offer a few specific callback routines. RELICS manipulates its internal states and invokes those callback routines to interact with the routing protocol. It has two main components: *rank manipulator* and *energy adapter*. Rank manipulator maintains a rank table in each node storing ranks of other nodes and realizes the message prioritization based on ranking, whereas energy adapter tunes the energy rate at which an individual node should deplete energy. Energy container regulates message communication to limit energy depletion as set forth by the energy adapter. In the following, we describe all these components of RELICS.

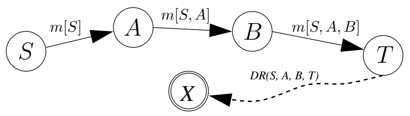


Fig. 2. Rank update upon message delivery

B. Rank Manipulator

RELICS introduces the notion of explicit *ranking* of nodes. *Rank*, a real number, measures the transit behavior of a node. Transit behavior means how many times a third node observes that a particular node participates in relaying messages originated from nodes other than the node itself. Every node accumulates rank, as a reward for cooperation, when it transits messages. The rank of a node assesses the level of cooperation demonstrated by the node. Higher rank indicates higher cooperation and lower rank means lower cooperation. In order to realize the rank into physical actions, these ranks are later translated into message *priority* in that a forwarder favors the messages originated from a high-ranked node over the messages from a low-ranked one.

1) Computing ranks: Rank is calculated and maintained as follows. When a message is successfully delivered to the destination, the destination node rewards all relay nodes that participated in forwarding of that message. The list of forwarders is populated in the message header as the message traverses from one hop to the next. Upon delivery, the destination floods a *delivery-receipt*, as a form of an acknowledgment, which contains the message ID, the source node ID and the list of forwarders.

Delivery-receipts are normally used for clearing messages from buffers once messages are delivered. We simply augment the functionality to manipulate ranks. Let us consider Figure 2. Node S creates a message that traverses through node A and B to reach the destination T (Figure 2). Then, T floods the delivery-receipt $DR(S, A, B, T)$ in the network. Let some node X (may be even S, A or B, and of course T) receive the receipt. Node X now rewards A and B for their transit service by increasing their associated ranks as follows:

$$rank_X(A) = rank_X(A) + \gamma \times rank_X(S) \quad (1)$$

$$rank_X(B) = rank_X(B) + \gamma \times rank_X(S) \quad (2)$$

$$rank_X(S) = rank_X(S) - \delta \quad (3)$$

where $rank_X(A)$ denotes the rank of node A maintained at node X. It is to note that both A and B gain the same amount of rank increment, which is directly proportional to the rank of the source node, $\Delta rank(.) = \gamma \times rank(S)$. RELICS considers each forwarder equally important, hence each forwarder is rewarded with the same amount of rank increment. It depends neither on their order nor on their count in the delivery-receipt.

In addition to updating ranks of the forwarders, the rank of the source is decremented by a constant amount, δ, as if the source node is required to *pay* the forwarders from its own accumulated rank. Delivery of messages decreases the source rank until it reaches to zero. Once the rank of a node goes down to zero, no other nodes forward messages originated from that node anymore, because the forwarding doesn't generate any reward for the forwarders. The source can, of course, accumulate ranks by relaying messages.

We need to explain why we use the rank update equations as they are. We use *proportional increment* for the forwarders and *constant decrement* for the source. Proportional increment reciprocates the forwarder in proportion to the source rank. This causes a forwarder to favor messages from higher rank sources over lower rank nodes, because those messages, if delivered, bring back more rewards (i.e., rank increment) than others. Thereby, the prioritization of messages based on source rank is enacted. On the other hand, if we had used the same strategy for the source rank, i.e., the proportional decrement, it would discourage nodes to acquire higher ranks, because higher rank in turn would deplete in the same scale. Instead, a constant decrement is proposed that offers positive incentives to source nodes to raise their own ranks by relaying.

What fraction of the source rank can a forwarder claim? That is, what should be the value of γ? That depends on the current rank of the source. Ideally, the source node distributes its own rank among all forwarders who participate in delivering its messages. The rank update scheme follows the following principle, called *conservation of rank*.

Definition. (Conservation of rank) *In the process of updating ranks, the sum of total increments of forwarders' ranks should remain equal to the rank of the source.*

Conservation of rank implies that the source exhausts its rank onto all forwarders. The following proposition gives the relation between γ and δ.

Proposition. (Relation between γ and δ) *Let the current rank of the source be $rank(S)^0 = r_0$. If the rank does not increase afterward, but decreases due to message delivery, the following is true:*

$$\gamma = \frac{2}{r_0/\delta + 1} \tag{4}$$

Proof. Let $r_0 = k \times \delta$. It means that the source can deliver k messages until its rank goes down to zero. After i-th delivery, the source rank becomes $rank(S)^i = r_0 - i\delta$, and the reward gained by each forwarder at that round is $\Delta rank(.)^i = \gamma \times rank(S)^i = \gamma(r_0 - (i-1)\delta)$. Due to the conservation of rank, the sum of increments in forwarders' ranks in k rounds should be equal to the rank of the source. That is:

$$\sum_{i=1}^{k} \Delta rank(.)^i = r_0 \tag{5}$$

This can further be reduced to:

$$\begin{aligned}\sum_{i=0}^{k-1} \gamma(r_0 - i\delta) &= k\delta \\ kr_0\gamma - \frac{1}{2}(k-1)k\gamma\delta &= k\delta \\ \frac{2}{k+1} &= \gamma \end{aligned} \tag{6}$$

Since, $r_0 = k \times \delta$, we have $\gamma = \frac{2}{r_0/\delta+1}$. □

For brevity, we use $\delta = 1$. So, $\gamma = \frac{2}{r_0+1}$. Hence, if the current source rank is r_s, the rank increment for a forwarder is given by:

$$\Delta rank(.) = \gamma \times r_s = \frac{2}{r_0+1} \times r_s \tag{7}$$

Example. Let the initial rank, r_0, of a source node be 4. Hence, $k = 4$ and $\gamma = \frac{2}{5}$. Rank updates occur for four times. At the first update, the forwarder's rank is raised by $\Delta r^1 = \gamma \times 4 = \frac{2}{5} \times 4$. Similarly, $\Delta r^2 = \frac{2}{5} \times 3$, and so on. We can verify that:

$$\begin{aligned}\sum_{i=1}^{4} \Delta r^i &= \frac{2}{5} \times (4+3+2+1) = \frac{2}{5} \times \frac{4 \times 5}{2} \\ &= 4 \quad (= r_0)\end{aligned}$$

There is an issue with Equation (7). It requires γ to be fixed and the same initial rank r_0 to be used at different updates. But at any arbitrary time, a node only knows the current rank of the source, $r_s = rank(S)^i = r_0 - i\delta$, not r_0. We consider two alternatives to fix this:

- **Moving r_0's:** At any time, r_0 is assumed to be the last rank of the source after which the rank has not been improved but decreased due to "repay". That is, r_0 stands for the last immediate highest rank after which the rank is continuously diminishing.
- **Amortizing:** To amortize, we use the current rank $rank(S)$ as the initial rank r_0, and $\frac{1}{2}rank(S)$ as the effective source rank r_s for the current update. As per Equation (7), it gives:

$$\begin{aligned}\Delta rank(.) &= \frac{2}{r_0+1} r_s \\ &= \frac{2}{rank(S)+1}\frac{rank(S)}{2} \\ &= \frac{rank(S)}{rank(S)+1}\end{aligned} \tag{8}$$

The above update follows the proportional increment of the forwarder rank, i.e., higher rank begets higher reward. This approach, however, fails to hold the conservation of rank and under-calculates the sum of rank increments by an order of $O(\log r_0)$. It can be shown as follows:

$$\begin{aligned}\sum \Delta r^i &= \frac{r_0}{r_0+1} + \frac{r_0 - 1}{r_0} + \ldots + \frac{1}{2} \\ &= r_0 - (\frac{1}{r_0+1} + \frac{1}{r_0} + \ldots + \frac{1}{2}) \\ &= r_0 - (H_{r_0+1} - 1) \\ &\leq r_0 - H_{r_0} \leq r_0 - O(\log r_0)\end{aligned}$$

For simplicity of implementation with less overhead per node, we used the amortizing approach to update ranks.

2) Rank maintenance: We notice that a node cannot reward itself. A node's rank is maintained and updated by other nodes. Each node maintains a *rank table* that stores ranks of other nodes. A node can, however, maintain a *shadow* rank of itself computed upon receiving of delivery-receipts. It increments its own rank when it finds itself as a forwarder in some delivery-receipt and decrements as well when one of its messages gets delivered. In this way, the node synchronizes its own rank with ranks stored in other nodes. One important issue is that, for a consistent rank update across the network, all nodes should keep the same rank of a particular node. As long as delivery-receipts are able to reach all nodes, this should hold. In Section IV, we present the corresponding evaluation results.

What would be the initial rank of nodes? Obviously, we need a non-zero value to begin with, otherwise no messages will be forwarded. In our implementation, we set initial rank to 2.0. This allows each node to deliver at least 2 messages before its rank gets down to zero. This bootstraps the process.

3) Forwarding decision: Every node holds a bunch of messages in its buffer that it carries to relay to another node or to deliver to the respective destinations. When a node meets another node, it decides which messages it would transfer to the peer node. Messages are transferred, *selfishly*, in the order of that would generate maximum rewards (i.e., largest rank increments after successful delivery). Let us define the following terms for a message m at node A for the contact, $c = A \rightarrow B$ (i.e., when A meets B).

$src(m)$	source of m
$dst(m)$	destination of m
$r_s(m)$	rank of the source node, $rank_A(src(m))$
$p_c(m)$	delivery probability of m, if forwarded to B
$s(m)$	size of m (in bytes)
$R_c(m)$	expected reward for relaying m to B
v_c	transfer volume of the current contact, c
$\Delta(m)$	reward per byte, $R_c(m)/s(m)$
S	source messages, $src(m) = A$
D	destination messages, $dst(m) = B$
T	transit messages, $src(m) \neq A \wedge dst(m) \neq B$

Forwarding of a message rewards the forwarder, but incurs some cost as well in terms of energy. While the reward is proportional to the source rank, the cost depends on the size of the message. Recall that the reward is achieved, only if the message is delivered to the destination. Hence, the delivery probability, $p_c(m)$, is taken into account to compute the expected reward, $R_c(m)$, as follows.

$$\begin{aligned} R_c(m) &= E[\Delta rank(A)] \\ &= p_c(m) \times \Delta rank(A) \\ &= p_c(m)\frac{r_s(m)}{r_s(m)+1} \end{aligned} \tag{9}$$

Again, each contact has certain *transfer volume* that counts the number of bytes that can be transferred on that contact. The transfer volume is determined by the contact duration or the limited energy budget for the contact, as imposed by the energy container (described later). The forwarder thus needs to choose messages that maximizes the sum of rewards subject to the transfer volume. This corresponds to the following $0/1$-Knapsack problem:

Selfish forwarding. Given a set of messages $T \cup D$ for a contact c, the forwarder chooses the candidate message set $Q \in T \cup D$, such that $\sum_{m \in Q} R_c(m)$ is maximized subject to $\sum_{m \in Q} s(m) \leq v_c - \sum_{m \in S} s(m)$.

But, v_c is a run-time property of the contact and can rarely be determined *apriori*. So, we use a greedy heuristic: transfer messages in descending order of reward per byte, $\Delta(m) = R_c(m)/s(m)$, until the transfer volume is exceeded or the contact is expired, whichever occurs first. The message transfer routine is shown in Algorithm 1.

Algorithm 1 FORWARDMESSAGES (Contact c)

```
$M =$ get-messages($c$)
Partition $M$ into $S$, $D$ and $T$
Populate $Q$, messages to be transferred via contact $c$
$Q = \emptyset$
if ($rank(self) > 0$) then
    $\Delta(m) = \infty$ for $m \in S$
    $Q = Q \cup S$
end if
for all $m \in T \cup D$ do
    $p_c(m) =$ compute-prob($m$), if $m \in T$, else 1
    $R_c(m) = p_c(m) \times r_s(m)/(r_s(m)+1)$
    $\Delta(m) = R_c(m)/s(m)$
    if $\Delta(m) > 0$ then
        $Q = Q \cup \{m\}$
    end if
end for
Sort messages in $Q$ in descending order of $\Delta(m)$
transfer-messages($c, Q$)
```

The message forwarding routine in Algorithm 1 uses three callback functions, namely `get-messages(c)`, `compute-prob(m)` and `transfer-messages (c, Q)` that are supposed to be provided by the underlying routing protocol. The function `get-messages(c)` returns the eligible messages chosen by the routing protocol for the current contact, c. `Compute-prob(m)` returns the probability that the message m would be delivered to its destination if forwarded in contact c. The function `transfer-messages(c, Q)`, when invoked, transfers the messages in Q in the specified order over the connection during the contact. Almost all DTN routing protocols can provide these three functions, but in different implementations. Particularly, `compute-prob(m)` varies significantly among protocols. Below, we describe a few.

PROPHET [16] computes delivery probability based on meeting histories. MaxProp [17] finds the sum of link probabilities along a path. This sum, when divided by hop-count with other small adjustments, can be converted into delivery probability. IC-Routing [18] calculates delivery probability within a deadline from the estimated delay distributions. Delegation forwarding [19] keeps the delivery probability in each message header and updates this probability at each hop.

In Spray-and-Wait [20], each message contains a replica count, indicating how many more copies of this message can be created. Delivery probability can be estimated from the ratio of current and initial replica count.

As a proof of concept, we mount RELICS on top of PROPHET [16]. Below we describe how computing delivery probabilities and selection of messages are done in PROPHET. We also evaluate the corresponding results in Section IV.

4) RELICS on PROPHET: PROPHET uses history of encounters and exploits the transitivity property to infer delivery predictability. In PROPHET terminology, $P_A(B)$ indicates how likely that node A can "meet" another node B (directly or via other nodes). Hence, $p_c(m) = P_A(dst(m))$ for contact c=$A \rightarrow B$. PROPHET uses the following update equations:

$$\begin{aligned} P_A(B) &= P_A(B) + (1 - P_A(B))P_{init} \\ P_A(C) &= P_A(C) + \beta(1 - P_A(C))P_A(B)P_B(C) \\ P_A(B) &= \theta^k P_A(B) \end{aligned}$$

PROPHET's `get-messages(c)` populates the eligible messages to be transferred on contact c as follows, $M = \{m : P_A(dst(m)) < P_B(dst(m))\}$.

C. Energy Adapter

We now explain how RELICS allows nodes to *unilaterally* adjust their energy depletion rate commensurate with their observed service from the network. We define *energy rate* as the amount of energy that a node intends to expend in a given period, called *epoch*. Usually, the epoch duration is an hour and the energy rate is expressed in Joule/Hour. A node expends energy in sending/receiving of its own messages as well as in offering relay services to others. While nodes are very much willing to expend energy for themselves, they feel reluctant in relaying messages originated from others. Our cooperative mechanism provides nodes to independently adjust their own energy rate as well as compels them to participate in providing transit services to others.

The service from the network is measured in terms of message delivery ratio—the fraction of messages that have been delivered to destinations out of what created. This notion of service is inspired from situations where DTNs use *erasure coding* or *fountain coding* to introduce redundancy to cope with lack of connectivity [21], [22]. An erasure coding (with a replication factor r) encodes messages into smaller *blocks* and spread them in the network in several directions. Once $1/r$ fraction of blocks or more are received at the destination, the original message can be reconstructed successfully. If these blocks are treated as messages, then the intended delivery threshold for the sender is $1/r$. Different sources can have different delivery thresholds based on their replication factors. Irrespective of erasure coding is in place or not, we assume that each node independently chooses its delivery threshold and expends energy accordingly.

1) Energy container: Energy container implements energy constrained communication. Functionally, it resembles to a token-bucket, popularly used in rate limiting communication. Unlike countable tokens, the container contains measurable *virtual* energy. At every epoch period, a certain amount of energy, i.e., *energy rate* $\times$ *epoch duration*, is accumulated to the container. Every time when messages are transferred or received, energy, as like tokens in token-bucket, is used up, until it becomes zero. Once the container becomes empty, no further communication is allowed until the next epoch when a new chunk of energy is replenished to the container. Energy rate, as adjusted by the energy adapter, thus regulates the amount of messages that can be transferred or received.

2) Adjusting energy rate: Each node starts with a low energy rate and sets a delivery *threshold* (say, 0.8). It then monitors its current delivery ratio once in a while to determine whether the service level is met or not. If the delivery ratio remains below the threshold, the node increases its energy rate so that the message delivery ratio rises. Once the delivery ratio exceeds the threshold, the node reduces its energy rate to expend less energy. Energy rate adjustment depends on individual rank. The rank of a node indicates at what priority messages from that node are treated in the network. For instance, top ranked nodes should not worry much about their delivery ratios, because other nodes, very much willingly, forward their messages, and their delivery will eventually rise. On the other hand, low ranked nodes should take care of their ranks, since others are giving less priority to them.

Rank is a real number that does not *directly* assess a node's position compared to others. For comparative assessment, each node computes percentile-rank, *p-rank*, which is defined as the fraction of nodes that has rank strictly less than the rank of a given node. P-rank of node n is given by:

$$p\text{-}rank(n) = \sum_{i=1}^{N} \frac{[rank(i) < rank(n)]}{N} \tag{10}$$

where N is the number of known nodes in the network and $[b] = 1$ if b is true, else 0. Basically, p-rank is the normalized rank within $[0, 1]$. At the beginning, when initial ranks of all nodes are equal, p-rank is zero. RELICS uses the procedure shown in Algorithm 2 to adjust energy rate.

Algorithm 2 ADJUSTENERGYRATE

```
if $delivery\ ratio < threshold$ then
    $energy\ rate = energy\ rate + \eta_1 \times (1 - p\text{-}rank)$
else
    $energy\ rate = energy\ rate \times (1 - \eta_2 \times p\text{-}rank)$
end if
```

The procedure is invoked when a message is delivered (notified by delivery-receipts) or a certain duration has been elapsed since the last update (we used, 1 hour). Two constants, η_1 and η_2, determine the rate at which energy is raised and lowered respectively in response to the delivery ratio. The adjustment rules are quite intuitive. We use the very well known strategy, *additive increase* and *multiplicative decrease*. If the delivery ratio falls behind the threshold, the node increases its energy rate, but in the linear scale of (1- *p-rank*). High-rank nodes raise their energy in a slower rate than low-rank nodes. On the other hand, for higher delivery ratio, the node sets back immediately and decreases its energy rate in the multiplicative scale of *p-rank*. This time, high-rank nodes decrease their energy rate quicker than low-rank nodes. Under some operating conditions, we choose $\eta_1 = 28.57$ J/Hr and $\eta_2 = 0.37$ (See Appendix A).

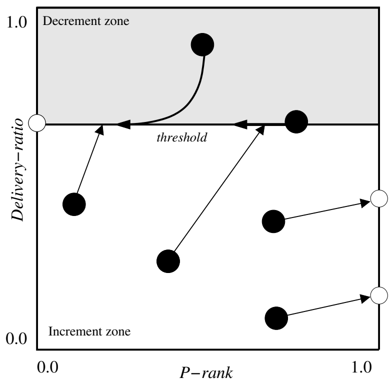


Fig. 3. Rank-delivery square

3) Convergence issues: It is easy to reason that energy rate, rank and delivery ratio all are dependent in a cyclic relationship. Energy rate sets limits on the rate of transit messages, which in turn impacts the rank of the node. Rank defines priority of messages, thus, affects delivery ratio. Delivery ratio again makes energy rate to be adjusted.

We build a *rank-delivery square* formed by p-rank and delivery ratio in X and Y axes respectively (Figure 3). Any node can be mapped onto a point in this square, depending on their current values of p-rank and delivery ratio. The square can be partitioned into two zones by the line *delivery ratio = threshold*. Nodes operating below the line increase their energy rate, whereas nodes above the line do the opposite. There are two line-segments in the square where a node does not change its energy rate. These are:

- *Delivery ratio<threshold* AND *p-rank* $= 1$. The node reached to the highest rank, yet its delivery ratio remains low. Further increase in energy do not lead to any better situation, because rank cannot be improved anymore. The node should remain there until there is any change in delivery ratio or rank.
- *Delivery ratio≥threshold* AND *p-rank* $= 0$. Service level is attained, and the node has moved backed to the lowest rank. Actually, all nodes intend to move to the point where *delivery ratio = threshold* and *p-rank*$= 0$, i.e., all ranks become equal.

In evaluation, we show that the delivery ratio for a certain threshold converges as nodes adjust their energy rate.

D. Measures against Cheating

In the current work, we mainly design incentive-aware data plane, leaving control plane issues as future work. Still we mention a few measures for protecting the control plane against some possible adverse and cheating actions. RELICS has two main components: rank table and delivery-receipts. Rank tables are stored by individual nodes and are not shared among nodes. Hence, they are somehow protected from inadvertent tampering. An adversary can drop delivery-receipts, but it is quite unlikely. This is because that the receipts are required for another purpose—to clear buffers—and that they are very small in size compared to data messages. False delivery-receipts can be checked by having them signed by destinations, if possible. The integrity of hop-list in the receipts can be spoiled by two ways: adding non-legitimate nodes and dropping existing nodes from the hop-list.

A node cannot insert itself in the hop-list of a delivery-receipt unless it forwards the same message. A node cannot insert itself more than once, because that can easily be caught by the next immediate node. An adversary can add a few other nodes into the list to illicitly credit them for forwarding. But this requires the node to collude with other nodes that it adds, which is not the case we assume in our selfish model. RELICS cannot handle cheating associated with collusion.

An adversary can drop a valid forwarder node from the hop-list. Dropping a node from the list does not, however, benefit the adversary anyway. This is rather a malicious action beyond being selfish. This kind of elimination can be prevented to some extend by applying a hop-by-hop one-way hash function, similar to techniques described in [23], [24].

IV. Evaluation

We simulate RELICS in ONE (Opportunistic Network Environment) simulator [25] on top of an urban vehicle movement model (Figure 4). We use a total of 40 moving nodes classified into five groups; two types of cars $(10+10 = 20)$, 10 pedestrians, three sets of trams in three fixed routes $(4+3+3 = 10)$. Cars and pedestrians follow the same *MapBasedRandomMovement* model (a Random Waypoint variant on map) supported by ONE. In this model, the moving entity starts from a random location and moves to another location on the map with a randomly chosen constant speed. Once reached to the destination, the entity chooses another location and moves there. Typical simulation parameters are:

Message generation rate	1 message/node/hour
Message size	$U[1, 10]$ KB
Simulation duration	48 hours
Transmit power	300 mA (1.05 J/sec)
Receive power	200 mA (0.7 J/sec)
PROPHET settings	$(P_{init}, \beta, \theta)$ = $(0.75, 0.25, 0.98)$

We evaluate RELICS in two aspects: firstly, we demonstrate that the reward scheme works; secondly, we show the performance of the proposed energy adapter. The primary metrics that we are interested in are, i) rank or percentile-rank of nodes, ii) delivery ratio, and iii) energy rate. For the sake of fair comparison, while computing delivery ratios, we eliminate the messages that are *directly* delivered to the destinations from sources without being relayed by a third node.

We verify whether RELICS is able to maintain a consistent rank of nodes across the network. Recall that upon receiving delivery-receipts, nodes *independently* reward the forwarders by incrementing their ranks in proportion to the source ranks.

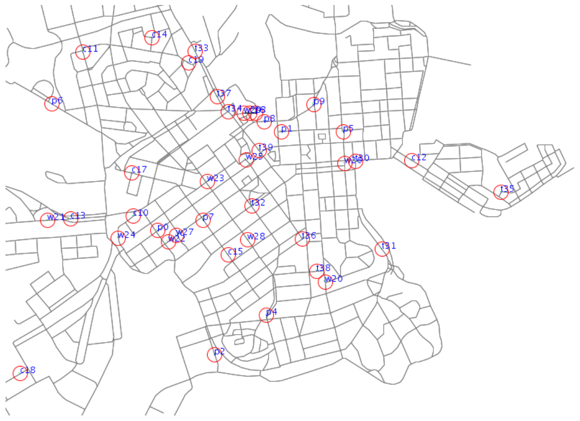


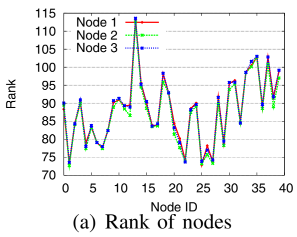
Fig. 4. Simulation map (Nodes are circled by their transmission range)

Figure 5(a) plots the rank of nodes as maintained in three *randomly* selected nodes at some random instance of the experiment. We observe that three nodes record nearly the same ranks of all nodes, subject to the successful flooding of delivery-receipts. This implies that the rank update is consistent across the network. Figure 5(b) shows p-rank of nodes against transit messages, which affirms that p-rank is positively correlated with the number of transit messages.

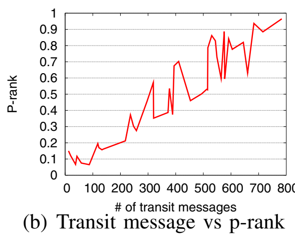
In Figure 6, we demonstrate that our reward scheme works. We verify that rank of nodes is affected due to their energy expenditure, so is the delivery ratio. We consider two cases, *constant* and *adaptive* energy consumption. In constant energy consumption, an individual node *apriori* decides a constant energy rate (we choose uniformly from 1.0-70.0 J/Hr) and does not adjust the rate afterward. Figure 6(b) shows that p-rank of nodes linearly increases with energy expenditure and that high-ranked nodes achieve higher delivery ratio, whereas low-ranked nodes achieve less, as per RELICS's mechanism.

In adaptive energy consumption case, each node starts with an initial energy rate (we use, 5.0 J/Hr) and then gradually adjusts the rate, based on its observed delivery ratio (Algorithm 2). We set the delivery threshold to 0.8. We show the results in Figure 6(b). We observe that nodes now expend a fairly constant amount of energy (approximately 1700 J) while achieving delivery ratio nearly 0.8.

Let us now consider an individual node. Figure 7 plots energy rate, p-rank and delivery ratio of a randomly selected node in every hour of a 48-hour simulation. We observe that the node gradually increases its energy rate at the beginning (up to first 8 hrs), which also raises its delivery ratio. Once the delivery ratio crosses *threshold*=0.8, energy rate gradually declines, yet keeping the delivery ratio above 0.8. P-rank also settles at around 0.4 subsequently. We also observe that energy rate of the node rises and falls while keeping a stable delivery ratio above the threshold line. Figure 7(b) shows the same results for the same node for *threshold*=0.6.



(a) Rank of nodes



(b) Transit message vs p-rank

Fig. 5. Consistent rank maintenance across the network

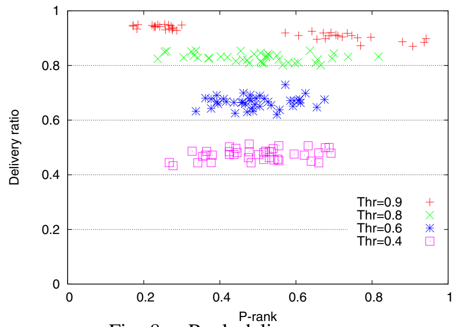


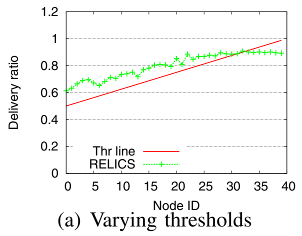
Fig. 8. Rank-delivery square

Figure 8 shows the *rank-delivery square* plotting the delivery ratios of all nodes against their p-ranks, for four delivery thresholds. In all cases, we find that delivery ratios remain fairly close to the respective thresholds. We also show results for the case when different nodes have different thresholds, in Figure 9(a). For simplicity of presentation, we set the threshold for node i by:

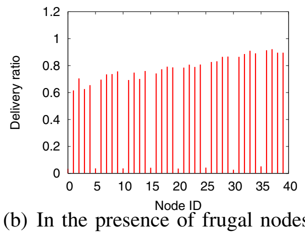
$$threshold(i) = \frac{1}{2} \times \left(1 + \frac{i}{\text{total nodes}}\right)$$

The delivery threshold varies from 0.5 to 1.0 (as presented by the slanted line in Figure 9(a)), and the delivery ratio of nodes remains more or less close to the given threshold.

So far, we assumed that *all* nodes are selfish and that they spend as low energy as possible. Now, we consider the case when some nodes are severely selfish. They do not want to deplete any least energy for others. They only forward their own messages, but never offer transit to others. These are *frugal* nodes. In Figure 9(b), we show delivery ratios of nodes for the same setting of Figure 9(a), but in the presence of a few frugal nodes. Without loss of generality, we choose nodes with ID $0, 5, \ldots, 35$ (multiple of 5) as frugal nodes. We observe that frugal nodes experience very low delivery ratio (≈ 0.04) compared to other nodes. Surprisingly, frugal nodes only hurt themselves; the average delivery ratio of non-frugal nodes remains nearly the same (0.80 vs. 0.79).



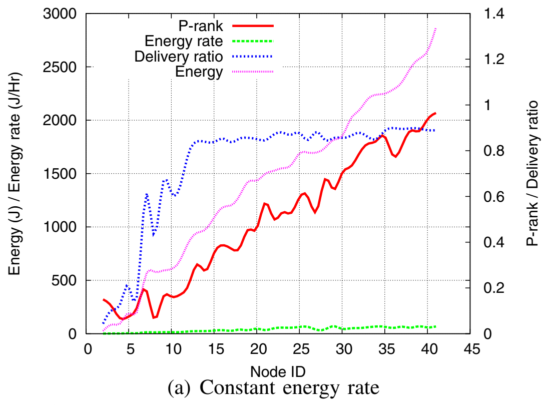
(a) Varying thresholds



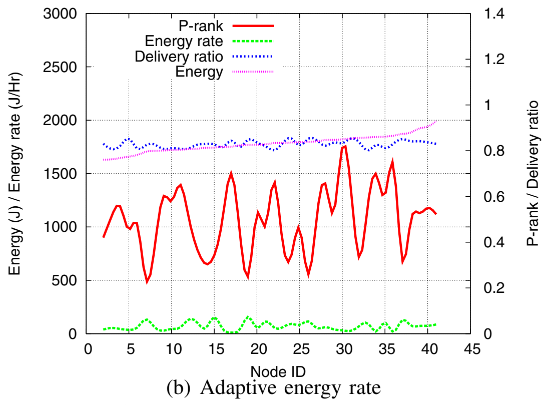
(b) In the presence of frugal nodes

Fig. 9. Effect of frugal nodes

We show energy economy of RELICS by comparing it with two other protocols, PROPHET [16] and IC-Routing [18]. Recall that RELICS is mounted on PROPHET. Figure 10(a) reveals that RELICS on top of PROPHET outperforms the base PROPHET protocol in terms of total energy expended.

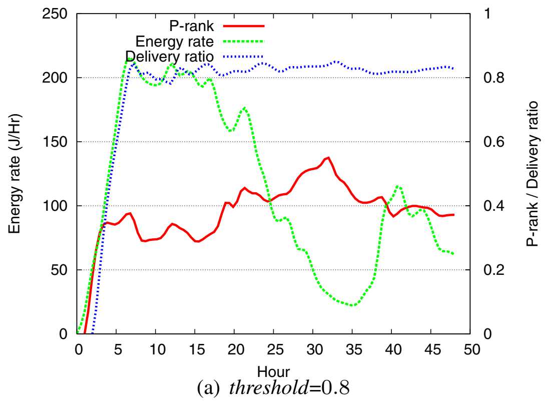


(a) Constant energy rate

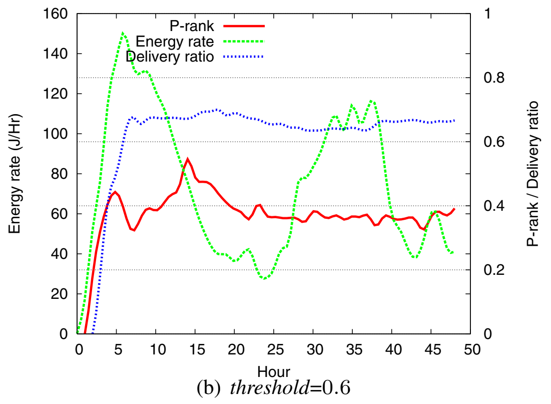


(b) Adaptive energy rate

Fig. 6. Energy rates and delivery ratios for two consumption case (nodes are in ascending order of their total depleted energy)



(a) *threshold*=0.8



(b) *threshold*=0.6

Fig. 7. Energy rate adjustment of a randomly selected node, for two different delivery thresholds

This is due to the fact that RELICS allows nodes to continuously update their energy rates and thus lets them operate at as low energy as possible. Figure 10(b) compares results against IC-Routing, a low-energy DTN routing protocol. The figure shows that RELICS is very competitive in achieving energy economy at various delivery thresholds.

Finally, we compare RELICS with the TFT-based protocol [15]. We run both of them under the same simulation environment for the same traffic pattern. To focus only on incentive part of the protocol, we do not incorporate the online route computation of TFT, rather use PROPHET-like message replication along several directions. We implement TFT with generosity and contrition as described in [15]. Generosity allows nodes not to reciprocate in exact byte count between a pair of nodes, instead a certain amount of asymmetry is allowed. Contrition restrains a node from reacting to a retaliation caused by its own failure to transfer in an earlier phase. Unlike TFT, RELICS explicitly sets limit on energy expenditure, hence is more energy economical than TFT. In Figure 10(c), we observe that RELICS expends less energy than TFT, while achieving nearly the same delivery ratio.

V. Conclusion and Future Work

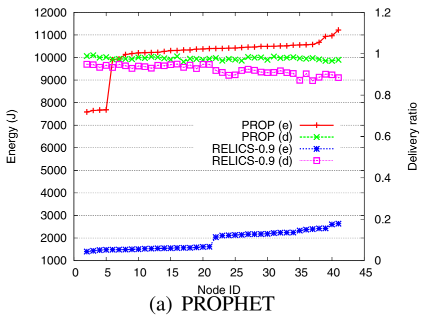
In this paper, we proposed RELICS, an energy-aware incentive mechanism for selfish DTNs. RELICS rewards nodes for their cooperation and then realizes those rewards into network actions by message prioritization. There are several avenues for future research. The current technique does not consider network *asymmetry*, a situation where the actual relay capacity of nodes varies due to their physical positions and mobility in the network. Again, RELICS focuses mainly on the data-plane of the protocol, but to combat selfishness completely, the control plane should also be properly incentivized so that selfish nodes restrain themselves from adverse actions, such as tampering protocols headers, violating protocol semantics, faking identities, cheating shared metrics, and colluding with others. In future, we plan to address these issues.

Acknowledgment

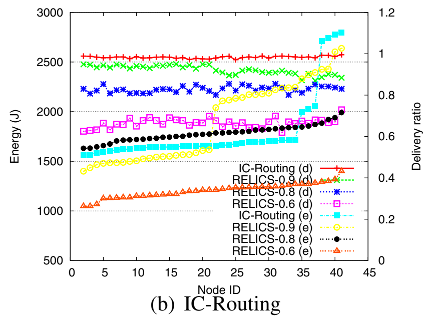
We thank our shepherd Kevin Almeroth and anonymous reviewers for their thoughtful suggestions. The work was funded in part by Fulbright S&T Fellowship, in part by ONR grant N00014-10-1-0172 and in part by NSF grant 0626825.

References

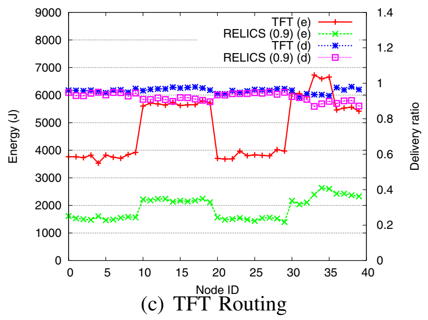
[1] J. J. Jaramillo and R. Srikant, "DARWIN: distributed and adaptive reputation mechanism for wireless ad-hoc networks," in *Proc. of MobiCom*, 2007.
[2] R. Mahajan, M. Rodrig, D. Wetherall, and J. Zahorjan, "Sustaining cooperation in multi-hop wireless networks," in *Proc. of NSDI*, 2005.
[3] V. Srinivasan, P. Nuggehalli, C. F. Chiasserini, and R. R. Rao, "Cooperation in wireless ad hoc networks," in *Proc. of INFOCOM*, 2003.
[4] P. Marbach and Y. Qiu, "Cooperation in wireless ad hoc networks: a market-based approach," *IEEE/ACM Trans. Netw.*, vol. 13, no. 6, pp. 1325–1338, 2005.
[5] B. Cohen, "Incentives build robustness in bit-torrent," in *Proc of 1st Workshop on Economics of Peer-to-Peer Systems*, 2003.
[6] M. Piatek, T. Isdal, T. Anderson, A. Krishnamurthy, and A. Venkataramani, "Do incentives build robustness in bittorrent?" in *Proc. of NSDI*, 2007.



(a) PROPHET



(b) IC-Routing



(c) TFT Routing

Fig. 10. Comparison with PROPHET, IC-Routing and TFT Routing. Legends: (e) = energy, (d) = delivery ratio

[7] V. Vishnumurthy, S. Chandrakumar, S. Ch, and E. G. Sirer, "KARMA: A secure economic framework for peer-to-peer resource sharing," in *Proc. of Workshop on the Economics of Peer-to-Peer Systems*, 2003.

[8] B. Chen and M. C. Chgan, "MobiCent: a credit-based incentive system for disruption tolerant network," Center for Internet Research, Natl. Univ. of Singapore, Tech. Rep., 2009.

[9] L. Buttyan, L. Dora, M. Felegyhazi, and I. Vajda, "Barter-based cooperation in delay-tolerant personal wireless networks," in *Proc. of IEEE WoWMom*, 2007.

[10] X. Xie, H. Chen, and H. Wu, "Bargain-based stimulation mechanism for selfish mobile nodes in participatory sensing network," in *Proc. of IEEE SECON*, 2009.

[11] Q. He, D. Wu, and P. Khosla, "SORI: A secure and objective reputation-based incentive scheme for ad-hoc networks," in *Proc. of IEEE WCNC*, 2004.

[12] S. Zhong, J. Chen, and R. Yang, "SPRITE: A simple cheat-proof, credit-based system for mobile ad-hoc networks," in *Proc. of INFOCOM*, 2002.

[13] S. Zhong, L. E. Li, Y. G. Liu, and Y. R. Yang, "On designing incentive-compatible routing and forwarding protocols in wireless ad-hoc networks: an integrated approach using game theoretic and cryptographic techniques," *Wireless Networks*, vol. 13, no. 6, pp. 799–816, 2007.

[14] F. Wu, T. Chen, S. Zhong, L. E. Li, and Y. R. Yang, "Incentive-compatible opportunistic routing for wireless networks," in *Proc. of ACM/IEEE MobiCom*, 2008.

[15] U. Shevade, H. H. Song, L. Qiu, and Y. Zhang, "Incentive-aware routing in DTNs," in *Proc. of IEEE ICNP*, 2008.

[16] A. Lindgren, A. Doria, and O. Schelén, "Probabilistic routing in intermittently connected networks," *Mobile Computing and Communications Review*, vol. 7, no. 3, pp. 19–20, July 2003.

[17] J. Burgess, B. Gallagher, D. Jensen, and B. N. Levine, "MaxProp: Routing for vehicle-based disruption-tolerant networks," in *Proc. of INFOCOM*, 2006.

[18] M. S. Uddin, H. Ahmadi, T. Abdelzaher, and R. Kravets, "A low-energy multicopy inter-contact routing protocol for disaster response networks," in *Proc. of IEEE SECON*, 2009.

[19] V. Erramilli, M. Crovella, A. Chaintreau, and C. Diot, "Delegation forwarding," in *Proc. of ACM MobiHoc*, 2007.

[20] T. Spyropoulos, K. Psounis, and C. Raghavendra, "Spray and wait: an efficient routing scheme for intermittently connected mobile networks," in *Proc. of SIGCOMM's WDTN*, 2005.

[21] S. Jain, M. Demmer, R. Patra, and K. Fall, "Using redundancy to cope with failures in a delay tolerant network," *SIGCOMM Comp Comm Review*, vol. 35, no. 4, pp. 109–120, 2005.

[22] B. N. Vellambi, R. Subramanian, F. Fekri, and M. Ammar, "Reliable and efficient message delivery in delay tolerant networks using rateless codes," in *Proc. of MobiOpp*, 2007.

[23] Y.-C. Hu, D. B. Johnson, and A. Perrig, "SEAD: Secure efficient distance vector routing for mobile wireless ad hoc networks," in *Proc. of IEEE WMCSA*, 2002.

[24] L. Subramanian, V. Roth, I. Stoica, S. Shenker, R. H. Katz, and Y. H. Katz, "Listen and whisper: Security mechanisms for BGP," in *Proc. of NSDI*, 2004.

[25] Opportunistic Network Environment (ONE) simulator, "http://www.netlab.tkk.fi/ jo/dtn/index.html."

Appendix

The choices for η_1 and η_2 (Algorithm 2) depend on how each node responds to energy rate update. The constant η_1 determines the increment of energy rate at every update, when delivery ratio remains below the threshold. Let $e(k)$ be the energy rate and $r(k)$ be the p-rank of a certain node at k-th round of update (i.e., k-th hour). While increasing energy rate, we expect that both p-rank and delivery ratio rise. Let k_u be the *upward settling time*, the time by which the p-rank of a node gradually rises to $\frac{1}{2}$ from its initial value 0. We assume that rank increment happens in a linear fashion, given by, $r(k) = \frac{k}{2k_u}$ $(0 \le k \le k_u)$. We have the following equation:

$$\begin{aligned}
e(k) &= e(k-1) + \eta_1(1 - r(k)) \\
&= e(k-2) + \eta_1(1 - r(k-1) + \eta_1(1 - r(k)) \\
&= e(k-2) + \eta_1(2 - (r(k-1) + r(k)) \\
&= e(0) + \eta_1 \left(k - \sum_{i=1}^{k} r(i) \right) \\
e(k_u) &= e(0) + \eta_1 \left(k_u - \sum_{i=1}^{k_u} \frac{k}{2k_u} \right) \\
e(k_u) &= e(0) + \frac{1}{4}\eta_1(3k_u - 1)
\end{aligned}$$

We set $k_u = 5$ Hr, allowing nodes 5 hours to raise their p-rank to $\frac{1}{2}$. Assuming each node has a battery with a total capacity of 1400 A-Hr or 4900J to survive 48 hours, each node can expend energy at a rate $4900/48 \approx 100$ J/Hr. If nodes are allowed to deplete energy at this rate once p-rank is raised to $\frac{1}{2}$, we can compute η_1 as follows:

$$\begin{aligned}
\eta_1 &= \frac{4e(k_u)}{3k_u - 1} = \frac{4 \times 100}{3 \times 5 - 1} \\
&= 28.57
\end{aligned}$$

Once delivery ratio exceeds the threshold, energy rate is reduced at each update by the following equation:

$$\begin{aligned}
e(k) &= e(k-1) \times (1 - \eta_2 \times r(k)) \\
&= e(0) \times \Pi_{i=1}^{k}(1 - \eta_2 \times r(i))
\end{aligned}$$

We define k_d as the *downward settling time* by which energy rate becomes 10% of the initial value while keeping the p-rank value unchanged at $r(0)$. This leads to:

$$\begin{aligned}
(1 - \eta_2 r(0))^{k_d} &= \frac{e(k_d)}{e(0)} = 0.1 \\
k_d \ln(1 - \eta_2 r(0)) &= \ln(0.1) = -2.3 \\
\eta_2 &= \frac{1}{r(0)} \times (1 - e^{\frac{-2.3}{k_d}})
\end{aligned}$$

We assume $k_d = 5$ Hr for $r(0) = 1$. So, $\eta_2 = 0.37$.