Explaining Routing Performance in Disruption Tolerant Networks

Brian Gallagher, David Jensen, and Brian Neil Levine
Dept. of Computer Science, University of Massachusetts, Amherst MA 01003
{bgallag, jensen, brian}@cs.umass.edu

Abstract

Many routing algorithms for both traditional and ad hoc networks require a complete and contemporaneous path of peers from source to destination. Disruption Tolerant Networks (DTNs) attempt to deliver messages despite a frequently disconnected link layer (e.g., due to peer mobility, limited communication range, and power management limitations). While several algorithms have been proposed for routing in DTNs, this has not yet led to an understanding of the fundamental issues underlying routing performance in these networks.

In this paper we explain the performance of routing algorithms for DTNs in terms of their ability to utilize a set of three no-cost drop criteria. *The criteria are necessary and sufficient for identifying messages that may be dropped without degrading the overall delivery rate. The criteria identify whether a route exists with sufficient bandwidth, whether a message has been delivered already, and whether some other peer will deliver the message. We also use the criteria to design a new routing algorithm that we call* NoCostDrop, *which appears to be the first routing algorithm to take advantage of all three criteria. We show that* NoCostDrop *outperforms existing algorithms over a wide range of network conditions. Most novel in our approach is the use of a distributed list of delivered messages, which can easily be combined with existing routing algorithms to improve their performance.*

1 Introduction

Despite the ubiquity of mobile computing devices, users are rarely able to take full advantage of the communications resources that they carry. IP destinations must be connected to the Internet at the moment a message is sent, and the path of routers that lead from source to destination must also be contemporaneously connected. Routing algorithms for ad hoc networks have not seen wide use or deployment because of these limitations. The density of wireless access points and always-on mobile peers has not reached the necessary critical mass to ensure contemporaneous end-to-end connectivity.

To address these limitations, there is a growing body of work on *disruption tolerant networks* (DTNs), which are networks that allow intermittent link-layer connections among peers. Solutions are beginning to appear for the provision of message forwarding services in DTNs.

Recently, a handful of DTN routing algorithms have been developed [VB00, DFL01, GV03, LDS04, SG04, BBL05]. However, to our knowledge, only one study has investigated the underlying issues of routing performance in DTNs [JFP04]. In this study, Jain et al. focus on scenarios in which topology dynamics are known in advance, routing is performed in the traditional store-and-forward fashion, and varying amounts of external knowledge are available to routing algorithms.

Here, we focus on the more general DTN routing problem, making the assumptions that external information is unavailable, that all peers in the system are mobile, and that contemporaneous paths of length greater than one are essentially non-existent. In this context, the traditional routing decision of choosing the next edge along which a message should be forwarded does not apply. Instead, routing consists of a series of message exchanges between pairs of peers as communication links are established and destroyed. Since each peer has a finite amount of space for storing others' messages, routing decisions amount to determining which messages to forward when meeting with a given peer and which messages to drop when a peer's buffer becomes full.

This paper makes several contributions towards understanding the factors affecting the performance of DTN routing algorithms. Our results are based on the identification of three *no-cost drop criteria* that are necessary and sufficient for identifying messages that may be dropped without degrading the overall delivery rate. These criteria identify whether a route exists with sufficient bandwidth, whether a message has been delivered already, and whether some other peer will deliver the message.

We explain the quantitative performance of existing DTN routing algorithms in terms of their ability to exploit each of the no-cost drop criteria. For example, no existing algorithm explicitly attempts to determine if a message has been delivered already as a criterion for dropping it from a buffer.

We propose a new routing algorithm, designed to exploit all three no-cost drop criteria, and we show em-

pirically that this algorithm outperforms previous algorithms under a variety of connectivity assumptions. The most novel part of our algorithm is the use of *delivery lists*. We show that their use alone results in dramatic increases in performance.

Additionally, we show how the underlying model of connections among peers affects the relative performance of certain algorithms. Notably, delivery lists improve performance independent of the underlying connection model.

This work is organized around the above goals. In the next section, we review related work. In Section 3, we describe the problem of routing in DTNs in more detail, including our assumptions. Section 4 explains the no-cost drop criteria. Section 5 describes the algorithms that we evaluate, including the new *NoCostDrop* and *DropDelivered* algorithms, and classifies them according to their use of the no-cost drop criteria. In Section 6 we describe our evaluation model and in Section 7 we discuss the details of our evaluation and results. Finally, in Section 8, we present our conclusions and describe some interesting new directions for future research that arise from our work.

2 Related Work

Previous work in this area is based on various assumptions regarding connectivity and the availability of environmental knowledge and control. In general, algorithms based on stronger assumptions have superior performance, but their applicability is more limited.

A number of the proposed routing algorithms for DTNs make weak assumptions and are therefore widely applicable. In general, these algorithms are based solely on deciding which messages to forward during a meeting with a given peer and which messages to drop when buffers reach capacity. The *epidemic routing* algorithm, proposed by Vahdat and Becker, manages buffers as *first-in-first-out* (*FIFO*) queues when finite buffers are assumed and requires no buffer management under the infinite buffer assumption [VB00]. The *Drop Least Encountered (DLE)* algorithm, proposed by Davis et al., was the first of a number of algorithms based on the idea of dropping messages with the lowest likelihood of delivery [BBL05, DFL01, GV03, LDS04, SG04]. All of these algorithms approximate delivery likelihood as the likelihood of a delivery path existing.

Others have taken a more pro-active approach to routing in DTNs, made possible by stronger assumptions such as knowledge of geographic location, prior knowledge of connectivity patterns, and control over peer movement [BBL05, GV03, JFP04, LR00, SG04, ZA03, ZA04].

3 Generalized Routing Scenario

In this paper, we address the most general DTN routing problem. We assume no prior knowledge of network connectivity, no control over peer movement, no knowledge of geographic location, and no solely stationary peers. We assume that each peer has an infinite buffer for messages that they originate, but a fixed-size buffer for messages originated by other peers.

In general, a DTN routing scenario proceeds roughly in four stages.

1. **Await transfer opportunities**. Each peer generates messages destined for other peers in the system and carries them until it comes within communication range of another peer.
2. **Exchange message headers**. When two peers meet, they exchange lists of the messages they carry.
3. **Apply routing algorithm**. Each peer takes the union of the two message lists, considers the available buffer space, and makes decisions about which old messages to drop and which new messages to accept.
4. **Exchange contents of selected messages**. For each message to be exchanged, a copy is created and passed between peers.

Once messages are exchanged, each peer continues to carry its new set of messages until the next meeting occurs. A peer will continue to forward a message to any number of other peers until its copy of the message times out or is delivered, or until the message is dropped due to a full buffer.

The message exchange process can be performed in fewer steps to account for short transmission opportunities. However, we are not concerned here with the properties of the link layer when a transfer opportunity occurs. We assume that when the channel is up, it is reliable and of sufficient duration.

4 The No-Cost Drop Criteria

In this section, we present a set of three *no-cost drop criteria*, which are necessary and sufficient for identifying messages that may be dropped without degrading the overall delivery rate. We later use these criteria to explain the performance of existing DTN routing algorithms and to design the new *NoCostDrop* algorithm.

Any DTN routing algorithm reduces to a series of local decisions about which messages forward to other peers and which messages to drop when buffers reach capacity. The most important performance metric here is the delivery rate, which depends on which messages are dropped. Accordingly, it is important to observe that a

peer can drop certain messages without any cost in terms of the overall delivery rate.

Let a DTN be composed of *P* peers. A peer, *p*, may drop a copy of message *m* without reducing the overall delivery rate if and only if it meets one of the following criteria:

1. ***Cannot-Deliver*** - Peer *p* cannot deliver the copy of message *m* to its destination through any set of intermediate peers no matter how long *p* chooses to hold the copy. In other words, no route with sufficient bandwidth will exist between *p* and *m*'s destination during the lifetime of message *m*.
2. ***Is-Delivered*** – A copy of message *m* has already been delivered to its destination.
3. ***Delivered-If-Dropped*** – No copy of message *m* has been delivered, but some copy of *m* will be delivered even if peer *p* drops its copy.

It is easy to prove that these three criteria are necessary and sufficient to describe the set of messages that can be dropped by a peer without degrading the overall delivery rate. First, the three criteria are mutually exclusive; it is not possible for any message to meet more than one of them. Second, the only possibility not covered by the criteria is that *m* has not been delivered and *m* can be delivered, but only if *p* holds on to *m*. Clearly, dropping this type of message may affect the overall delivery rate.

Since the propagation of information in a DTN is relatively slow, a peer will generally not know the values of *Cannot-Deliver*, *Is-Delivered*, and *Delivered-if-Dropped* with certainty. However, we can use information available in the network to estimate the likelihood that each criterion is satisfied.

We define a DTN routing algorithm to have two components:

(1) One or more *estimators*. An estimator is simply a method for (explicitly or implicitly) estimating the likelihood that one or more of the no-cost drop criteria are satisfied.

(2) A *decision procedure*. A decision procedure takes information from one or more estimators as input and outputs routing decisions (i.e., decisions about which messages to forward and which messages to drop).

Seen in this context, what we describe here amounts to a routing framework in which estimators and decision procedures are interchangeable components that can be modified to suit a variety of applications. For instance, in Section 7.5 we discuss alternative estimators that make use of external sources of information (e.g., prior knowledge about networks or out-of-band communication).

5 Routing Algorithms

In this section, we classify existing algorithms according to their use of the criteria introduced in the previous section. In addition, we propose two new algorithms, *DropDelivered* and *NoCostDrop*. *NoCostDrop* is the first algorithm to exploit all three no-cost drop criteria.

5.1 Broadcast

A peer using the *Broadcast* algorithm floods all its messages to all other peers it meets. This algorithm assumes that peers have an infinite message buffer and messages are not dropped until they time out. This is equivalent to the epidemic routing algorithm described by Vahdat and Becker with no limit on buffer size [VB00]. *Broadcast* is included strictly for reference as a performance ceiling. This algorithm does not make use of any of the no-cost drop criteria.

5.2 Random

A peer using the *Random* algorithm floods all its messages to all other peers it meets. When a peer's buffer reaches capacity, it chooses messages for dropping at random. This algorithm does not make use of any of the no-cost drop criteria.

5.3 FIFO

A peer using the *first-in-first-out* (*FIFO)* algorithm floods all its messages to all other peers it meets. When a peer's buffer reaches capacity, it chooses messages for dropping based on the length of time messages have been in the peer's buffer. Messages that have been buffered longest are dropped first. This is equivalent to the epidemic routing algorithm with a buffer size limit [VB00].

The *FIFO* algorithm implicitly exploits both the *Is-Delivered* and *Delivered-if-Dropped* criteria. Suppose that peer *p* is carrying two messages, m_1 and m_2. Suppose also that we know m_1 has been in *p*'s buffer for 50 seconds and m_2 has been in *p*'s buffer for 5 seconds. In general, this also indicates that m_1 has been in the network longer than m_2. The longer that a message is in the network, the more chances we expect the source and intermediaries to have to create multiple message copies and spread them throughout the network. Thus, all else being equal, our expectation is that more peers will have received copies of m_1 than m_2. Therefore, in general, m_1 is more likely than m_2 to be delivered by some other peer even if dropped by *p* (*Delivered-if-Dropped*) and m_1 is also more likely than m_2 to have already been delivered (*Is-Delivered*).

5.4 DLE

Most work on DTN routing algorithms to date has been based on the idea of dropping messages with the lowest likelihood of delivery [DFL01, GV03, LDS04, SG04, BBL05]. We use *Drop Least-Encountered (DLE),* proposed by Davis et al., as an algorithm representative of this class [DFL01]. Using the *DLE* algorithm, each peer, *p*, maintains a *meeting value* for every other peer in the network, *q*. This value estimates the likelihood of a future path from *p* to *q*. Every time peer *p* meets peer *q*, *p* adds 1 to its current meeting value for *q*. In addition, *p* adds a portion of *q*'s meeting value for each other peer, *r*, to *p*'s current meeting value for *r*, capturing the likelihood of multi-hop paths as well as one-hop paths. Meeting values also degrade over time, so meetings that occur more regularly end up with higher values.

Note that *DLE*, as well as several of the other algorithms proposed in the literature, employ a technique that we refer to as *selective routing*, in which a peer with a lower likelihood of delivering message *m* will never accept *m* from a peer with a higher likelihood of delivering *m* [BBL05, DFL01, LDS04]. There are several variations on this technique, each of which we have found to exhibit very similar performance characteristics. For this work, we use the following variation: When peers *p* and *q* meet, *p* only offers messages to *q* for which *q* has a higher meeting value than *p*. Likewise, *q* only offers messages to *p* for which *p* has a higher meeting value than *q*. The exchange then proceeds as usual, each peer joining its list of buffered messages with the list of offerings from the other peer. If *p*'s buffer will not accommodate all messages in the combined list, the messages with the lowest meeting values are dropped (if buffered by *p*) or refused (if buffered by *q*).

DLE and similar algorithms exploit the *Cannot-Deliver* criterion indirectly by considering the likelihood that a path will exist along which a message could potentially be delivered. However, just because such a path exists does not mean that a message sent along that path will reach its destination. Messages may be dropped along the way due to congestion. In order to achieve the most accurate estimate of delivery likelihood, an estimator must consider not only the existence of a path, but also the capacity of that path.

Since DLE and similar algorithms ignore congestion and focus exclusively on path existence, we could say that they exploit *Cannot-Deliver* incompletely. It is easy to imagine situations in which certain peers are central to the network in the sense that they have regular meetings with a variety of other peers. These *hub* peers are heavily favored by *DLE* and similar algorithms since they are involved in many delivery paths. However, they may also be highly congested for exactly the same reason. In such a scenario, the performance of *DLE* and similar algorithms will suffer. Our results demonstrate this effect (see Section 7.3).

5.5 DropDelivered

Before we describe the details of the *DropDelivered* algorithm, we introduce a new technique for exploiting the *Is-Delivered* criterion: *delivery lists.*

5.5.1 Delivery Lists

To fully exploit the *Is-Delivered* criterion, a peer must know when messages it has forwarded have been delivered. Delivery lists provide a way for peers to notify each other of message deliveries.

Using this technique, each peer maintains a list of delivery notifications (message IDs) for messages that have already been delivered in the network. Initial lists can be created by each peer since a peer knows with certainty the messages that it has successfully received as a destination. Lists can then be exchanged between peers during meetings so that a global list propagates throughout the network. Then, when a peer receives a message that is on its delivery list, the peer may drop the message immediately.

The cost of maintaining a delivery list is that it reduces the buffer space available for messages. If we think of a buffer in terms of the number of messages it can hold, the cost of a delivery list of a given length depends on the size of message IDs relative to the size of messages. We assume that messages may be uniquely identified by a 128-bit ID (e.g., from an MD5 hash). In our experiments, we assume that messages range in size from 160 B to 50 KB.

We have observed empirically that the optimal proportion of total buffer space devoted to the delivery list depends on both the total buffer size and the average message size. In our evaluation, we fix this proportion to .15, a value that we have found to perform reasonably well over a range of parameter settings. Adjusting this proportion dynamically based on network characteristics would likely yield additional performance increases. However, this is beyond the scope of this paper.

We manage our delivery lists as *FIFO* queues. It is possible that some more sophisticated scheme might further improve the utility of this technique.

5.5.2 The DropDelivered Algorithm

The *DropDelivered* algorithm uses delivery lists to drop delivered messages from buffers as soon as possible. Whenever a peer receives a new message, it checks its delivery list. If the delivery list contains the new message, the message is dropped immediately and not buff-

ered. Whenever a peer adds a new message ID to its delivery list, the peer checks its buffer and immediately drops the corresponding message if it is buffered. If a peer must drop additional messages due to limited buffer space, messages are chosen at random for dropping.

5.6 NoCostDrop

The *NoCostDrop* algorithm drops messages according to all three no-cost drop criteria. Since there are many possible estimators for each of the criteria and many possible decision procedures for combining these estimators, there are many choices for implementing the *NoCostDrop* algorithm. For this work, we evaluate a relatively simple implementation based on a combination of *DLE, FIFO,* and *DropDelivered.* This implementation works as follows. First, any buffered messages that appear in a peer's delivery list are dropped immediately. Then, messages are ranked according to the value of

$$DLE.meetingValue - \gamma FIFO.timeEnqueued$$

Messages with the lowest values are dropped first. We set γ=1.0 based on an empirical study of the performance of various γ values.

Like *DLE*, the *NoCostDrop* algorithm uses selective routing, but decisions are made with respect to the value of *DLE.meetingValue -*γ*FIFO.timeEnqueued* instead of just the *DLE* meeting value.

To the best of our knowledge, *NoCostDrop* is the first routing algorithm to take advantage of all three no-cost drop criteria.

6 Evaluation Model

In this section, we describe the DTN simulations that we use to evaluate the routing performance of the algorithms described above. First, we discuss the goals of the simulations and the performance metrics used for evaluation. Then, we describe the details of our models of connectivity and communication.

The goal of these simulations is to demonstrate differences in performance among the routing algorithms we have described. Specifically, we evaluate two hypotheses:

(1) *The* NoCostDrop *algorithm, which exploits all three* no-cost drop criteria, *will have significantly better* routing performance *than* DLE *or* FIFO *alone*.

(2) *Delivery lists will improve the performance of all existing algorithms tested, since none of these algorithms explicitly models* Is-Delivered. *Furthermore, delivery lists will improve performance regardless of the underlying connectivity model.*

Before we can evaluate these hypotheses, we need a more precise definition of *routing performance*. Since the ultimate goal of these algorithms is to deliver as many messages as possible, our primary performance metric is *delivery rate*, measured as the proportion of messages generated in the network that are delivered to their destination. We also consider the *latency* of delivered messages, which is the delay from source to destination.

6.1 Modeling Meetings

The algorithms we evaluate assume no knowledge of geographic location. Therefore, we simulate peer connectivity instead of peer movements. We create a link between each pair of peers in the simulation and links are activated (representing the start of a transfer opportunity) and deactivated (representing the end of a transfer opportunity) at random. Since previous work is based on mobility models of peer movements (specifically, the random waypoint model), we model peer movement explicitly for some of our simulations and then convert the movement data into a series of link activations and deactivations that are imported into our basic simulator.

We simulate three different types of networks by using different patterns of link activation: *small-diameter networks, wide-diameter networks,* and *random waypoint networks.*

1. Small-Diameter networks: Each peer *p* has a small set of preferred peers that it meets with often and fairly regularly over the course of a simulation. Peer *p* meets with the remaining set of peers infrequently or not at all. These networks are intended to produce varied patterns of linkage over different pairs of peers that remain relatively consistent over time. The expectation is that this will give *DLE*, and other algorithms that exploit *Cannot-Deliver*, a regular pattern to learn from.

We create small-diameter networks as follows: Each pair of peers in the network has a link connecting them. For each link, we draw a *meeting count, c*, at random from an exponential distribution with mean λ. For any links with $c < \lambda$, we reset c=0. This creates fewer direct meetings between peers and reduces the number of one-hop deliveries in the network. Let *s* represent the duration of the simulation. We then calculate an *inter-meeting time, i=s/c*, for each link independently. The actual times between meetings during the simulation are drawn randomly from a Poisson distribution with a mean of *i*.

2. Wide-Diameter Networks: In comparison to small-diameter networks, this second type of network has

longer path lengths and less-varied connectivity patterns across pairs of peers. In these networks, direct deliveries are less frequent and propagation times of delivery notifications (i.e., delivery lists) are longer.

We create a long string of peers such that only a single path exists between any two peers in the network. In these networks, all links are inactive for the duration of the simulation except for those between peers p_1 and p_2, peers p_2 and p_3,…, and peers p_{N-1} and p_N. The actual times between meetings for each link are again drawn randomly from a Poisson distribution with a mean of $i=s/c$, except all links have the same parameter for c.

3. Random Waypoint Networks: We model peer connectivity from movements using the same basic random waypoint movement scenario described by Vahdat and Becker [VB00]. Specifically, our peers move in a 1500×300 meter rectangular area. Each peer begins in a random spot, chooses a destination at random, and moves there with a speed chosen uniformly from 5 to 20 meters per second. Over the course of our simulations, the average speed of a peer is approximately 10 m/s. The communication range of each peer is 250 meters. Once a peer reaches its destination, a new destination and speed is chosen at random and so on.

6.2 Modeling Communication

Each peer generates messages independently. The interarrival times of messages are drawn from a Poisson distribution, and message destinations are drawn uniformly from among the other peers in the simulation. Choosing message destinations uniformly assumes as little as possible about the applications running in our networks.

When peers meet, messages are exchanged as described in Section 3.

7 Evaluation

In this section, we describe our experimental methodology and present our results.

7.1 Methodology

For all experiments, we run 100 experimental trials on each type of network (small-diameter, wide-diameter, and random waypoint), unless otherwise noted. For each trial, we generate peers and create links between them as described in Section 6.1. We use defaults of 50 peers per network and a buffer size of 40 messages. We generate a 1,000-time-step routing scenario, consisting of message generation and link activation data, and run each of the routing algorithms against this same scenario. Each peer generates messages as described in Section 6.2, with a mean interarrival time of 10.0 time-steps. Messages have a timeout of 100 time steps after generation.

We use an average meeting count of $\lambda=10$ along each link for small-diameter networks and let $c=250$ for wide-diameter networks. For the random waypoint movement scenarios, each time step is equal to one second.

For all algorithms employing a delivery list, the proportion of the buffer devoted to the delivery list is fixed at 0.15. As discussed in Section 5.5.1, this value is not optimal across all buffer sizes and message size assumptions. However, it does perform reasonably well over a range of parameter settings. For the *Drop Least Encountered* algorithm, we use the same parameter values used by Davis et al. in their work (i.e., using their variable names: $\alpha=0.1$, $\lambda=0.95$,) [DFL01].

We evaluate the following average message size assumptions in our simulations:

- **160 Bytes**. This is the size of a GSM phone Short Message Service (SMS) text message.
- **1 KB**. This is the size of a small email. As a point of reference, an email message with no subject and no content, created with Apple's Mail application for OS X, and sent to a single recipient is larger than 1 KB. A similar message created with the Pine email application is 0.8 KB.
- **5.86 KB**. This is the size of an average email without attachments [T04].
- **50 KB**. This is the size of an average email with a small attachment [T04].
- **Infinite**. This is intended as a ceiling to demonstrate the performance limits of delivery lists. When we make the assumption of infinite (or arbitrarily large) message sizes, we assume that we can store any number of delivered message IDs in the space required to store a single message. Note that this is different than assuming that the delivery list takes up no buffer space.

7.1.1 Static vs. Adaptive Algorithms

In our evaluation, we compare two classes of routing algorithms: *static* and *adaptive*. Adaptive algorithms (*DLE*, *DropDelivered*, *NoCostDrop*) change their behavior in response to changing network conditions. Static algorithms (*Broadcast*, *Random*, and *FIFO*) behave in the same way regardless of network conditions (e.g., *FIFO* always drops the messages that have been enqueued the longest, regardless of network topology or the number of meetings between peers).

Since it may take some time before peers using an adaptive algorithm gather sufficient information to behave in an appropriate way, the performance of these algorithms generally improves over time. For this reason, re-

searches often include an initial *warm up* before beginning their evaluation. We have found that the use of a warm up period has little effect on results over the course of a 1,000 time-step simulation.

7.2 Baseline Routing Performance

All results presented here are averages over 100 trials. We assess significance of differences between algorithms using a two-tailed paired t-test.

Figures 1a–1c show the proportion of generated messages that were successfully delivered by various routing algorithms on small-diameter, wide-diameter, and random waypoint networks with varying buffer sizes. Note the differences in the scale of the y-axis between the figures.

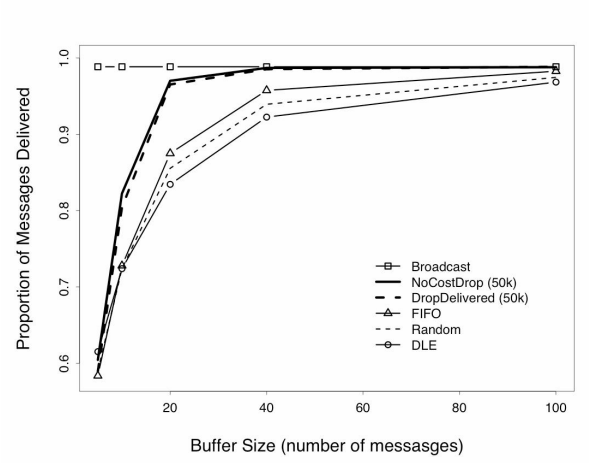

Figure 1a: Routing performance on small-diameter networks with 50 peers and varying buffer size.

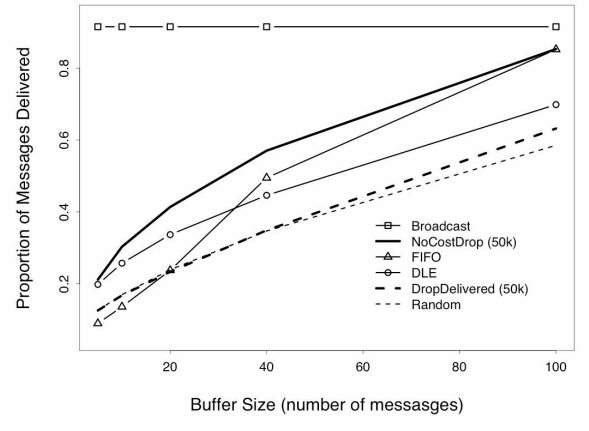

Figure 1b: Routing performance on wide-diameter networks with 50 peers and varying buffer size.

We see from these figures that, while the performance of all other algorithms (except *Broadcast*) varies across the range of network types and buffer sizes, *NoCostDrop* is consistently the top performer. We also see that, in many cases, the performance of *DropDelivered* is roughly equivalent to that of *NoCostDrop*, indicating that delivery lists can be very effective on their own. In fact, we have observed that delivery lists of sufficient size improve the performance of any of the existing algorithms we test (*FIFO*, *Random*, and *DLE*).

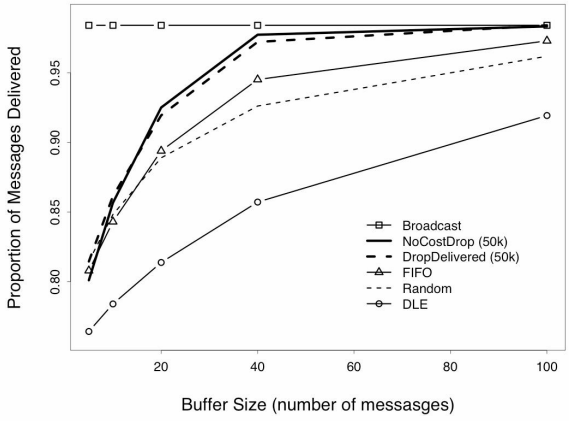

Figure 1c: Routing performance on random waypoint networks with 50 peers and varying buffer size.

However, there are cases where *DropDelivered* performs poorly. In wide-diameter networks, delivery lists are much less effective, in general, probably due to slower propagation of delivery notifications throughout the network. Note that, even here, the use of delivery lists (i.e., *DropDelivered*) does significantly improve performance over *Random* at the larger buffer sizes, though the improvement is small.

For Figure 1a, *NoCostDrop* is significantly better than all other algorithms, except at buffer sizes 5 (*DLE* is better) and 100 (*DropDelivered* is better). *DropDelivered* is significantly better than all others besides *NoCostDrop*, except at buffer size 5. For Figure 1b, *NoCostDrop* is significantly better than all others, except at buffer size 100 (*FIFO* is equivalent). *DropDelivered* is significantly better than *Random* at buffer sizes 40 and above. For Figure 1c, *NoCostDrop* is significantly better than all others besides *DropDelivered*, except at buffer size 5 (*Random* is significantly better). *DropDelivered* is significantly better than all others besides *NoCostDrop*, except at buffer size 5 (*Random* is equivalent). *DropDelivered* is significantly better than *NoCostDrop* at sizes 5, 10, and 100. The p-values for all reported differences are less than 0.005.

Figure 2 shows the performance of *NoCostDrop* on small-diameter networks under various message size assumptions. The results for the other network types are qualitatively similar. *Random* and *Broadcast* are in-

cluded for reference. Recall that as message size increases, we can store an increasing number of delivery notifications in the buffer space given up by a single message. We see a substantial increase in performance between messages ≤1K in size and messages ≥ 6K in size. This indicates that the utility of delivery lists will

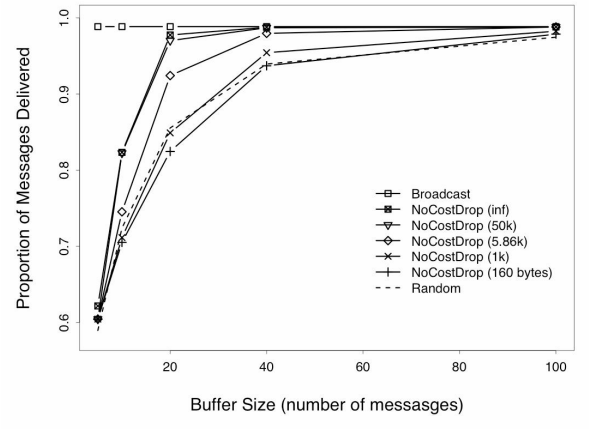

Figure 2: Routing performance of *NoCostDrop* on small-diameter networks with 50 peers, varying buffer and message size assumptions.

be application dependent. Specifically, for applications requiring very small messages (i.e., applications with a small ratio of message size to message ID size), buffer space may be better used for storing messages themselves, rather than delivery notifications.

For Figure 2, all differences are significant, except at buffer sizes 5 (*infinite* is better than the rest and all finite delivery list algorithms are equivalent and better than *Random*) and 100 (*5.86K*, *50K*, and *infinite* are equivalent and better than all others). The p-values for all reported differences are less than 0.005. Figure 3 shows the mean latency of successfully delivered messages. We also measured the mean number of hops to delivery (average delivery path length), which yielded qualitatively similar results to those in Figure 1. That is, it appears that algorithms with higher delivery rates generally have higher average hop counts.

We see from Figure 3 that the latency for algorithms employing delivery lists, *NoCostDrop* and *DropDelivered*, are notably low given the significantly higher proportion of messages delivered by these algorithms. In fact, on small-diameter and random waypoint networks, *NoCostDrop* and *DropDelivered* have the lowest latency overall. On wide-diameter networks, where delivery lists propagate much more slowly, the latencies for *NoCostDrop* and *DropDelivered* are higher relative to the other algorithms. *DLE*'s low latency on wide-diameter networks is likely due to its ability to perfectly segment the traffic in this extremely simple topology (i.e., if the destination is to the left, the peer to our right will always have a meeting value lower than the peer to our left).

For Figure 3a, *NoCostDrop* is significantly better than all others, except at buffer sizes 5 (*FIFO* is better) and 100 (*DropDelivered* is better). *DropDelivered* is significantly better than all others, except at buffer sizes 5 and 10. For Figure 3b, all differences are significant, except *Random* and *DropDelivered* are equivalent at buffer size 5. For Figure 3c, *NoCostDrop* is significantly better than all others, except at buffer sizes 5 (*FIFO* is better) and 100 (*DropDelivered* is better). *DropDelivered* is significantly better than all others at buffer sizes 40 and above. The p-values for all reported differences are less than 0.005.

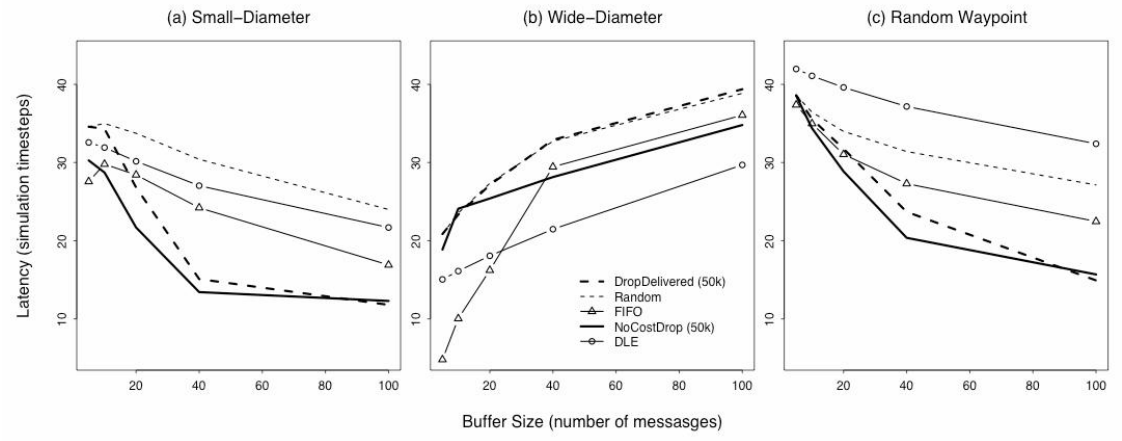

Figure 3: Average latency of delivered messages on (a) small-diameter, (b) wide-diameter, and (c) random waypoint networks with 50 peers and varying buffer size.

7.3 Varying the Number of Peers

Figures 4a-4c show the proportion of generated messages that were successfully delivered by various routing algorithms on small-diameter, wide-diameter, and random waypoint networks with a varying number of peers. For these experiments, the buffer size was fixed at 40 messages. Note the differences in the scale of the y-axis between the figures.

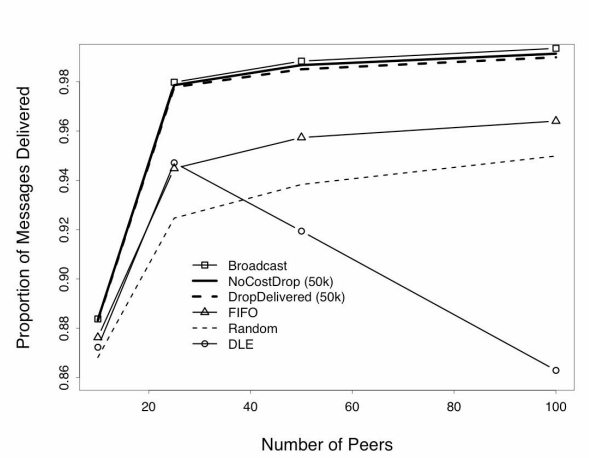

Figure 4a: Routing performance on small-diameter networks with buffer size 40 and a varying number of peers.

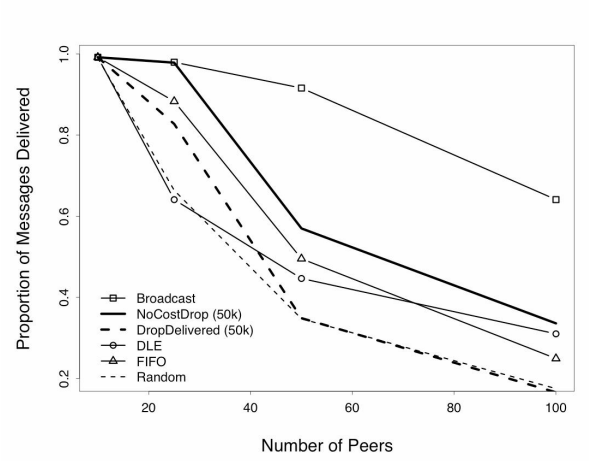

Figure 4b: Routing performance on wide-diameter networks with buffer size 40 and a varying number of peers.

For Figures 4a-4c, the results are averages over 100 trials. For Figure 4a and 4c, *NoCostDrop* is significantly better than all other algorithms, except with 10 peers (*DropDelivered* is better). *DropDelivered* is significantly better than all others besides *NoCostDrop*

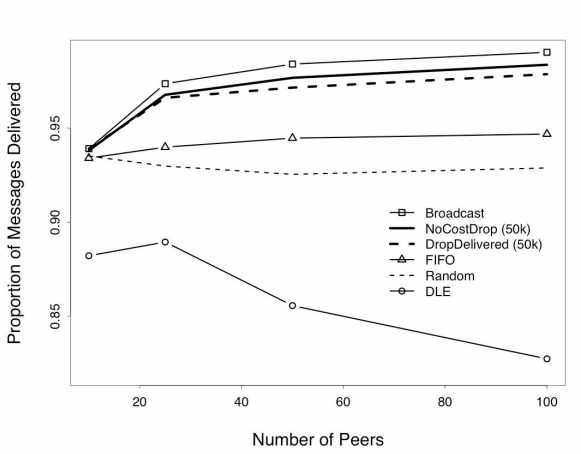

Figure 4c: Routing performance on random waypoint networks with buffer size 40 and a varying number of peers.

for all numbers of peers. For Figure 4b, *NoCostDrop* is significantly better than all others, except with 10 peers (all algorithms besides *Random* are equivalent and are significantly better than *Random*). *DropDelivered* is significantly better than *Random*, except with 100 peers (*Random* is better) The p-values for all reported differences are less than 0.005.

We see from figures 4a-4c that the results of these experiments are qualitatively very similar to those shown in Figures 1a-1c. Again *NoCostDrop* is the top performer across the range of conditions and *DropDelivered* performs almost as well, except on wide-diameter networks. However, there is one striking difference. As the number of peers increases, the performance of *DLE* drops sharply on both small-diameter and random waypoint networks. This performance dip is a direct result of *DLE* focusing exclusively on path existence and ignoring congestion.

In both small-diameter and random waypoint networks, central *hub* peers arise naturally due to the non-uniformity of meetings between pairs of peers. As the number of peers in the network increases, some hubs become so central to the network that they become overwhelmed with traffic and begin dropping messages at a high rate. Due to the large number of other peers these hubs meet with, *DLE* favors passing messages to these hubs. As congestion increases, messages are dropped with increasing frequency.

Figure 5 shows the correlation between the number of messages received and the proportion of those messages dropped for each peer in small-diameter networks of 25 and 100 peers and in wide-diameter networks of 100 peers. All peers have a buffer size of 40 messages. The

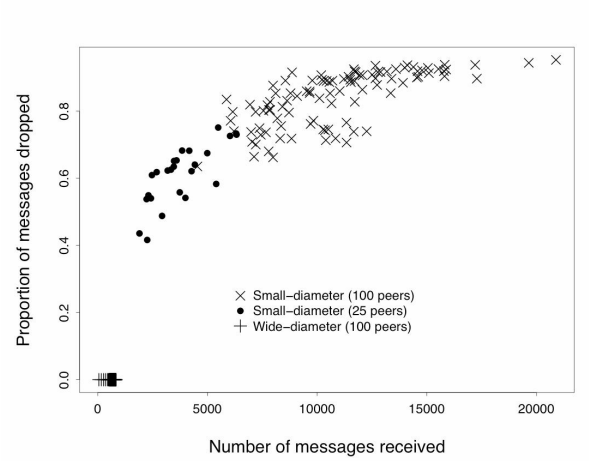

Figure 5: Number of messages received and proportion of messages dropped by peers using *DLE*.

results shown are for a single experimental trial, but are representative of our observations over all 100 trials. Note that the wide-diameter data is bunched in the lower left-hand corner of the plot.

We see several things from Figure 5. First, we see that congestion is a problem in small-diameter networks. Peers in small-diameter networks create a much larger number of message copies than peers in wide-diameter networks. In addition, there is a much larger proportion of received messages dropped in small-diameter networks. Small-diameter networks also have larger differences in the number of messages received across peers, which indicates the presence of hubs. The peers that are more hub-like (receive a larger number of messages) drop a higher proportion of the messages they receive. This indicates that congestion is more of a problem for hubs than for less loaded peers. Finally, we see that these problems are exacerbated as the number of peers increases. That is, as the number of peers in the network increases, hubs receive a larger number of messages and, correspondingly, drop a higher proportion of the messages they receive. The results for random waypoint networks have been omitted for readability due to their similarity to small-diameter network results.

It turns out that the plot shown in Figure 5 looks qualitatively similar if we use *FIFO* to route messages instead of *DLE*. This is what we expect since hubs result from the network topology, not from a particular routing algorithm. So, why doesn't *FIFO* exhibit the drop in performance we see with *DLE* on small-diameter networks of 100 peers (Figure 4a)?

As we have seen, the *DLE* algorithm is designed to predict path existence and it does so quite well. With infinite buffers, it makes a lot of sense to route messages through hubs. However, when hubs become congested, *DLE*'s performance suffers because it continues to route messages into the congested hubs. Recall that a peer using *DLE* only forwards a message *m* to peers with a higher delivery value for *m*. On the other hand, a peer using *FIFO* forwards all of its messages to every peer it meets. This allows more messages to be routed around hubs, when possible, and also gives messages more chances to be routed through hubs by different peers at different times. This effect is demonstrated in Figure 6.

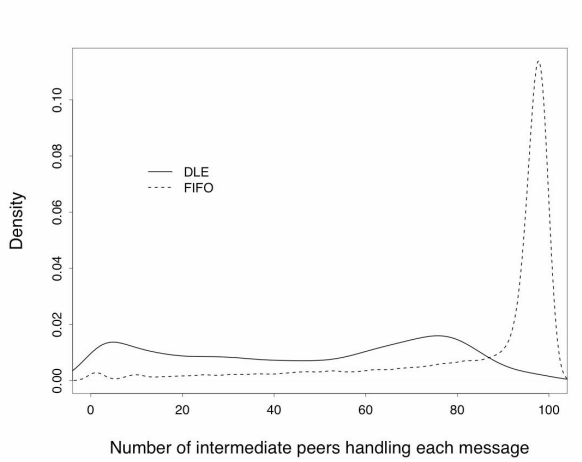

Figure 6: Distribution of the number of intermediate peers receiving each message in a small-diameter network of 100 peers.

Figure 6 shows the distribution of the number of intermediate peers receiving each message in a small-diameter network of 100 peers using the *DLE* and *FIFO* algorithms. All peers have a buffer size of 40 messages. The results shown are for a single experimental trial, but are representative of our observations over all 100 trials. An intermediate peer for a message *m* is any peer that is neither the source nor the destination of *m*.

We see from Figure 6 that when peers use *DLE*, the number of peers that receive a given message varies almost uniformly over the possible range. Using *FIFO*, on the other hand, a significant proportion of messages make their way to almost every peer in the network. Although this may not be very efficient, it appears to be effective, at least in cases where central peers are congested.

7.4 Discussion

As previously discussed, *DLE* exploits *Cannot-Deliver* and *FIFO* exploits *Delivered-if-Dropped* and *Is-Delivered*. The *DropDelivered* algorithm also exploits *Is-Delivered*. *NoCostDrop* utilizes all three no-cost drop criteria.

NoCostDrop dominates both *DLE* and *FIFO* in terms of delivery rate for all network types and conditions tested.

NoCostDrop also has lower latency overall in both small-diameter and wide-diameter networks. Furthermore, *NoCostDrop* is robust to a variety of conditions, including ones that cause its constituents to perform poorly (e.g., *DLE* with a large number of peers).

The use of delivery lists to estimate *Is-Delivered* (i.e., *DropDelivered*) also appears to dramatically improve performance across a range of conditions. Given large enough message sizes, algorithms that use delivery lists deliver a dramatically higher proportion of messages and have lower average latency than algorithms that do not. The exception to this is wide-diameter networks, where the very simple topology limits the propagation of delivery notifications.

A major reason for the performance boost we get from delivery lists is their accuracy in estimating the likelihood of the *Is-Delivered* criterion. Assuming sufficiently fast propagation of delivery notifications and sufficient storage for these notifications, our estimates of *Is-Delivered* are nearly perfect. On the other hand, it is not apparent how to obtain perfect estimates of the *Cannot-Deliver* or *Delivered-if-Dropped* criteria even with perfect information available. This issue is discussed further in the next section. We know that any message in a delivery list is absolutely safe to drop. For the other algorithms, we never know the values of the drop criteria with certainty. Since we cannot determine each of the criteria with the same degree of certainty, it is difficult to make conclusions about the relative importance of the three criteria, independent of the techniques used to estimate them.

It is worth noting that delivery lists may be less effective when fewer messages are delivered, since they do not provide as much information in these cases. However, even in such cases, delivery lists should not degrade performance, assuming that resources are allocated appropriately between the message buffer and delivery list.

As noted above, delivery lists also seem to improve average latency. Here, we offer an initial explanation, but this deserves further study. By dropping delivered messages as soon as possible, we create more space in message buffers for undelivered messages. This causes fewer undelivered messages to be dropped on the way to their destination, thus decreasing average latency. In addition, delivery lists provide a type of positive feedback mechanism. The more messages are delivered, the more can be dropped, which further improves the delivery rate and average latency.

7.5 Routing under Relaxed Assumptions

It is likely that *NoCostDrop* may be improved by using more sophisticated estimation techniques or more relaxed assumptions about the availability of information. For some applications, it may be practical to carry out some amount of communication out-of-band. For others, peers may have knowledge of their geographic location through the use of technologies such as GPS. In other cases, peers may have known connectivity patterns or we may be able to learn a model of connectivity patterns offline. Finally, for some applications, we may be able to control the movement of peers for the purposes of improving routing. We have started an initial exploration of ways to improve estimates of the no-cost drop criteria by utilizing these additional resources and sources of information.

We have tried techniques such as running a simulation once, estimating *Cannot-Deliver* values based on the proportion of successful deliveries between pairs of peers, and then running the simulation again, routing using the learned *Cannot-Deliver* values. So far, this has yielded only minor performance increases relative to the performance gains achieved through the use of delivery lists.

On the other hand, it is not difficult to achieve perfect estimates of *Is-Delivered* through the use of out-of-band communication. If we broadcast delivery notifications to all peers in the system out-of-band, any peers buffering a delivered message can drop the message immediately without the need for maintaining delivery lists. Based on our findings, this would lead to a huge performance increase in situations where out-of-band communication is possible, independent of assumptions about message size.

8 Conclusions and Future Work

DLE and *FIFO* are representative of the existing routing algorithms for *DTN*s. We explain the performance of these algorithms in terms of their ability to exploit a set of three *no-cost drop criteria* (*Cannot-Deliver*, *Is-Delivered*, and *Delivered-if-Dropped*). We have shown that *DLE* exploits *Cannot-Deliver*, although incompletely, and *FIFO* exploits *Delivered-if-Dropped* and *Is-Delivered*, although indirectly. In this paper, we introduce the *DropDelivered* algorithm, which also exploits *Is-Delivered*.

By utilizing all three no-cost drop criteria, the *NoCostDrop* algorithm outperforms *DLE*, *FIFO*, and *DropDelivered* in terms of delivery rate and, in many cases, latency as well. We also find that the performance of the extremely simple *DropDelivered* algorithm is comparable to *NoCostDrop* in many cases, and that delivery lists perform quite well in general. Part of the power of delivery lists lies in the fact that they can easily be combined with many existing algorithms.

This work represents a first attempt at explaining routing performance in disruption tolerant networks. Our findings suggest a number of interesting research questions and directions, including:

- How can we more accurately estimate the no-cost drop criteria? For example: Can we leverage the notifications in delivery lists to better estimate *Cannot-Deliver*? How much better can we do with relaxed assumptions about the availability of information in the network (e.g., message deliveries, connectivity patterns)? Can we modify *DLE* and similar algorithms to account for congestion implicitly (e.g., by dropping messages stochastically) or explicitly?
- How can we combine estimates of the no-cost drop criteria more effectively to create better routing algorithms?
- All of the messages that do not meet the no-cost drop criteria for peer *p* are equal in the sense that they are all deliverable and all depend on peer *p* to deliver them. However, these messages may not be equal in terms of the resources required to deliver them (e.g., the number of peers a message must pass through on the way to its destination). This suggests a fourth criterion that could be used to further improve performance.
- Are there optimizations that can improve the performance of delivery lists? For example, can we adaptively set the proportion of buffer space devoted to undelivered messages and the delivery list? Can we reduce the space requirements of delivery lists, for instance, by using Bloom filters [B70]?

9 Acknowledgments

Thanks to members of the UMass Knowledge Discovery Laboratory for helpful comments on this work at various stages. Special thanks to Cindy Loiselle for her help revising the paper.

10 References

[B70] B. Bloom. Space/Time Trade-offs in Hash Coding with Allowable Errors. *Communications of the ACM*, v.13 n.7, pp.422–426, July 1970.

[BBL05] B. Burns, O. Brock, and B. Levine. MV Routing and Capacity Building in Disruption Tolerant Networks. In *Proc. IEEE Infocom*, March 2005.

[DFL01] J. Davis, A. Fagg, and B. Levine. Wearable Computers And Packet Transport Mechanisms In Highly Partitioned Ad-Hoc Networks. In *Proc. Intl. Symposium on Wearable Computers*, October 2001.

[GV03] M. Grossglauser and M. Vetterli. Locating Peers With Ease: Mobility Diffusion Of Last Encounters In Ad hoc Networks. In *Proc. IEEE Infocom*, April 2003.

[JFP04] S. Jain, K. Fall, and R. Patra. Routing in a Delay Tolerant Network. In *Proc. ACM SIGCOMM*, Aug 2004.

[JM96] D. B. Johnson and D. A. Maltz. Dynamic Source Routing In Ad Hoc Wireless Networks. In *Mobile Computing*, volume 353 of *The Kluwer International Series in Engineering and Computer Science*. Kluwer Academic Publishers, 1996.

[LR00] Q. Li and D. Rus. Sending Messages to Mobile Users in Disconnected Ad hoc Wireless Networks. In *Proc. MobiCom*, August 2000.

[LDS04] A. Lindgren, A. Doria, and O. Schelén. Probabilistic Routing in Intermittently Connected Networks. In *Proc. Workshop on Service Assurance with Partial and Intermittent Resources*, August 2004.

[PR99] C. E. Perkins and E. M. Royer. Ad Hoc On-Demand Distance Vector Routing. In *Proc. IEEE Workshop on Mobile Computing Systems and Applications*, pp.90–100, February 1999.

[SG04] N. Sarafijanovic-Djukic and M. Grossglauser. Last Encounter Routing Under Random Waypoint Mobility. In *Proc. IFIP-TC6 Networking Conference*, May 2004.

[T04] T-Mobile Support Knowledge Base. http://support.t-mobile.com/knowbase/root/public/tm21399.htm.

[VB00] A. Vahdat and D. Becker. Epidemic Routing for Partially-Connected Ad hoc Networks. *Technical Report CS-2000-06, University of California San Diego*, July 2000.

[ZA03] W. Zhao and M. Ammar. Message Ferrying: Proactive Routing In Highly-Partitioned Wireless Ad Hoc Networks. In *Proc. IEEE Workshop on Future Trends in Distributed Computing Systems*, May 2003.

[ZAZ04] W. Zhao, M. Ammar, and E. Zegura. A Message Ferrying Approach for Data Delivery in Sparse Mobile Ad hoc Networks. In *Proc. ACM Mobihoc*, May 2004.