International Journal of Emerging Technology and Advanced Engineering
Website: www.ijetae.com (ISSN 2250-2459 (Online), An ISO 9001:2008 Certified Journal, Volume 3, Special Issue 1, January 2013)

International Conference on Information Systems and Computing (ICISC-2013), INDIA.

DETECTING ROUTING MISBEHAVIOR AND ENHANCING THE SECURITY IN DISRUPTION TOLERANT NETWORK

Gunasundari.S[1], Vinothkumar.G[2]

[1,2]*M.E Network Engineering, Vel Tech Multi Tech Dr.Rangarajan Dr.Sakunthala Engineering College Avadi, Chennai.*

[1]gunasundari20@gmail.com, [2]gvcse79@gmail.com

Abstract

Disruption tolerant network exploit the intermittent connectivity between mobile nodes to transfer the data.In such networks, selfish or malicious nodes may drop received packets. Such routing misbehavior reduces the packet delivery ratio and wastes system resources such as power and bandwidth. Although techniques have been proposed to mitigate routing misbehavior in mobile ad-hoc networks, they cannot be directly applied to DTNs because of intermittent connectivity between nodes. To address the problem, we propose a distributed scheme to detect packet dropping in DTNs. In our scheme, a node is required to keep a few signed contact records of its previous contacts, based on which the next contacted node can detect if the node has dropped any packet. Since misbehaving nodes may misreport their contact records to avoid being detected, a small part of each contact record is disseminated to a certain number of witness nodes, which can collect appropriate contact records and detect the misbehaving nodes. We also propose a scheme to mitigate routing misbehavior by limiting the number of packets forwarded to the misbehaving nodes. Trace-driven simulations show that our solutions are efficient and can effectively mitigate routing misbehavior.

Keywords— **Detection, disruption tolerant networks, mitigation, routing misbehavior, security.**

I. INTRODUCTION

DTN routing usually follows “store-carry-forward;” i.e., when a node receives some packets, it stores these packets in its buffer, carries them around until it contacts another node, and then forwards the packet. In DTNs, a node may misbehave by dropping packets even when it has sufficient buffers. Routing misbehavior can be caused by selfish nodes that are unwilling to spend resources such as power and buffer on forwarding packets of others, or caused by malicious nodes that drop packets to launch attacks. Routing misbehavior will significantly reduce the packet delivery ratio and waste the resources of the mobile nodes that have carried and forwarded the dropped packets.

We simulate the effects of routing misbehavior in two popular DTN routing algorithms, SimBet [4] and Delegation [5] based on the Reality trace [6]. SimBet is a *forwarding-based* algorithm where a packet only has one replica. Delegation is a *replication-based* algorithm where a packet may have multiple replicas.

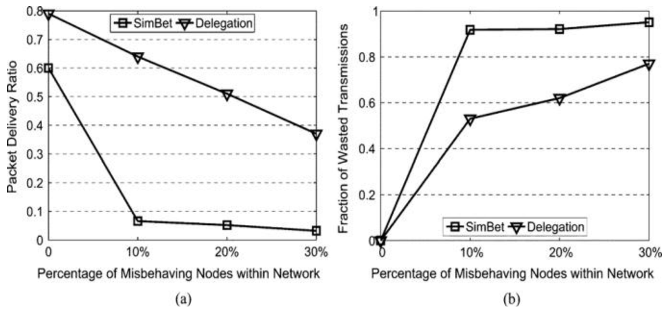


Fig 1: Effects of routing misbehavior with simbet and delegation

In this paper, we address routing misbehavior in DTNs by answering two questions: how to detect packet dropping and how to limit the traffic flowing to the misbehaving nodes. We first propose a scheme which detects packet dropping in a distributed manner. In this scheme, a node is required to keep previous signed contact records such as the buffered packets and the packets sent or received, and report them to the next contact node which can detect if the node has dropped packets.

II. PRELIMINARIES

2.1. Network and Routing Model

Each node has two buffers: one has unlimited space and it is used to store its own packets. Another one is used to store the packets received from other nodes. We assume the network is loosely synchronized; i.e., any two nodes should be in the same time slot at any time. Since the intercontact time is usually at the scale of minutes or hours, the time slot can be at the scale of one minute.

2.2. Security Model

There are two types of nodes: Misbehaving nodes and normal nodes. A misbehaving node drops the received packets even if it has available buffers, but it does not drop its own packets. It may also drop the control messages of our detection scheme. We assume a small number of misbehaving nodes may collude to avoid being detected, and they may synchronize their actions via out-band communication channels. A normal node may drop packets when its buffer overflows, but it follows our protocol.

Some DTN applications, each packet has a certain lifetime, and then expired packets should be dropped whether or not there is buffer space. In identity-based authentication, only the offline trusted private key generator can generate a public/private key pair, so a misbehaving node itself cannot forge node identifiers.

2.3. Overview of Our Approach

The misbehaving node is required to generate a *contact record* during each contact and report its previous contact records to the contacted node. Based on the reported contact records, the contacted node detects if the misbehaving node has dropped packets. The misbehaving node may misreport to hide its misbehavior, but forged records cause inconsistencies which make misreporting detectable. To detect misreporting, the contacted node also randomly selects a certain number of *witness nodes* for the reported records and sends a summary of each reported record to them when it contacts them.

It reduces the data traffic that flows into misbehaving nodes in two ways:

I. If a misbehaving node misreports, it will be blacklisted and will not receive any packet from other nodes;
II. if it reports its contact records honestly, its dropping behavior can be monitored by its contacted nodes, and it will receive much less packets from them.

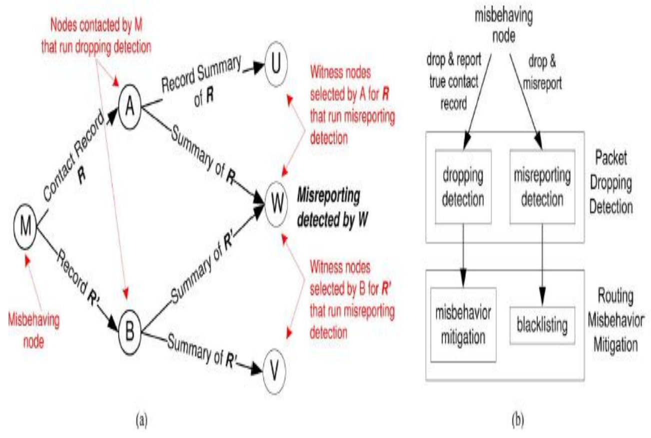

Nodes contacted by M that run dropping detection
Contact Record R
Record Summary of R
U
A
Summary of R
Witness nodes selected by A for R that run misreporting detection
M
W
Misreporting detected by W
Record R'
Summary of R'
B
Witness nodes selected by B for R' that run misreporting detection
Misbehaving node
Summary of R'
V
(a)
misbehaving node
drop & report true contact record
drop & misreport
dropping detection
misreporting detection
Packet Dropping Detection
misbehavior mitigation
blacklisting
Routing Misbehavior Mitigation
(b)

Fig 2: Overview of our approach, (a) Detecting routing misbehaving nodes R and R' (b) misbehavior mitigation

2.4. Terms and Notations

We use N, N_i , N_j , etc. to denote a node, and use or to denote a misbehaving node. We use SIG {*} to denote a node's signature over the content in the bracket. H{*} denotes a cryptographically strong hash function, and " | " denotes the concatenation operation.

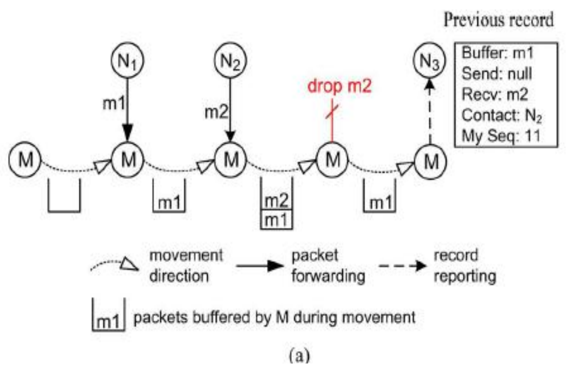
III. PACKET DROPPING DETECTION

We first introduce the basic idea and then describe how to detect packet dropping and how to detect misreporting.

3.1 Basic Idea

When two nodes contact, they generate a contact record which shows when this contact happens, which packets are in their buffers before data exchange, and what packets they send or receive during the data exchange. The record also includes the unique sequence number that each of them assigns for this contact.

To detect misreporting, for each contact record that a normal node generates with other nodes, the normal node selects w witness nodes and transmits the record summary to them. The summary only includes a part of the record necessary for detecting the inconsistency caused by misreporting.



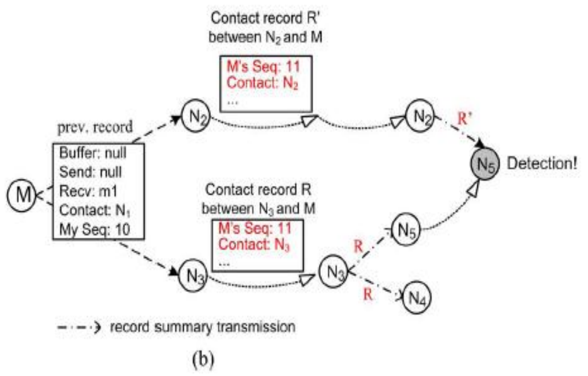


Fig 3: Packet dropping and misreporting detection. In (b) M reports the same contact record with N2 and N3 as its previous.

3.2 Contact Record and Record Summary

Suppose a contact happens between node N_i and N_j at time t. Without loss of generality, suppose i<j. Let BV_i and BV_j denote the vector of packets buffered by N_i and N_j before this contact, respectively. Let RV_i and RV_j denote the identifiers of the packets received by N_i and N_j during this contact, respectively. Then the record of this contact that N_i will store and report is as follows:

$R=N_i, sn_i, N_j, sn_j, t, BV_i, RV_i, RV_j, sig_i^1, sig_j^1, sig_i^2, sig_j^2$

3.3 Packet Dropping Detection

In this contact, the two nodes also exchange their current vector of buffered packets. In this way, one node knows the two sets of packets the other node buffers at the beginning of the previous contact and the beginning of the current contact, which are denoted by S_{beg} and S_{end}, respectively.

A misbehaving node may drop a packet but keep the packet ID, pretending that it still buffers the packet. However, the next contacted node may be a better relay for the dropped packet according to the routing protocol. Thus, the next contacted node can easily detect this misbehavior and will not forward packets to this misbehaving node.

3.4 Misreporting Detection

M cannot modify the true record since it is signed by the previous contacted node. Also, M cannot forge a contact record because it does not know the private key of any other node. Thus, the only misreporting it can perform is to replay an old record generated before the previous contact.

1) Consistency Rules: There exist two simple rules which are obeyed by normal nodes but violated by misreporting nodes:

• Rule 1: Use a unique sequence number in each contact.
• Rule 2: For two records signed by the same node, the record with a smaller contact time also has a smaller sequence number.

2) Detection: To detect the inconsistency caused by misreporting, for each contact record generated and received in a contact, a node selects random nodes as the witness nodes of this record, and transmits the summary of this record to them when it contacts them.

3) Alarm: After detection, the witness node floods an alarm to all other nodes. The alarm includes the two inconsistent summaries. When a node receives this alarm, it verifies the inconsistency between the included summaries and the signature of the summaries. If the verification succeeds, this node adds the appropriate misreporting node into a blacklist and will not send any packets to it.

IV. Iv Dealing With Collisions

4.1 Misreporting with Collisions

Suppose a misbehaving node M drops a packet. If M reports the true record of its previous contact to the next contacted (normal) node, the dropping can be detected. To hide the dropping, M can forge a contact with its colluder M’ after dropping the packet, and report the falsified contact record to the next contacted node.

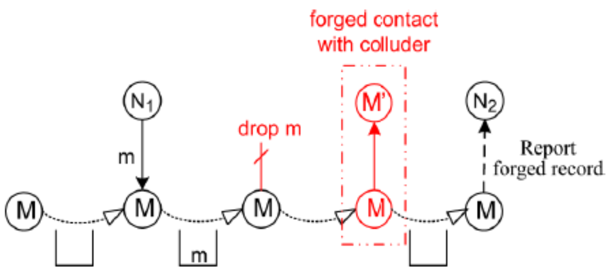


Fig 4: Two colluding nodes M and M’ try to hide dropping of packet m by forging a contact

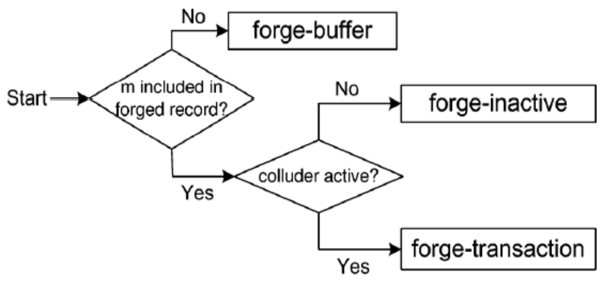


Fig 5: Decision tree based exploration misreporting

M and M' generate two different records for the same forged contact. This misreporting behavior is referred to as *forge-transaction.*

4.2 Forge Buffer

Forge-buffer can hide packet dropping because the normal nodes contacted by the misbehaving node and its colluder cannot see a long-enough contact history of them. To address this problem, the basic scheme is extended such that each node is required to report more than one previous record in its contact. The number of records the node needs to report is determined by its current *report window*.

In forge-buffer, since the misreporting node is required to report the recent records with at least distinct nodes but it only has r-1 colluders, the record of its previous real contact has to be reported.

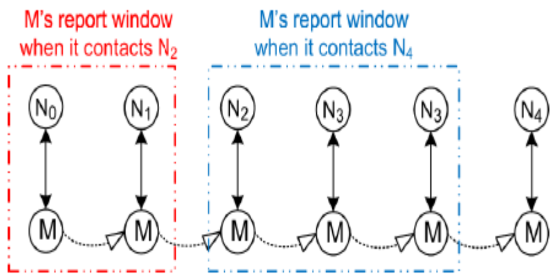

M's report window when it contacts N2
M's report window when it contacts N4
N0
N1
N2
N3
N3
N4
M
M
M
M
M
M

Fig 6: Examples of report window with r=2.

4.3 Forge In-Active

To address forge-inactive, a forwarding rule is added which requires that packets can only be forwarded to an *active node*, which is defined as the node that contacts at least different nodes within the recent time interval T_{active}.

The activeness of a node can be easily proved by the recent contact records it reports to other nodes. During a contact one node is supposed to be able to obtain the contacted node's most recent contact records with nodes.

4.4 Record and Summary Deletion

A node deletes the record that it generates in a contact after the contact has been purged out of its report window, probably after a few contacts. It deletes the records received from the contacted node right after this contact, since these received records are only used to check if the contacted node has dropped packets recently.

4.5 Discussion

After M tells the next contacted node that it has forwarded the dropped packet m to the colluder M', M' may tell its own next contacted node that it has forwarded m to M' or another colluder M''. Since the forged contacts between the two colluding nodes are within their report window, their next contacted node can detect that is forwarded back and- forth between the two colluding nodes.

V. ANALYSIS OF MISREPORTING DETECTION

5.1 Models and Notations

Let n denote the number of nodes in the network, and q denote the proportion of misbehaving nodes. Without loss of generality, the following analysis considers forge-transaction in which misreporting node M and its colluder M' generate two inconsistent contact records *R* and *R*'. M reports *R* to its next two contacted nodes which constitute a set *S* , and M' reports *R*' to its next two contacted nodes which constitute a set *S*'.

5.2 Detection Probability

The detection succeeds when: 1) there is at least one normal node in *S* and *S*' ; 2) the normal nodes in *S* and those in *S*' select one or more common witness nodes for the two records; and 3) at least one common witness is a normal node. Obviously, the three conditions hold independently.

It shows the minimum w required to detect misbehaving nodes with probability not less than 0.95 when different numbers of misreporting instance k can be tolerated. Basically w, will be significantly reduced if k increases.

5.3 Detection Delay

Detection delay is defined as the elapsed time from when *R* and *R*' have been reported to the nodes in set *S* and *S*' , respectively, to the time when the misreporting instance is detected by a witness node.

Now we have an upper bound for the expected detection delay

$$E(T_d) \leq (3 / 2\lambda)$$

5.4 False Positive

Our misreporting detection scheme does not have false positive when detecting replay-record and forge-transaction. An attacker may spoof another normal node and insert new inconsistent contact records or record summaries on behalf the normal node, so that when the witness receives the inconsistent record summaries it detects the normal node as a misreporting attacker.

5.5 Cost

We analyze the computation cost per contact in terms of signature generations and verifications. In each contact, the two nodes invocate six signature generations and verifications to generate the contact record. The reported records require 8r signature verifications in total. To sum up, each contact requires 6 signature generations and at most 12r +6 signature verifications. This cost is low considering that signature verification is usually much faster than signature generation.

VI. ROUTING MISBEHAVIOR MITIGATION

To mitigate routing misbehavior, we try to reduce the number of packets sent to the misbehaving nodes. If a node is detected to be misreporting, it should be blacklisted and should not receive packets from others. However, if a misbehaving node does not misreport, we cannot simply blacklist it because it is dropping packets, since a normal node may also drop packets due to buffer overflow.

6.1 FP Maintenance

A misbehaving node may “forward” packets to or “receive” packets from its colluder via the out-band channel, to deceive its contacted nodes to increase the FP for it. To address this problem, node N_i keeps a *gauge list* for every other node N_j it has contacted, which includes the nodes contacted by N_j that N_i observes in the recent contacts with N_j .

6.2 Dealing with Packet Dropping

Suppose node N_i has selected a set of packets to forward to its contacted node N_j . Let S_{opt} denote this set of packets. Our scheme can effectively mitigate routing misbehavior, since if a node is misbehaving and frequently drops packets, it will be assigned a low FP by its contacted nodes and receive few packets from them.

VII. PERFORMANCE EVALUATION

7.1 Experiment Setup

In our simulations, misbehaving nodes are either *randomly deployed* or *selectively deployed*.

In the latter case, they are the high-connectivity nodes which have the most frequent contacts with others. Misbehaving nodes drop all or part of the packets they receive from others. In default, each normal node generates one packet (with size 10 KB) every day to a random destination. A misbehaving node does not generate packets as done in [7] and [5]. Each node has buffer size of 5 MB and bandwidth of 2Mb/s. Results are averaged over 10 runs with different random seeds.

7.2 Routing Algorithms

SimBet [4]: a forwarding-based routing algorithm. It calculates a metric using two social measures (similarity and betweenness), and a packet is forwarded to a node if that node has higher metric than the current one.
Delegation [5]: a replication-based routing algorithm, where the receiving node replicates the packet to a neighbor if the neighbor has the highest utility it has seen. We use the contact frequency with the destination as the utility metric.

7.3 Metrics

Packet delivery ratio: The percentage of packets delivered to their destinations out of all generated packets.
Number of wasted transmissions: In forwarding-based routing, a transmission is wasted if the transmitted packet is dropped by misbehaving nodes before reaching the destination;
Detection rate: The percentage of misreporting nodes detected by normal nodes.
Detection delay: The time needed for misreporting to be detected.
Bytes transmitted per contact: The number of bytes transmitted in a contact for control.

7.4 Experimental Results

We compare our scheme to two solutions: one is called *No-Defense*, which does not deal with routing misbehavior; the other one is the *optimal* scheme, which assumes that all misbehaving nodes are known and no packet will be forwarded to them.

1. Routing Misbehavior Mitigation:

Fig. 8 shows the comparison results on the Reality trace. Generally speaking, our scheme performs much better than No-Defense. When SimBet is used as the routing algorithm, as shown in Fig. 8(a), the packet delivery ratio of all three schemes decreases as the percentage of misbehaving nodes increases, because fewer nodes can be used for packet forwarding.

For similar reasons, our scheme has a much lower number of wasted transmissions than No-Defense, as shown in Fig. 8(b).

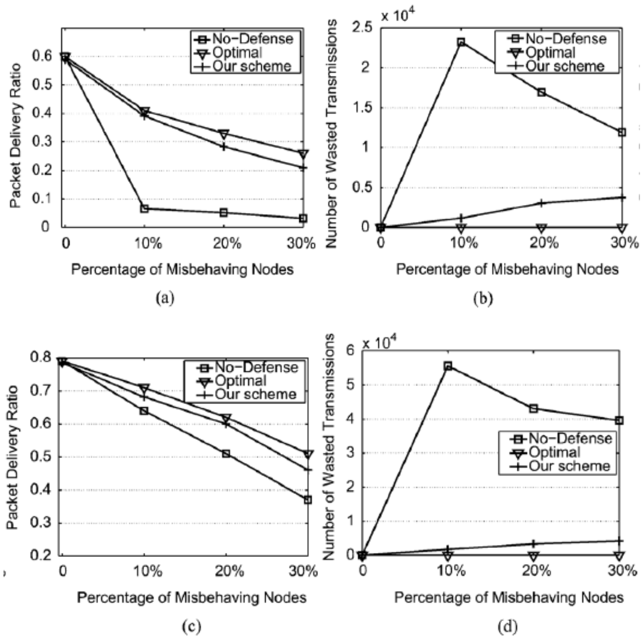


Fig 7: Comparison results when misbehaving nodes are selectively deployed to high connectivity nodes. (a) simbet (b) simbet (c) Delegation (d) Delegation

2. *Misreporting Detection*

In this group of simulations, 10 pairs of misbehaving nodes (i.e., 20 in total) launch forge-transaction independently. As shown in Fig. 10, the detection probability in the simulations is higher than the analytical lower bound, but the difference is small which means the analytical result is a good approximation. The detection delay in the simulations is lower than the analytical upper bound. As shown in Fig. 11(a), the detection rate increases as the number of witness nodes for each record summary increases.

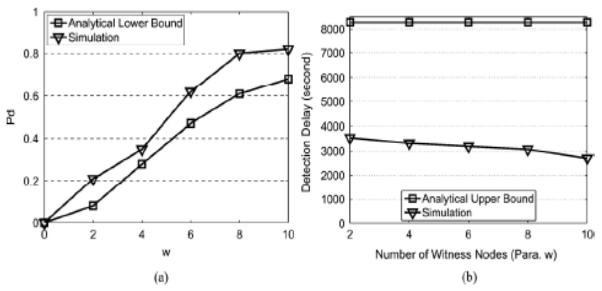


Fig 8: Comparison analysis of simulation results on the Detection probability and detection delay.

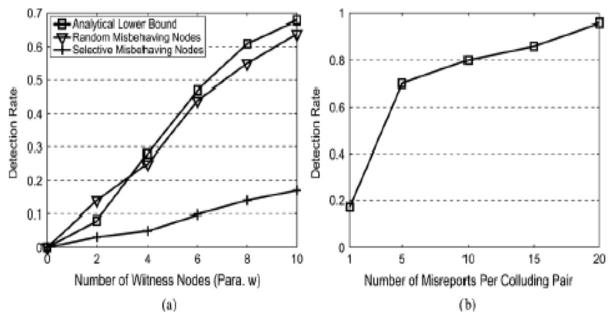


Fig 9: Detection Rate of our reality trace (a) each collusion pair misreports once (b) parameter w=10

3. *Cost*

The size of record and summary is set as follows: node ID, sequence number, and timestamp have 4 B each; a hash has 16 B; a signature has 40 B [30]. In default, w=4 and T_{delete}=30 days. Delegation is used and all nodes are normal. The storage overhead increases significantly with the parameter w and T_{delete} since record summaries are stored at more nodes and for a longer time. However, the storage overhead only increases slowly with the packet generation rate.

VIII. CONCLUSION

In this paper, we presented a scheme to detect packet dropping in DTNs. The detection scheme works in a distributed way; Moreover, the detection scheme can effectively detect misreporting even when some nodes collude. Analytical results on detection probability and detection delay were also presented. Based on our packet dropping detection scheme, we then proposed a scheme to mitigate routing misbehavior in DTNs. The proposed scheme is very generic and it does not rely on any specific routing algorithm.

REFERENCES

[1] E. Daly and M. Haahr, “Social network analysis for routing in disconnected delay-tolerant manets,” in Proc. ACM MobiHoc, 2007, pp. 32–40.

[2] J. Burgess, G. D. Bissias, M. Corner, and B. N. Levine, Surviving attacks on disruption-tolerant networks without authentication,” in Proc. ACM MobiHoc, 2007, pp. 61–70.

[3] H. Yang, J. Shu, X. Meng, and S. Lu, “Scan: Self-organized network- layer security in mobile ad hoc networks,” IEEE J. Sel. Areas Commun., vol. 24, no. 2, pp. 261–273, 2006.

[4] B. Awerbuch, D. Holmer, C.-N. Rotaru, and H. Rubens, “An on-demand secure routing protocol resilient to byzantine failures,” in Proc. ACM WiSe, 2002, pp. 21–30.

[5] W.Gao,Q.Li, B. Zhao, andG. Cao, “Multicasting in delay tolerant networks: A social network perspective,” in Proc. ACM MobiHoc, 2009, pp. 299–308.

[6] A. Seth, D. Kroeker, M. Zaharia, S. Guo, and S. Keshav, “Lowcost communication for rural internet kiosks using mechanical backhaul,” in ACM Proc. Mobicom, 2006, pp. 334–345.

[7] V. Conan, J. Leguay, and T. Friedman, “Characterizing pairwise intercontact patterns in delay tolerant networks,” ACM Autonomics, pp. 19:1–19:9, 2007.