Practical Routing in Delay-Tolerant Networks

Evan P.C. Jones, Lily Li, Jakub K. Schmidtke, and Paul A.S. Ward, *Member*, *IEEE*

Abstract—Delay-tolerant networks (DTNs) have the potential to interconnect devices in regions that current networking technology cannot reach. To realize the DTN vision, routes must be found over multiple unreliable, intermittently-connected hops. In this paper we present a practical routing protocol that uses only observed information about the network. We designed a metric that estimates the average waiting time for each potential next hop. This learned topology information is distributed using a link-state routing protocol, where the link-state packets are "flooded" using epidemic routing. The routing is recomputed each time connections are established, allowing messages to take advantage of unpredictable contacts. A message is forwarded if the topology suggests that the connected node is "closer" to the destination than the current node. We demonstrate through simulation that our protocol provides performance similar to that of schemes that have global knowledge of the network topology, yet without requiring that knowledge. Further, it requires significantly less resources than the alternative, epidemic routing, suggesting that our approach scales better with the number of messages in the network. This performance is achieved with minimal protocol overhead for networks of approximately 100 nodes.

Index Terms—Routing protocols, mobile communication systems, nomadic computing.

1 Introduction

Delay-tolerant networks (DTNs) have the potential to connect devices and areas of the world that are underserved by traditional networks. DTNs enable communication by taking advantage of temporary connections to relay data in a fashion similar to the postal network [1], instead of requiring an end-to-end network path to be available. These networks are being investigated for education [2], telecommunications [3], government services [4], environmental monitoring [5], vehicular communication [6], and deep space [7].

One obstacle that currently limits deployment of these networks is that it is difficult to determine how to get data from the source to the destination. Simple DTN-like networks have been built using static routing [2], [3], which is an effective approach for small networks. However, the benefit will increase if the networks can be scaled to service larger areas. To achieve this goal, routing protocols are needed to automate the configuration and to cope with changes and failures.

This paper presents a routing protocol designed to be easy to deploy. This is an extended version of the original workshop paper where this protocol was first presented [8]. In order to make the protocol easy to deploy, it must meet three design goals. First, the routing must be self-configuring. This reduces the cost to deploy and maintain a network, as fewer people will be required. Additionally, it is critical for equipment that may be deployed far from network experts and to maintain connectivity even when some components fail. Many application domains where DTNs can provide significant benefits have both of those problems. Second, the protocol must provide sufficient performance over a wide variety of connectivity patterns for most DTN scenarios. A self-configuring routing protocol that has very poor delivery ratios or extremely large latency may well be of limited value. Finally, the protocol must make efficient use of buffer and network resources. If the DTN becomes a valuable resource, it will be used frequently by a large number of users. Thus, it must be capable of scaling with demand.

Before discussing the details of our protocol, we first describe our model of DTNs, a simplified version of the model presented by Jain et al. [9]. A DTN is composed of computing systems, called nodes, that participate in the network. Bidirectional links with constant bandwidth and latency connect some nodes together. These links may go up and down over time due to mobility, failures, or other events. When the link is up, the nodes have an opportunity to send data to each other. This opportunity is called a contact [1]. Only a single contact may exist between two nodes at one time. The contact schedule is the set of times when the contact will be available. In graph theory, this model is a time-varying graph.

This model differs from the more general model presented by Jain et al. in three major ways. First, we assume that the contacts are always bidirectional. Our protocol requires a two-way exchange of data between nodes and, thus, it will not operate in networks with unidirectional links. The second is that contacts have constant bandwidth and latency. Our protocol does not adapt based on bandwidth and latency, as this would only complicate our evaluation. Finally, we only permit a single contact to exist between two nodes, which means that our protocol cannot take advantage of multiple network interfaces. This is not a fundamental limitation. Our protocol can handle this by treating the combination of all the contacts between a pair of nodes as a single virtual contact. We do not evaluate this here.

- *E.P.C. Jones, L. Li, and P.A.S. Ward are with the Department of Electrical and Computer Engineering, University of Waterloo, Waterloo, Ontario, N2L 3G1, Canada. E-mail: {ejones, l7li, pasward}@uwaterloo.ca.*
- *J.K. Schmidtke is with the Faculty of Electronics and Information Technology, Warsaw University of Technology, Warsaw, Poland. E-mail: jschmidt@elka.pw.edu.pl.*

Manuscript received 20 Apr. 2006; revised 28 Sept. 2006; accepted 23 Oct. 2006; published online 7 Feb. 2007.
For information on obtaining reprints of this article, please send e-mail to: tmc@computer.org, and reference IEEECS Log Number TMC-0112-0406.
Digital Object Identifier no. 10.1109/TMC.2007.1016.

The remainder of this paper is organized as follows: First, we discuss the previous work on the delay-tolerant routing problem, demonstrating its limitations. In Section 3, we describe the design of our protocol and justify our design decisions. We investigate the performance of specific aspects of our protocol in Section 5 and then present the performance for some realistic scenarios in Section 6. Finally, we discuss our conclusions and future work.

2 Related Work

Because the routing problem must be solved in order to use a new network, there has been extensive research on routing in delay-tolerant networks. The work dates back to before the term "delay-tolerant" was widely used. The adjectives "intermittently-connected," "sparse," and "disconnected" are also used to describe networks without constant end-to-end connections. The previous work can be divided into two broad categories: Flooding protocols and forwarding protocols.

2.1 Flooding Protocols

Flooding protocols distribute many copies of the message to a large number of nodes with the hope that one of these intermediate nodes will reach the destination. These protocols typically operate without any information about the network. The representative flooding protocol is epidemic routing [10]. It attempts to give all nodes a copy of every message through random exchanges between nodes. This approach can achieve high delivery ratios. If it is provided infinite bandwidth and buffer resources, it will deliver all the messages that can possibly be delivered in the minimum amount of time. It requires no knowledge about the network and, thus, it satisfies the first two of our three design criteria. Unfortunately, it is very expensive in terms of the number of transmissions and buffer space. Thus, it does not satisfy our requirement for an efficient protocol, as it does not appear that this approach can scale as the number of messages and nodes in the network increases.

Many papers have studied ways to make epidemic routing more efficient [11], [12], [13], [14], [15], [16]. Critical resources for epidemic routing are the buffer and bandwidth. An intelligent buffer management scheme can improve the delivery ratio over the simple FIFO scheme [11]. An effective buffer policy is to drop packets that are the least likely to be delivered based on previous history. If node A has met B frequently, and B has met C frequently, then A is likely to deliver messages to C through B. Similar metrics are used in a number of epidemic protocol variants [11], [12], [13], [15], [16]. This approach takes advantage of physical locality and the fact that movement is not completely random. While these protocols are more efficient than the original epidemic routing protocol, they still transmit many copies of each message.

An extension is to use "death certificates" [17] to remove delivered messages from the network. Small and Haas show that, the more aggressively the death certificates are propagated, the less storage is required at each node [14], while Harras and Almeroth show that the more aggressive strategies transmit more messages [18].

An alternative to epidemic routing is to spread copies of a message to a limited number of nodes. A variety of schemes are possible, such as limiting the message distribution to a tree [14], [19] or limiting the number of hops the message can travel to some small number [10], [20], [21], [22], [23]. All of these schemes reduce resource consumption, but also reduce the delivery ratio. Additionally, in realistic scenarios where nodes are not simply moving randomly in a space, these limits might mean that it is impossible for two nodes in the network to communicate, simply because they are far apart.

2.2 Forwarding Protocols

An alternative approach to flooding is to forward a single copy of each message along a carefully selected path. There are many possible ways to select a path. One approach is to use location-based routing, where nodes pass the message on to the next hop that is closest to the destination. The measure for "closeness" could be physical distance. Lebrun et al. proposed using the motion vector of mobile nodes to predict their future location. Their scheme passes messages to nodes that are moving closer to the destination [24]. Leguay et al. presented a strategy that uses a virtual coordinate system [25], so the measure of "closeness" represents the probability that the nodes will come into contact with each other. These studies show that location-based routing is feasible for delay-tolerant networking. However, there are two well-known problems that these studies do not address. The first is that, even if two nodes are "close," there is no guarantee that they will ever be able to communicate. Thus, messages can get trapped in local minima and not be delivered. The second problem is that the source needs to know the location of the destination in order to send it a message. This is easy if the location of nodes never changes, but it is likely that will not be the case in a delay-tolerant network. These problems are well-studied in the context of ad hoc wireless networks [26], but no DTN-specific protocols have been proposed that solve them.

Jain et al. showed that it is possible to apply traditional shortest-path routing techniques to DTNs [9]. They use the future contact schedule to compute a metric for each contact, then use source routing to forward the message over the shortest path. Their results show that the efficiency and performance increases with the amount of information used for the metric. Unfortunately, this scheme cannot be self-configuring because it requires the complete contact schedule. Handorean et al. explore alternatives for distributing connectivity information, but they still assume that each node knows its own connectivity perfectly [27].

Jain et al. also introduced the "First Contact" routing scheme, where the message is forwarded on a randomly chosen contact or on the first available contact if none of them are connected at the time of message arrival. This simple technique requires no information about the network and, thus, no configuration. However, it "performs poorly in nontrivial topologies" [9] because it makes random decisions. In particular, it never attempts to learn the topology, having the same routing performance for all time. By contrast, our approach, while it too starts with zero knowledge, will learn the network behavior and, assuming

that behavior is not random, improve its performance over time. Thus, we do not consider this protocol further.

3 Routing Protocol Design

Our protocol is a shortest-path routing protocol for delay-tolerant networks. Its design is based on routing in traditional networks, but some design decisions were modified for this new environment. First, we discuss some of the issues in selecting a path metric and present the metric we use. Next, we investigate when to make routing decisions. Finally, we describe how to distribute topology information.

3.1 Path Metrics for DTNs

Paths must be carefully selected to extract the best performance from a network. In a DTN, the primary requirement is that messages are reliably delivered. Thus, the delivery ratio is a very important metric. Unfortunately, it is not clear how a metric can be constructed to directly maximize the delivery ratio along a path. To resolve this problem, we follow the same approach as Jain et al. and choose to minimize the end-to-end delay [9]. This reduces the amount of time a message occupies buffers in the network, which intuitively should reduce the number of messages dropped, assuming that buffer overflow is the primary cause of loss. Additionally, previous work in the distributed systems community has shown that it is not possible to implement many important algorithms, such as consensus, election, or membership, using networks that provide an asynchronous, time-free model [28]. However, it is possible to implement them using an asynchronous timed mode [29]. Thus, it seems useful to strive for timely delivery of messages.

In a delay-tolerant network, the end-to-end delay has four components. First, the message must wait for the next contact to arrive (waiting time). Next, the data queued ahead of the current message must be delivered (queuing delay). The message must then be transmitted (transmission delay) and, finally, the signal must propagate to the next hop (latency). Delay is an attractive metric because these four factors can be combined into a single number, assuming that sufficient information is available. However, to simplify the discussion in this paper, we assume that links have a very high throughput and low latency, which means that the waiting time is the only significant factor.

A variety of metrics for minimizing the end-to-end delay in a DTN have been explored by Jain et al. [9]. However, most of them require knowledge of future contact arrival times. An exception is the Minimum Expected Delay (MED). This metric assigns a cost to each edge equal to the average waiting time plus the transmission delay. Once this value is computed, the contact schedule is not needed. Assuming that message arrival times are uniformly distributed, the waiting time probability distribution is a piecewise linear function. It is a straightforward application of basic probability to compute the expected value. The derivation of the metric is included in the Appendix.

We propose a variant of MED we call the Minimum Estimated Expected Delay (MEED). Instead of computing the expected waiting time using the known contact schedule, MEED uses the observed contact history. This assumes that the future connectivity will be similar to the previously observed connectivity. The details of this metric are reviewed in Section 3.6.

3.2 Routing Decision Time

The earliest opportunity that the path for a message can be decided is when it is generated at the source, which is called source routing. This is a simple approach but it is inappropriate for our protocol because, as the message travels closer to the destination, the nodes will likely have more recent and accurate information about the destination's connectivity. Hence, it seems natural that these intermediate nodes can make better decisions than the source.

The next time to make forwarding decisions is when a message arrives at an intermediate node, which is called per-hop routing. When the message arrives, the node determines the next hop for the destination and places it in a queue for that contact. This is also not a good solution for DTNs, as changes to the topology could occur after the message arrives. This would result in the message waiting to be forwarded over a suboptimal link.

In order to make routing decisions with the best possible information, we use what we call per-contact routing. Instead of computing the next hop for a message in advance, the routing table is recomputed each time a contact arrives. After a new routing table has been computed, we examine each message in the buffer to determine if any of them need to be forwarded over any of the available contacts. This assures that each routing decision is made with the most recent information. The disadvantage is that the routing table may be recomputed more often if the contacts go up and down frequently. Additionally, the routing must be recomputed before any messages may be forwarded. Thus, there may be some additional delay before a link will be used. As long as the processing power of the nodes is appropriate for the size of the network, this delay will not be significant. However, it may be a limiting factor in scaling this approach to very large networks.

Since we are recomputing the routing table each time a contact becomes available, we can improve the performance further by temporarily assigning the contact a cost of zero in our local routing table. This value is used when computing the routing table, but is not propagated to other nodes. This "short circuits" the routing decisions made by the link-state protocol, allowing messages to take advantage of good timing. This is similar to the approach used in some epidemic routing variants [11], [15], and to what Handorean et al. call a "path update" [27]. Per-contact routing combined with short circuiting is effective for delay-tolerant networks because it guarantees that decisions are always made with the most recent information possible, and it can take advantage of serendipitous contact arrivals to make the routing more efficient.

For example, imagine we have a network with four nodes. Node A has a message for node D. There are two possible next hops: B with a total path cost of 5, and C with a total path cost of 10. This topology is shown in Fig. 1a. Thus, the current routing state says the message should

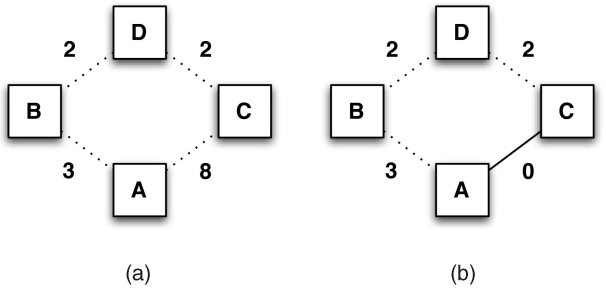

Fig. 1. Per-contact routing example.

wait for B to reach D. However, node C connects first. Thus, the cost to go from A to C becomes zero, as shown in Fig. 1b. With per-hop or source routing, the message would remain queued at A waiting for the path with cost 5 through node B. However, per-contact routing takes advantage of this unforeseen contact and delivers the message to C, where it will wait for a path with cost 2.

3.3 Impact of Deferring Routing Decisions

Making routing decisions as late as possible seems like a clear win for delay-tolerant networks because it allows the path taken by a message to change while it is in transit. Consider the situation where the next hop for a message does not become available due to a failure. At some point, source or per-hop routing would give up and drop the message because the next hop has disappeared. However, per-contact routing automatically uses an alternate contact as soon as the routing metric decides that is the best path.

Unfortunately, this design decision has one drawback. Link-state routing is loop free only if the same topology is used to make all the routing decisions along the path of a message. Source routing guarantees that this occurs because all the routing decisions are made at the source. In connected networks, per-hop routing assumes that the topology does not change while the message is in transit. This is reasonable since the end-to-end delay is measured in milliseconds and topology changes are relatively rare, but this assumption does not hold for DTNs. If the link weights change while the data is in transit, it is possible for a packet to get passed between two nodes indefinitely. For a loop to occur, the link weights must change enough so that, when the packet gets to one of the nodes, the routing directs the packet back the way it came.

A situation where this loop could occur in a DTN is shown in Fig. 2. Initially, node B has a message for node A, and the shortest path is BCA. When node C connects to B, the message is forwarded to C. Now, imagine that the cost of the contact BA decreases because A connects to B. Meanwhile, the cost of the CA contact increases to 6 because C has not been connected to A for some time. At this point, the situation is the mirror image of how it began. The message would have been delivered if it had stayed at B. Now, if B connects to C, the message will be sent back to B. If the connectivity pattern is periodic, the message will bounce between B and C indefinitely. Short circuit routing aggravates this problem because the link cost between B and C no longer matters, so the link costs need to fluctuate less.

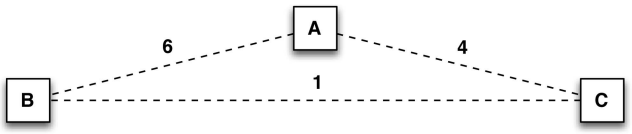

Fig. 2. Routing loop caused by per-contact routing.

This problem could be mitigated by adding hysteresis. In order to backtrack, the path must improve upon the next best path by some threshold. The threshold could increase if a node is revisited multiple times. This would require per-message state and, thus, additional resources. We do not implement a solution in our simulations because we did not observe any loops. We did observe a number of situations where messages backtrack, sometimes in less than ideal ways, but they continued to make forward progress. A real implementation may need to include a solution because, while this situation seems unlikely to occur, it would cause significant disruption if it did.

3.4 Topology Distribution

Once we have costs for individual links, the information needs to be distributed throughout the network. Traditional networks typically use link-state or distance vector algorithms for this purpose, although other choices are possible. We chose to implement a link-state routing algorithm for two reasons. The primary reason is that there is a natural match between flooding in link-state algorithms and epidemic message distribution. Flooding distributes a copy of each link-state table to all nodes. Epidemic replication distributes a copy of each message to all nodes. Thus, we can implement a link-state protocol in a DTN using an epidemic algorithm, which is known to be very robust.

The secondary reason we chose to implement link-state routing is that it provides the complete topology at each node, which allows the topology to be updated in a single contact. If a node has been disconnected for a long time, it can obtain the entire topology in a single exchange with any other node. A distance vector algorithm would only distribute paths that pass through the other node, and multiple exchanges would be required to obtain the entire topology.

Link-state routing does have its disadvantages. First, each node must store the entire topology in its routing tables, which could be larger than the state required for distance vector routing. Second, merging topology information from multiple nodes becomes more complicated because there is more information to be synchronized.

3.5 An Epidemic Link-State Protocol

Upon connection, nodes exchange summary vectors that list the link-state tables the nodes have received. Each table is tagged with a sequence number which permits the nodes to determine which ones are the most recent. The nodes exchange any missing updates so that they both have the same topology state. Then, they recompute their routing tables and finally can forward messages to the other node. Since we use per-contact routing, the metric for each link is temporarily set to zero to represent the fact that messages

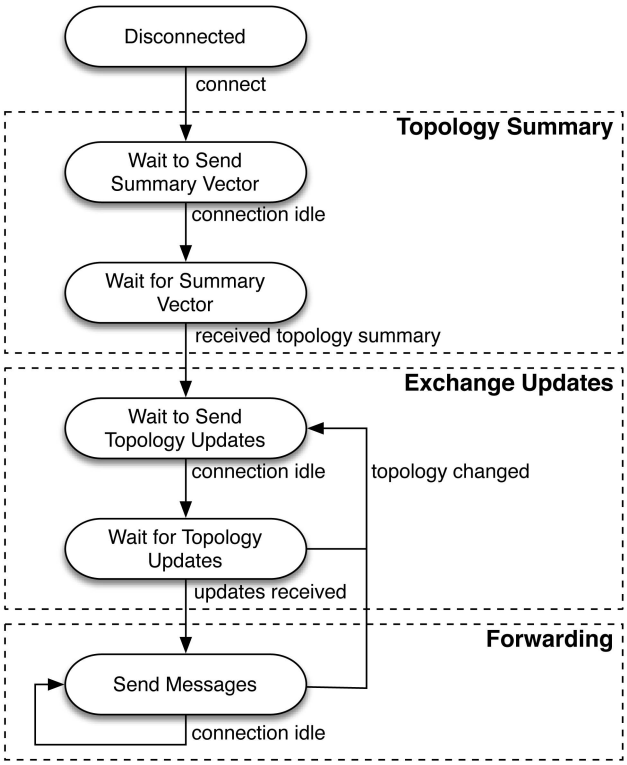

Fig. 3. The epidemic link-state protocol state machine.

can be immediately sent to the other node. A protocol state machine, shown in Fig. 3, is maintained for each connection.

When the local link state table changes, an update must be propagated to all nodes in the network. This is an expensive operation. To reduce the overhead, a node may suppress updates that it decides are unnecessary. However, it is essential that it continues to make routing decisions using the table that it last advertised; otherwise, the routing tables could be inconsistent between connected nodes, which might cause an immediate routing loop. Our simple implementation propagates an update if at least one weight changed by more than 5 percent or if a new contact has been added.

To estimate the overhead of this protocol, the size of each protocol message must be determined. There are two protocol messages exchanged: summary vectors and topology updates. The summary vector contains one (source id and sequence number) pair for each node in the network. If we assume each value has a fixed length, the size of this message scales linearly with the size of the network. The topology update contains a set of (source id, sequence number, link partner id, and metric) tuples. In the worst case, the complete topology must be transmitted. The complete topology has one tuple for each node's contacts. If there are N nodes in the network and each node has D contacts, then a total of ND tuples are required to specify the full topology. If we assume that the average node degree is bounded by a constant, this overhead also scales linearly with the size of the network. As a somewhat realistic example, if we encode these messages as arrays and assume each value is a 32-bit integer, an 802.11 packet containing 1,500 bytes of data can store an update with information about 92 links, which would be sufficient for a network with 15 nodes with an average degree of 6. Thus, the overhead amounts to a single 802.11 transmission for small networks. A 100 node network, with average degree 10, will require about 11 such overhead packets, which at the maximum transmission rate for 802.11 amounts to about 5 ms, which, in many scenarios, is negligible. Even a 300 node network with average degree 50 will require less than a second of transmission time at the 802.11 base rate. Overhead may be a problem for very large networks.

3.5.1 Example

Consider three nodes, A, B, and C, that have begun the routing algorithm, but have no knowledge about each other. Initially, their link state tables are empty. If A and B encounter each other first, they will begin the algorithm by exchanging empty summary vectors, representing the fact that they have no network knowledge. Examining these vectors, they will see that they are each completely up to date, as there is no network topology yet. Additionally, A and B will record that they have seen each other in order to begin learning the network topology. At this point, the contact is up and they can exchange messages with each other. After this contact goes down, A and B have a nonzero link cost for the AB contact. If the BC contact becomes available next, B and C will exchange summary vectors. B will see that C knows nothing about the AB contact, so B will send C its current link estimate. After this contact goes down, the AB and BC contacts will each have nonzero values.

3.6 The MEED Metric

Our protocol relies on an estimate of the connectivity in order to make intelligent routing decisions. To do this, we use the expected-delay metric originally presented by Jain et al. [9]. This metric computes the expected delay for a message to go from one node to another using a given contact, assuming that all message arrival times are equally likely. The derivation of this metric is simple and is included in the Appendix. The final computed metric value is shown in (1), where n is the total number of disconnected periods, d_i is the duration of a given disconnected period, and t is the total time interval over which these disconnections were observed:

$$\frac{\sum_{i=1}^{n} d_i^2}{2t}. \tag{1}$$

The original metric is computed using the contact schedule for the entire period that the network is in use. However, it is possible to compute this metric for any arbitrary time period. If we assume that the future behavior of a contact will be similar to the past, we can use the value for the past as the current estimate. We present three techniques for computing this metric: the infinite history window, the sliding history window, and the exponentially-weighted moving average.

3.6.1 Infinite History Window

The simplest approach is to compute the metric over the entire history of a node. This is easy to do since the

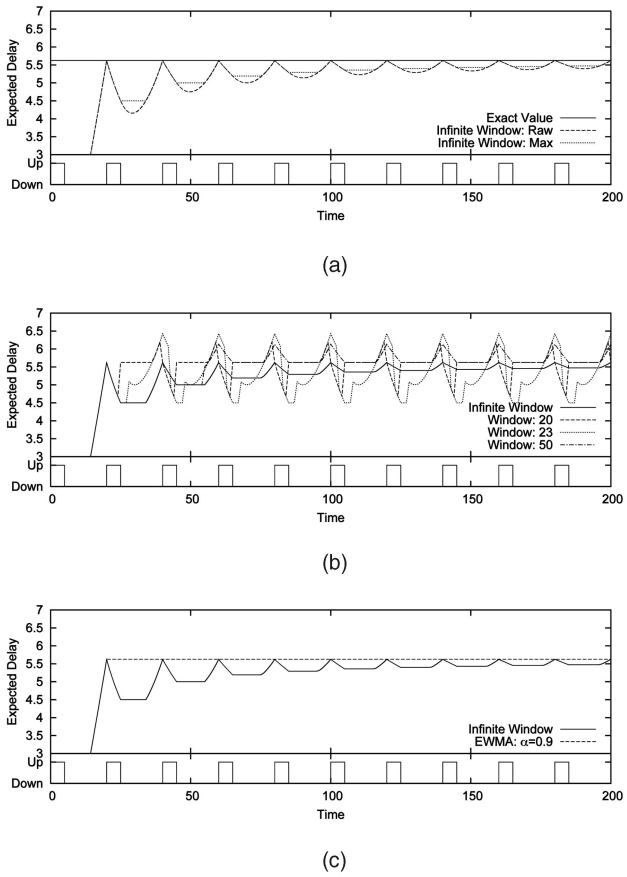

Fig. 4. Comparison of MEED metric variants. (a) Infinite window expected delay. (b) Sliding window expected delay. (c) Exponentially weighted moving average expected delay.

computation only requires a sum of squared disconnection times, the current time, and the initial time where the observations began. Unfortunately, this scheme is somewhat impractical. First, the sum of disconnection times continues to increase without bound. This means that it may be difficult to compute this value if a node is running for a long time. Second, if the connectivity pattern for a contact changes for some reason, the metric will average both the old behavior and the new behavior. This means that this approach will take a long time to adjust to changes.

There is also a small problem with this simple approach. If the contact is currently down and we include the interval between now and the last time the contact was available, the metric value can actually decrease. Consider the case where a contact has a periodic pattern and is up for 5 time units, then down for 15. This pattern has an expected delay of $15^2/(2 \times 20) = 5.625$. In Fig. 4a, we plot this contact's state in the bottom part of the graph and the metric in the top part of the graph. The raw metric line in Fig. 4a increases when the contact is down and decreases when it is up. However, immediately after the contact goes down, it continues to decrease. This occurs because we do not know when the contact will become available next, so including the current disconnected interval makes a best-case estimate, assuming that the contact will come up in the next instant. To solve this, we take the maximum of this best-case estimate and the metric when the contact was last available, as shown by the "max" line in Fig. 4a. This way, the metric will only decrease when the contact is up and never when the contact is down. For a periodic contact, the infinite window computation equals the ideal value just before the contact comes up. However, the metric approaches the correct value as time approaches infinity.

3.6.2 Sliding History Window

In order to make the metric more responsive to changes, we can compute it over a sliding window. This means that, if the contact-connectivity behavior changes, the old behavior will eventually be removed from the history window and no longer included in the metric. Unfortunately, this also works in reverse. If the history window does not capture a large enough sample of the contact behavior, it may fluctuate significantly and not accurately represent the average behavior. In order to avoid some undesirable behavior that happens when averaging over partial up or down periods, we round our history window up to the last connect or disconnect event. Because of this, the metric will oscillate around the true average, even if the window is exactly the same size as the period of the contact.

We again show a contact with an up period of five time units and a down period of 15 time units in Fig. 4b. Even when the sliding window size is a multiple of the period (20), it oscillates around the exact metric. However, windows that are much larger than the period, such as the window of size 50 in the figure, oscillate less.

This version is more complex to compute than the infinite history window approach. For this metric, we must record all the connection and disconnection times within the window. In theory, this means that the potential amount of data that must be stored is infinite, as the contact could go up and down an infinite number of times within a given time window. Practical systems will have some logical minimum connection period that is dictated by the link technology. However, the number of disconnected periods that must be stored could potentially be very large.

3.6.3 Exponentially Weighted Moving Average (EWMA)

It is not initially obvious how to use an exponentially weighted moving average to compute a continuous metric like the expected delay. However, it is possible to compute it if we pretend that contact behavior is perfectly periodic and use an EWMA to estimate the average connection and disconnection lengths. Thus, each time a contact goes up or down, we update the estimate as follows:

$$D = \alpha D + (1-\alpha)d_i,$$
$$C = \beta C + (1-\beta)c_i,$$
$$E = \frac{D^2}{2(D+C)}.$$

E is the estimated delay, D is the average disconnection time, and C is the average connection time. With this computation, α and β are tuning parameters, where a higher value for the parameter means that the corresponding metric will react more slowly to changes, but will be more robust to short-term perturbations. We distinguish the two parameters as the behavior of connection and

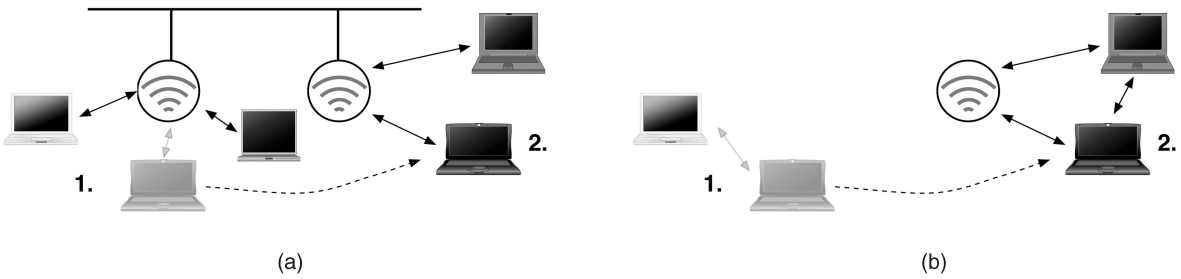

Fig. 5. A wireless LAN trace converted into a DTN scenario. A user disconnects at time 1 and reconnects at time 2. (a) Wireless LAN scenario. (b) Ad hoc DTN scenario.

disconnection times may differ. However, in our simulations, we have maintained $\alpha = \beta = 0.9$. With a periodic contact, as shown in Fig. 4c, the metric will match the exact average.

This implementation has the advantage that it requires fixed resources to implement. Unfortunately, it also has a built-in tendency to underestimate the average waiting time. The reason is that the averaging of D weights all disconnection periods equally. However, the exact expected delay computation weights long disconnections much more than small disconnections because, in those periods, a message must wait longer, on average, and because there is a higher probability of arriving during a long wait period.

The figures in this section show that the differences between these three metrics are relatively small. Thus, we use the infinite window metric unless otherwise specified.

4 Scenarios

In order to validate our requirement of providing good performance over a range of connectivity patterns, we present simulation results from two very different scenarios. To give the scenarios a basis in reality, we used real mobility data. The first scenario is based on extensive wireless LAN traces from Dartmouth College [30]. The second scenario uses the schedule from Seattle's bus network [31].

4.1 Wireless LAN Scenario

In this scenario, mobile users carry computing devices with radios. When they are in range of another user, they exchange data. This represents an ad hoc DTN that could be created by students at a school, employees at a company, or people attending a conference. We used mobility data from Dartmouth College's extensive wireless LAN traces [30]. They record the network connectivity of more than 2,000 users over two years. The traces show when each user connects and disconnects to any of Dartmouth College's 500 access points. This data is useful because, while the mobility appears to be random, there are patterns that can be exploited.

To transform the wireless LAN traces into an ad hoc DTN, we consider two nodes to be connected when they are associated with the same access point at the same time. Access points are also DTN routers. In order to make the scenarios a manageable size, only a connected subset of the nodes from the wireless LAN trace are included. As an example, consider the wireless LAN scenario shown in Fig. 5a. At time 1, the trace file will show that the laptop user is connected to the access point on the left. Later, it moves out of range of the access point and, at time 2, it associates with the second access point. A DTN scenario that could be generated from this data is shown in Fig. 5b. One laptop and one access point from the original scenario were removed.

The raw Dartmouth data set is much too large for us to simulate. To select a subset of the data that our simulator could process, we used the data from a single month (February 2003). We then randomly selected an initial node. Next, we found that node's "good" neighbors. We defined a "good" neighbor as one that had at least 10 opportunities to communicate over a period of one month. Then, we randomly selected a new node from the good neighbor set and added that node's good neighbors to the set. We repeated this until we had 30 nodes and we generated 10 different topologies in this fashion. This simple algorithm guarantees that we do in fact have a connected graph, where each node has some opportunities to communicate with the rest of the network. In order to eliminate the warm-up and cool down effects, we only record statistics for messages generated in the second week.

4.2 Bus Scenario

This scenario represents a city that is providing a DTN network by equipping its buses with wireless devices. As the buses come within wireless range of each other, they exchange messages. To generate this data, we used the Seattle bus network schedule, as provided by the University of Washington's Intelligent Transportation Systems group [31]. This region's bus system is quite large, so, again, we needed to select a subset of it. To do this, we selected all the buses that ever service a single route as part of their day. This includes 36 buses, which is comparable in size to the parameters used for the Dartmouth data. We compute all the times when the buses are within wireless range, which we considered to be 200 meters, the nominally quoted range for 802.11b. The scenario lasted for five weekdays and each day's schedule was identical. We only recorded statistics for messages generated during the second day.

4.3 Comparison

These two scenarios were selected because they represent very different connectivity patterns. The wireless LAN scenario has unpredictable mobility, with some statistical regularity. The bus scenario, on the other hand, is a planned and scheduled system and, thus, has the exact same

TABLE 1
Comparison of the Wireless LAN and Bus Scenario Parameters

Parameter	Wireless LAN Scenario	Bus Scenario
Scenario Duration	28 days	5 days
Nodes	30	36
Avg. Node Degree	15.6	22.8
Avg. Up Time	2050 seconds	77.0 seconds
Avg. Inter-contact Time	9.27 hours	5.32 hours
Avg. Total Connected Duration	23.7 hours	0.405 hours
Avg. Num. Connections	41.6	20.0

connectivity on each of the five days. Additionally, the nature of the connectivity is very different between the scenarios. To quantify this difference, some scenario metrics are shown in Table 1. The scenario duration is obviously different, as they were generated from different data sets. The number of nodes is comparable in the two scenarios. The average node degree is the average number of contacts that appear at least once during the scenario. The bus scenario's average degree is larger, even accounting for the fact that there are more nodes in the scenario. This indicates that the nodes tend to be connected to more neighbors than in the wireless LAN scenario. The up time is the time where each contact is available, the intercontact time is the time where each contact is down, and the total connected duration is the total up time for a single contact. These values are much, much longer in the wireless LAN scenario because users carrying laptops tend to stay in one place and move slowly, whereas the buses only stop for breaks and move quickly. Finally, the number of connections is the number of times a single contact becomes available. Counter-intuitively, this is higher in the wireless LAN scenario. The reason is that there are some contacts which go up and down a large number of times in a short period of time. These anomalies are likely caused by poor wireless LAN connectivity and not frequent mobility.

5 Microbenchmarks

In order to understand the performance of our protocol, we first consider microbenchmarks designed to investigate specific aspects of the performance of the protocol. First, we look at the performance under ideal conditions before varying buffer space and bandwidth. Next, we consider the MEED metric and variants. Third, we closely examine the performance of per-contact routing and hop-by-hop flow control. Finally, we study the overhead of our protocol.

We compare the performance of the MEED protocol to three other delay-tolerant network routing protocols: the earliest delivery (ED) and minimum expected delay (MED) metrics [9] and epidemic routing [10]. The ED protocol is used to illustrate the performance that can be achieved if complete contact schedule data is available. MED is presented because it uses the same average-delay metric as MEED, except that its values are computed using future knowledge. Finally, epidemic is another protocol that does not require schedule information. For the evaluation, we created DTNSim2, a discrete-event simulator for delay-tolerant networks [32]. It is based on the simulator used for the original DTN routing paper by Jain et al. [9]. It uses FIFO, reliable links with fixed bandwidth, and delay.

To provide a baseline for comparison, we measure the performance of an "ideal" protocol. This protocol is the ED metric with infinite buffer space and infinite bandwidth. This protocol has two attractive properties. First, if it cannot deliver a message, then it is not possible for that message to be delivered by any protocol. Thus, it provides an upper bound on the delivery ratio. For this reason, we express all delivery ratios as a percentage of the messages delivered by the ideal protocol. Second, when it delivers a message, it is not possible for that message to be delivered earlier. Thus, it provides a lower bound for delay. When computing average delay, we only include delivered messages.

In order to quantify the cost of the protocols, we measure the total number of bytes transmitted. This includes protocol bytes and data bytes and measures the total amount of bandwidth consumed by the protocol. In order to establish a minimum value, we modify the ideal protocol to select minimum hop-count paths for each message. Thus, it delivers all the messages with the minimum number of transmissions.

In this section, we run the simulator with an ad hoc e-mail workload. Each node generates 10 messages at regular intervals, sent to a single random destination node. Each message is 10,000 bytes long, which corresponds to users exchanging small files or e-mail messages. For the wireless LAN scenario, the interval is every 6 hours, for a total of 112 message generation times. For the bus scenario, the interval is every hour, for 120 message generations.

5.1 Ideal Performance

First, we consider the performance of the four protocols under ideal conditions. We simulate the two scenarios with infinite buffer space and infinite bandwidth. The delivery ratios are shown in Fig. 6a. For all scenarios with infinite resources, ED is the same as the ideal protocol and, by definition, delivers all the messages. Epidemic delivers 100 percent of the messages in these scenarios by flooding the network. For the bus scenario, all protocols deliver all the messages because the bus schedule is predictable and repetitive. In the wireless LAN scenario, MEED manages to deliver little more than 80 percent of the messages. Considering that this protocol has no future knowledge and the mobility in this scenario is random, this is respectable. It is important to recall that, while the ED and MED protocols outperfom MEED, in reality, it is not possible to use them in the wireless LAN scenario because it is not possible to have a schedule of human mobility in advance. The MED protocol, which uses the same metric as MEED except with future information, delivers around 90 percent of the messages simply because its metric is more accurate because it is computed using future knowledge. MEED and MED are not able to deliver all the messages because, in some cases, there is only a single path to the destination that requires very tight timing. These protocols try to select the best average path, which may not be good enough when a node is only connected a small number of times.

The delay results, shown in Fig. 6b, are more interesting. This figure makes the differences between the two scenarios

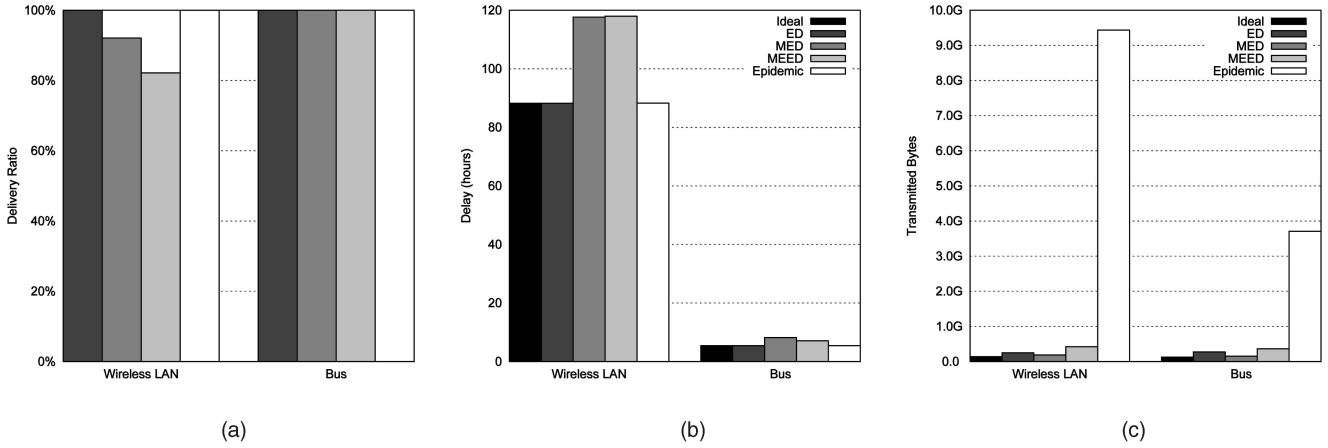

Fig. 6. Performance under ideal conditions. (a) Delivery ratio. (b) Delay. (c) Transmitted bytes.

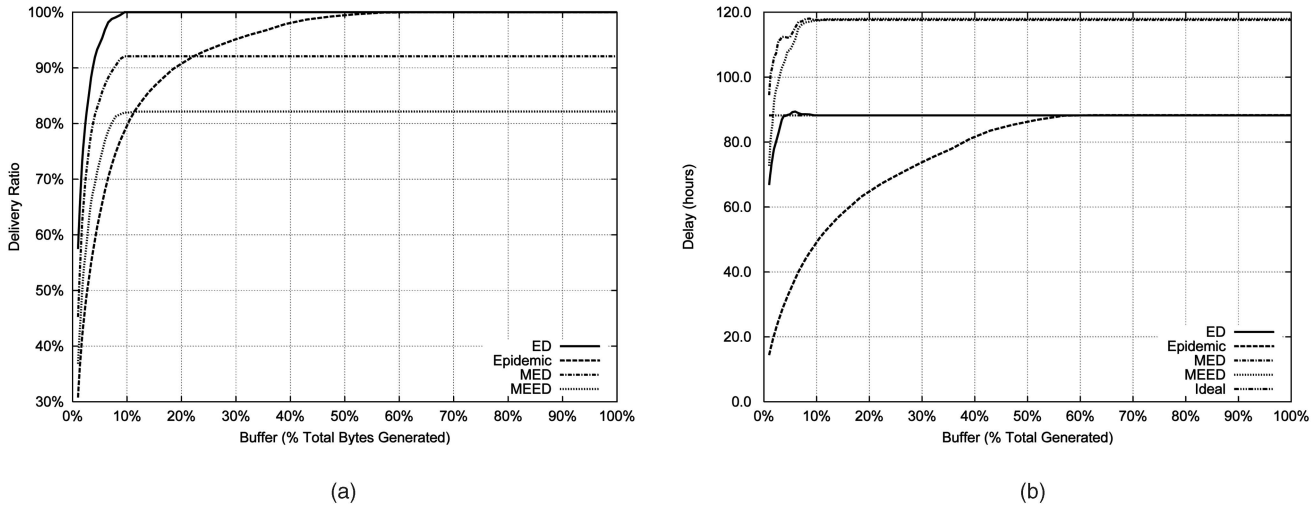

Fig. 7. Wireless LAN scenario with varying buffer size. (a) Delivery ratio. (b) Delay.

very clear, as the ideal average delivery delay for the wireless LAN scenario is around 85 hours, whereas it is under 10 for the bus scenario. ED and epidemic match the ideal delay, as expected. MEED's delay is larger than the ideal by a fair margin. However, it compares favorably with MED, matching it in the wireless LAN scenario and slightly beating it in the bus scenario.

We plot the number of transmitted bytes in Fig. 6c. This graph shows that epidemic transmits the most messages by a very wide margin. This is not surprising as it floods the network with each message. MEED transmits more messages than the other shortest path protocols. This is because the MEED metric is constantly being recomputed, so it is possible that a message will take a bad path only to be forced to backtrack later. This problem is emphasized in the wireless LAN scenario because the connectivity is random. This is the reason that MEED requires significantly more transmissions in that scenario than in the bus scenario. Additionally, ED and MED have no protocol overhead, as they assume that all nodes have the topology information from the start. ED transmits more than the ideal number of bytes because it finds minimum latency paths, while the minimum number of bytes are transmitted when using minimum hop-count paths. Occasionally, the fastest path requires additional hops.

5.2 Impact of Buffer Size

Next, we evaluate the impact of buffer space on the DTN routing protocols. We evaluate each scenario with infinite bandwidth contacts, but limited buffer space. The amount of buffer space is shown as a percentage of the total number of bytes generated. For this experiment, we only consider the wireless LAN scenario because the results for the bus scenario are similar. Looking at the graph of the delivery ratio in Fig. 7a, we can immediately see that buffer size has a significant impact on the epidemic protocol. This is because it relies on having a sufficient buffer to have a copy of every message at every node. In this particular scenario, it needs buffer for approximately 60 percent of the data generated, and there is a very predictable relationship between the buffer size and the delivery ratio. The other protocols require much less buffer space because they use a single copy of each message. Only when the buffer size drops below 10 percent of the total traffic generated does the delivery ratio of the other protocols decrease. A buffer-constrained network is similar to a well-provisioned

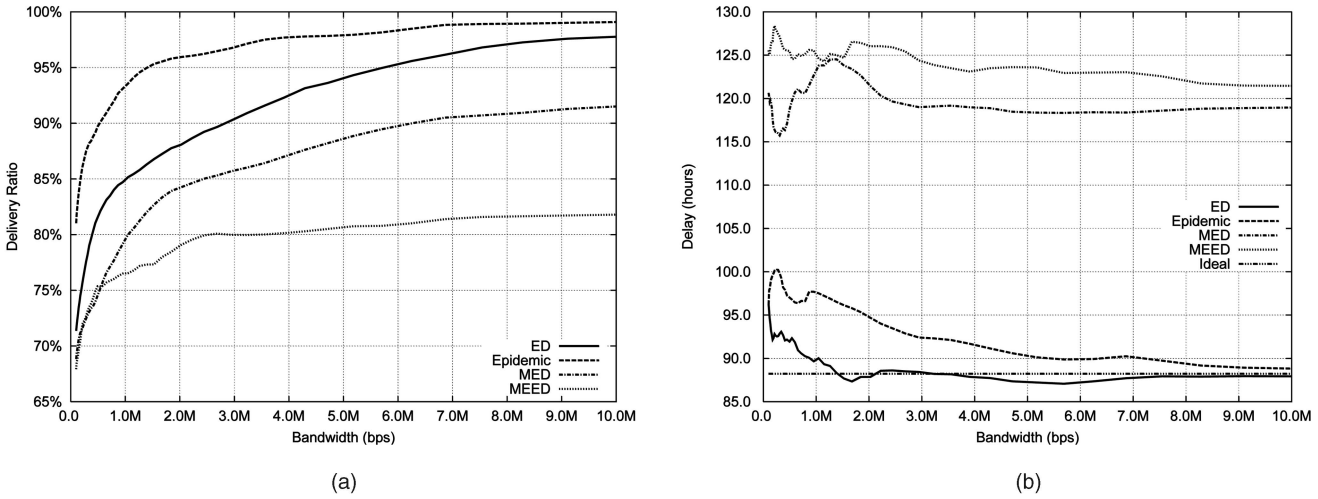

Fig. 8. Wireless LAN scenario with varying link bandwidth. (a) Delivery ratio. (b) Delay.

network that is being used to tranfer large amounts of data. In this situation, MEED has better performance than epidemic routing, which is the only other protocol that does not use future information.

Looking at the latency results in Fig. 7b, we can see that the delay is not sensitive to the buffer space as the performance does not change much. The average delay for the ED and epidemic protocols matches the ideal minimum latency, as expected. Epidemic's delay decreases as it drops messages because the average latency calculation only includes delivered messages and it drops the oldest messages first. Thus, the delivered messages are those that are delivered quickly. Interestingly, the latency for MED and MEED are very similar even though MEED is an estimate of the MED value and, intuitively, should be worse. The reason is that it does not deliver as many messages, so it is not valid to compare the two directly without taking into account the delivery ratios in Fig. 7a.

5.3 Impact of Bandwidth

For this experiment, all nodes have infinite buffer space but the contacts have varying data rates. We test the rates between 100 kbps and 10 Mbps. Again, we only present the results from the wireless LAN scenario as the bus scenario results do not provide any additional insight. It is important to note that, because the average connection times are so long for the wireless LAN scenario, a large amount of data can be transferred even at very low data rates. The e-mail workload generates a small amount of data (336 MB), so, assuming that all the contacts are up for the average connection time shown in Table 1, a data rate of 1.31 Mbps is sufficient to deliver all the messages that are generated in the scenario over a single hop. Thus, it is important to note that it is only at data rates below 2 Mbps that this scenario begins to become bandwidth limited.

For all four protocols, the delivery ratio decreases slightly as the bandwidth decreases, as shown in Fig. 8a. However, there is no change in their relative positions until the bandwidth is extremely low. It is important to note that ED and MED are more sensitive to the reduced bandwidth than epidemic and MEED. With infinite bandwidth, ED was able to deliver 100 percent of the messages, as shown on the right-hand side of Fig. 7a. However, even with 10 Mbps links, it drops approximately 2 percent of the messages and it drops 15 percent of the messages with 1 Mbps links. MEED's performance, by contrast, is nearly flat. It only drops an additional 4 percent of the messages over the same interval. The reason is that ED and MED use source routing and select paths assuming that there is no competing traffic. With the bandwidth restrictions, messages might miss the contact they were supposed to take and be undeliverable. Epidemic, on the other hand, tries all paths. Thus, if one contact is being used, it will try others. Similarly, MEED's per-contact routing allows it to adapt to the varying conditions.

The results for the latency in Fig. 8b show that the delay for all protocols increases slightly as the bandwidth decreases due to the longer transfer times. This trend is probably understated, as all the protocols drop messages as the data rate decreases. This graph shows that MEED's performance is slightly worse than MED, as expected. At data rates below 1 Mbps, all the protocols behave erratically. The reason is that this is the point where the protocols really begin to drop significant numbers of messages, so the data points here are all measuring the delay for slightly different sets of messages. MED shows an unusual bump around 2 Mbps, where its delay increases before decreasing again. The increase is due to the bandwidth restrictions increasing the average delay for most of the messages. The decrease is then caused by the delivery ratio decreasing since, at that point, MED is only able to deliver the messages which take the least time to arrive at the destination.

5.4 MEED Metric Error

The previous results have shown that MEED can achieve delivery ratios and delays comparable to, and sometimes exceeding, existing approaches, without their disadvantages. This implies that it is a reasonable metric for selecting paths. Since this metric estimates a physical quantity, average delay, we wished to test how useful this metric might be for predicting delivery delay. To answer this question, we

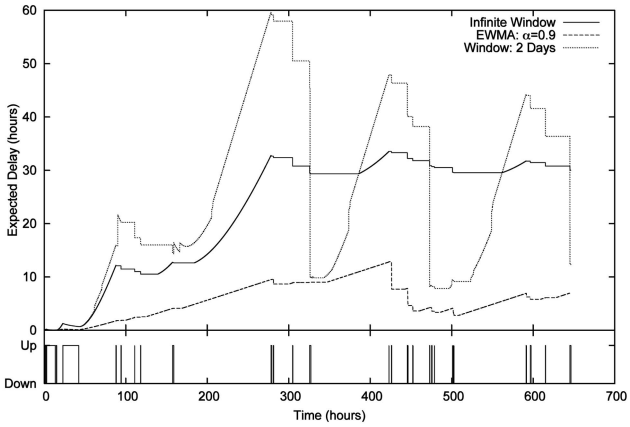

Fig. 9. Sample expected delay metrics for a single contact.

analyzed the performance of the infinite window metric, the sliding window metric $(\text{window} = 2 \text{ days})$, and the EWMA metric $(\alpha = 0.9)$ over a single hop. We selected 30 random contacts from the wireless LAN scenario, sampled the metric values at one minute intervals, and compared them to the actual waiting time. Sample metric values for a single contact are shown in Fig. 9. In order to make the error between different contacts comparable, we divided the absolute error by the average waiting time for each contact. A histogram of the relative errors is shown in Fig. 10. This histogram shows that most of the samples fell between -4 and 2 average waiting times of the true value. This is a large range of errors, which indicates that none of the metric variants are precise. Additionally, the histogram clearly shows that the infinite window metric has a tendency to overestimate the waiting time, while the other two metrics tend to underestimate. Both the sliding window and EWMA variants have a peak at zero error, which is what we would ideally like to see. However, their distribution clearly underestimates the delay, on average. The infinite window metric peaks a bit above zero, but, due to the shape of the distribution, its overall average is somewhat closer to zero.

We also evaluated the metric's performance over complete paths. We instrumented the simulator to record the

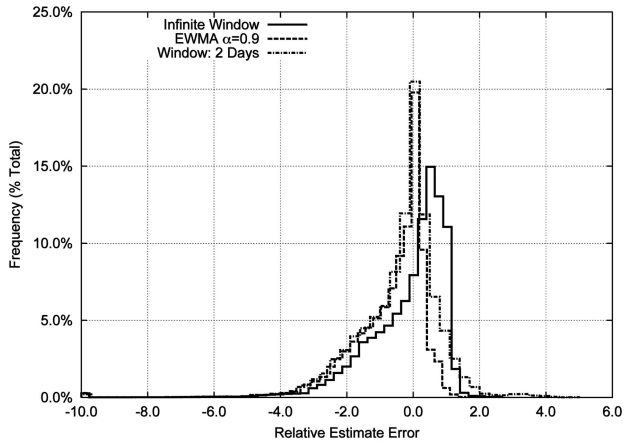

Fig. 10. Histogram of relative estimation error for 30 contacts.

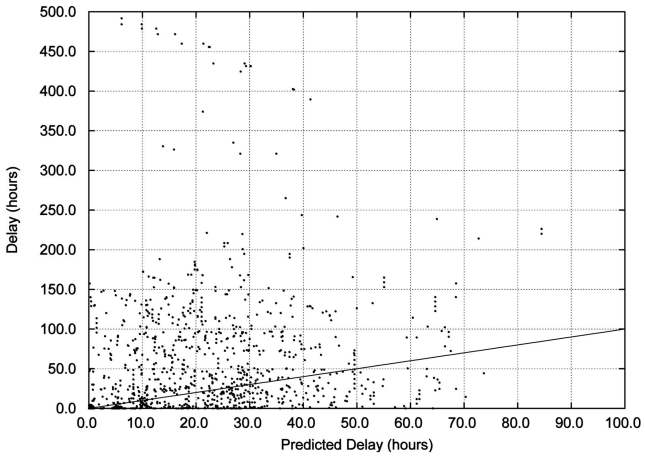

Fig. 11. Scatter plot comparing estimated and measured end-to-end delay.

MEED metric value when the message is forwarded from the source to another node. This is the latest time that the source can make an estimate and, thus, should be the most accurate estimate available to that node. We plotted the prediction for each message against the actual recorded end-to-end latency. Ideally, these two values would be equal and all the points on the $y = x$ line. The results for the infinite window metric from the wireless LAN scenarios are shown in Fig. 11. This figure shows that there is limited correlation between the MEED value and the actual end-to-end delay, as the points appear to be randomly distributed.

The results in this section indicate that the ability for MEED to estimate the end-to-end delay is not ideal. Part of this limitation is simply because it is an average. Consider a contact that has two different behaviors, one where the waiting time is small, and another where the waiting time is very large. The exact average for this contact will be a bad estimate of the end-to-end delay for both sections, although the average errors will be zero. This suggests that more research is required here. It may be possible to estimate a worst-case end-to-end delay by incorporating the variance or a confidence estimate. Alternatively, time-varying statistical or machine learning approaches may be more accurate.

5.5 MEED Variants

The previous results all used the infinite window variant of the MEED metric. Here, we will consider the two additional variants presented in Section 3.6. Additionally, in order to determine what impact the choice of metric has on the performance, we also implemented a fourth metric variant that is intentionally a bad choice. This metric uses the time since the contact was last available and, so, we call it the "Last Up" metric. Intuitively, this should not help in selecting a good path. In fact, in many scenarios, the last available contact will not be seen for a long time and, so, this will be a poor choice.

The delivery ratios for all four variants and both scenarios are shown in Fig. 12a. Interestingly, in the wireless LAN scenario, the Last Up metric actually delivers more messages than the infinite window and EWMA

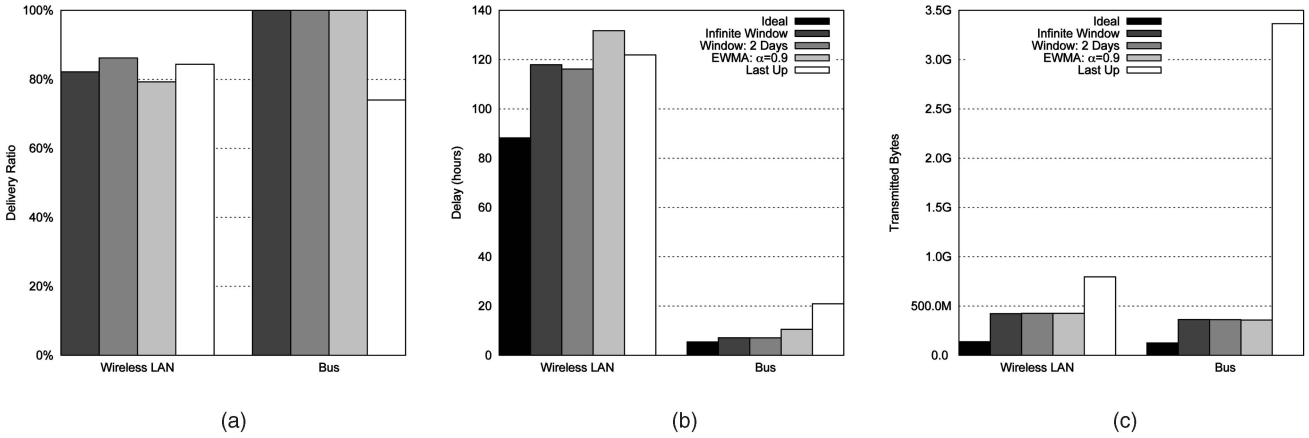

Fig. 12. Performance of MEED variants. (a) Delivery ratio. (b) Delay. (c) Transmitted bytes.

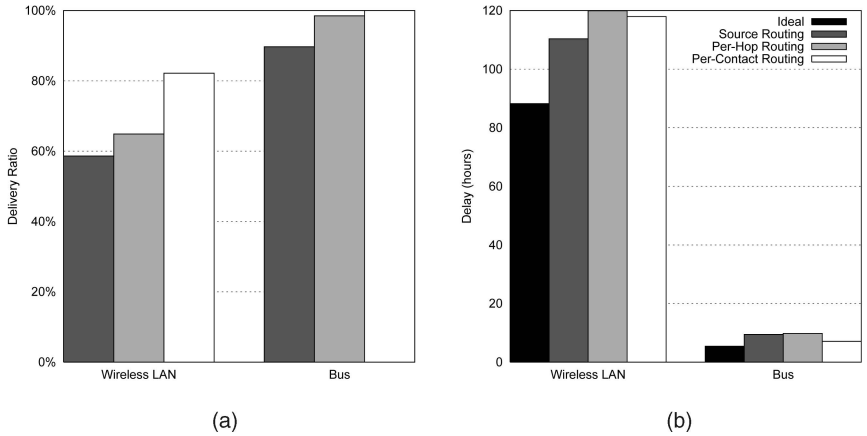

Fig. 13. Comparison of MEED with source, per-hop, and per-contact routing. (a) Delivery ratio. (b) Delay.

metric. The reason for this becomes clear when looking at the transmitted bytes, shown in Fig. 12c. The Last Up metric transmits far more bytes than the others. Since the metric changes so often, it effectively ends up forwarding the message along nearly every available contact, which explains the large number of transmissions. In the wireless LAN scenario, because the mobility is pseudorandom, this strategy actually works fairly well.

Last Up performs significantly worse in the bus scenario. The reason is that this scenario is one where Last Up is exactly the wrong choice. When a bus passes another one, it will be a long time before it passes it again, as, generally, they are traveling in opposite directions. Thus, in this case, the message ends up getting passed to buses that have just passed the destination and will not see it for a while. If it remained on one bus, the messages would likely end up being delivered.

For the other three metrics, the performance is very similar. They are each slightly better than the others in some aspect and worse in another, with no clear winner. However, it is clear that changing the metric can have a significant impact on system behavior, as shown by the Last Up metric. Thus, it may be possible to improve the performance of our system simply by substituting a superior metric.

5.6 Routing Decision Time

In Section 3.2, we argued that making decisions as late as possible is advantageous in delay-tolerant networks, as it allows the protocol to adapt to changes in both the estimate of metric values and network topology. In order to investigate the performance impact of this design choice, we implemented two variants of the MEED protocol: source routing and per-hop routing. The source routing variant plans the entire path as soon as possible at the source, after which no changes are made. The per-hop routing variant calculates the next hop when the message arrives at a new node. At that point, the message can only be sent to that next hop. The delivery ratio for these variants is shown in Fig. 13a, and the delay is shown in Fig. 13b. These graphs show that per-contact routing is significantly better. In the Wireless LAN scenario, per-contact routing delivers a third more messages than source routing. The delay increases because more messages are being delivered. In short, per-contact routing is a key part of the MEED protocol's performance.

We also studied the impact this design decision has on other protocols to show that it is applicable to other protocols. We modified the MED protocol to support per-contact routing instead of source routing. This means that, if a contact becomes available that does not have the best

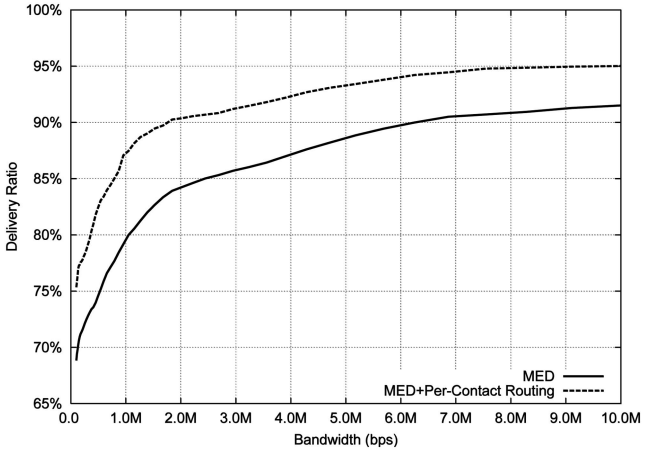

Fig. 14. Comparison of MED with and without per-contact routing.

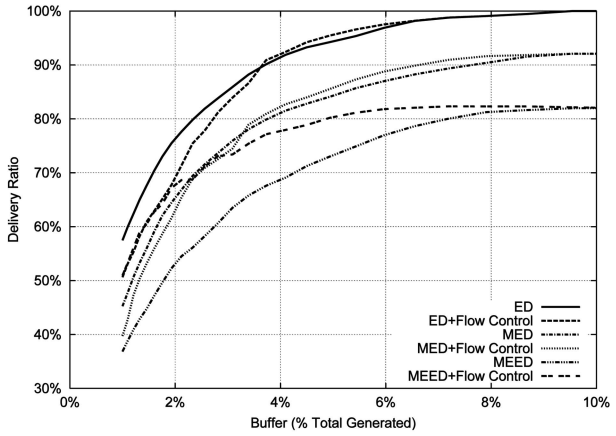

Fig. 15. Wireless LAN performance with hop-by-hop flow control and limited buffer space.

average performance, but is a good choice when it is available, the MED per-contact protocol can take advantage of it. This changes the performance of MED for all scenarios, since it has the most impact when there are bandwidth limits. Thus, we plot the delivery ratio of this protocol when there is infinite buffer space but limited bandwidth in Fig. 14. This figure shows that per-contact routing improves MED in two ways. First, in increases the raw delivery ratio by around 4 percent. Second, it decreases MED's sensitivity to the low bandwidth, as its delivery ratio decreases less than MED with source routing. Per-contact routing also decreases the average delay from 119 hours to 111 hours. It achieves these gains with some cost: It increases the number of transmitted bytes from 190 MB to 260 MB.

5.7 Hop-by-Hop Flow Control

The shortest-path protocols studied here use a "drop tail" queue policy. If there is insufficient buffer space when a message arrives, it is dropped. This source of loss could be reduced by using hop-by-hop flow control, so that the message is only forwarded if sufficient buffer is available. This scheme has been shown to be effective at dealing with congestion in wireless sensor networks [33].

The performance with and without flow control for the wireless LAN scenario with limited buffer space is shown in Fig. 15. For ED and MED, flow control increases the delivery ratio by a few percent when the buffer space is between 3-10 percent of the generated messages. When the buffer is very small, it actually makes things worse. The reason is that, with flow control, if the next hop is full, there is now less buffer space available at the previous node. This can, in turn, cause other nodes to block, which prevents some messages from making progress toward the destination. If the message had been dropped instead, at least one previous message would be able to make forward progress.

For MEED, flow control improves the performance by a significant margin when the buffer space is less than 8 percent of the generated bytes. The difference is that MEED uses per-contact routing, so when one next hop is blocked because the buffer is full, it is able to attempt delivery via an alternative route. This means that it can spread the load across the network. From these results, it seems that hop-by-hop flow control is a significant improvement for dealing with low-buffer scenarios, although care must be taken when the available buffer space is extremely limited.

5.8 Protocol Overhead

In order to measure the overhead introduced by the epidemic distribution of the link-state tables, we generated 10 topologies from the Dartmouth data with different sizes. We simulated them for one month without any traffic and discarded the overhead in the first week. The protocol is initiated each time a connection is established, so if a scenario has more connections, it will generate more overhead. To compensate for this, we normalized the overhead by dividing the total protocol bytes by the total number of connections. This gives us the average bytes of overhead that are exchanged each time a connection is established.

The average overhead with the 90 percent confidence interval is shown in Fig. 16. The theoretical analysis showed that the overhead grows linearly with the size of the network, assuming that the node degree stays constant. The overhead here appears to grow slightly faster than linearly.

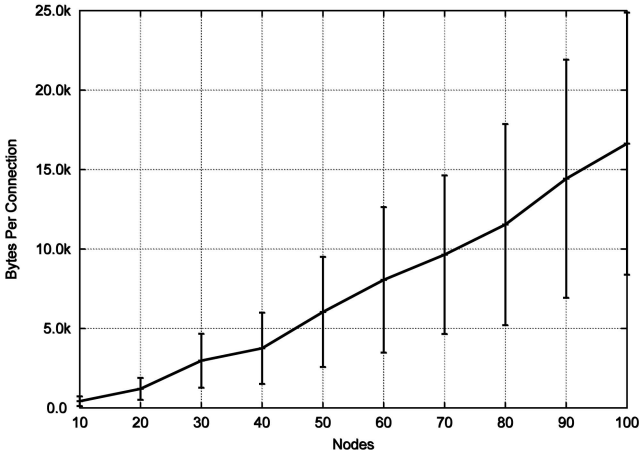

Fig. 16. Protocol overhead per connection.

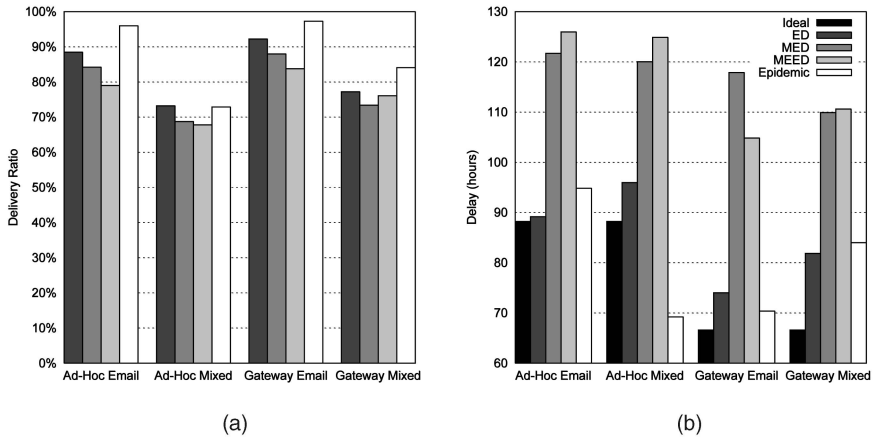

Fig. 17. Wireless LAN scenario performance under realistic conditions. (a) Delivery ratio. (b) Delay.

The reason is that, in our scenario, nodes that share an access point form a clique, which means that adding nodes tends to increase the average node degree.

The actual amount of overhead data exchanged is small relative to the amount of message data that can be exchanged during a contact. With a network of 50 nodes, less than 10 kB is exchanged each time a connection comes up. This will take less than a tenth of a second to transmit at 802.11's base rate of 2 Mbps. This is a tiny portion of the bandwidth. Recall from Table 1 that, even in the bus scenario, the average up time is 77 seconds. With 100 nodes, the average overhead per connection is less than 25 kB, which is still less than a second of transmission time. However, the overhead could become unmanageable as the network grows. This suggests that hierarchical routing may be necessary for very large networks.

6 Realistic Simulations

The previous section looked at the performance of the protocol under various ideal conditions to understand the impact of individual parameters on the performance of our protocol. In this section, we look at the performance of four workloads with plausible combinations of parameters. For each workload, we set the buffer size on each node to 1 GB because the current price of a 1 GB flash disk is under $20 USD, so it is reasonable to assume that even small embedded devices can be equipped with at least this much storage. We set the bandwidth of each contact to 2 Mbps since that is the base rate for 802.11b and, thus, represents a lower bound of the bandwidth available between two nodes within wireless range. Experiments using 802.11 in drive-by scenarios report a wide range of average goodputs. The results vary from an average goodput of 5.5 Mbps [34], down to 0.9 Mbps [6], [16], depending on speeds and antenna location. Thus, 2 Mbps seems to be a realistic midpoint. Additionally, it is safe to assume that future wireless technology improvements will push that data rate even higher.

The scenarios are created by varying two parameters: traffic composition and communication pattern. For traffic composition, we have an e-mail workload and a mixed message workload. The email workload is the one that was used in the previous section, where each node generates 10 messages that are each 10,000 bytes long. The mixed-message workload only generates five messages of 10,000 bytes, but also generates two messages that are 1,000,000 bytes each, representing larger files being exchanged, such as digital photos. The mixed-message workload generates 20 times more traffic than the e-mail workload.

For the communication pattern, we use an ad hoc pattern and a gateway pattern. The ad hoc pattern is the same as was used in the previous section, representing users in the network communicating with each other. The gateway pattern represents users communicating with a single Internet gateway. To simplify the scenario, we assume that the gateway node has an infinite bandwidth and zero latency connection to the Internet, so all outbound and inbound messages are delivered instantly. We selected the node with the highest degree as the gateway since it is the node with the best connectivity to the rest of the network.

6.1 Wireless LAN

The results for the four workloads in the wireless LAN scenario are shown in Fig. 17. While they are generally similar, a few differences can be observed. The epidemic protocol consistently delivers more messages than any other protocol. In the mixed-message workload, the performance of all the protocols decreases due to the increased amount of traffic in the network. However, MEED's performance decreases less than the others, to the point that it outperforms MED in the gateway workload. MEED's ability to adapt to changing network conditions gives it a distinct advantage in this scenario, where there is a significant amount of network congestion. Unfortunately, MEED still has a longer delay than the other protocols, except in the gateway e-mail case where it has lower delay than MED.

6.2 Bus

The bus scenario has a higher average node degree than the wireless LAN scenario. Correspondingly, the average path

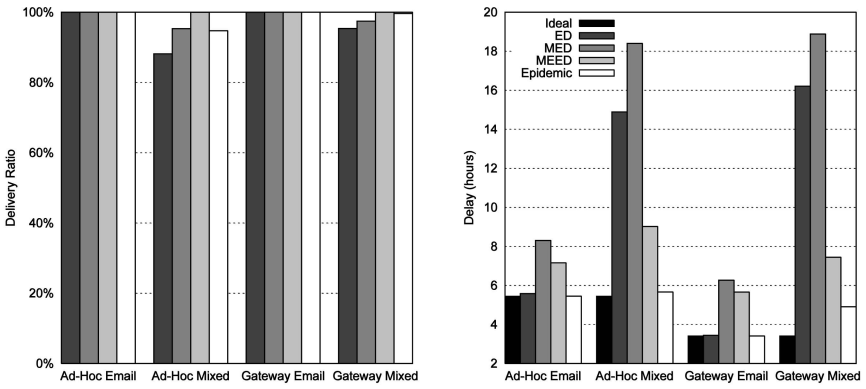

Fig. 18. Bus scenario performance under realistic conditions. (a) Delivery ratio. (b) Delay.

length is also lower. This means that bandwidth and buffer restrictions typically have less impact on the performance in this scenario, as many of the deliveries occur directly from the source to the destination. Thus, the delivery ratios are still 100 percent for all the protocols with the e-mail workloads, as shown in Fig. 18a. However, this scenario also has a very short average connection time when two buses pass each other on the road. When there are larger messages in the network, as is the case with the mixed workload, the performance changes dramatically. With this workload, MEED outperforms the other protocols. The reason is that ED and MED plan their routes with the assumption that there is no competing traffic. This assumption is violated with this workload, as a single large message can occupy almost all of a contact's time. Epidemic is not able to deliver all the messages in this scenario as there is only 11 percent buffer space and because the contacts do not last long enough to exchange all the messages.

As for the delay, shown in Fig. 18b, MEED outperforms MED for all the bus scenarios. For the gateway scenarios, MEED's delay is second only to the epidemic protocol. In the other cases, it is only an hour or two more than the ideal minimum delay.

7 Conclusions and Future Work

In this paper, we introduced the minimum estimated expected delay (MEED) path metric, which uses observed information to estimate waiting times for each contact. We presented an epidemic protocol for propagating topology updates through a delay-tolerant network. The result is a routing system that can deliver data in a DTN without any knowledge about the communication schedules. The only other protocols with this property are flooding protocols that rely on sending many copies of each message. We have shown through simulations that it approaches, and, in some cases, exceeds, the performance of protocols that have complete knowledge of the network topology. This shows that it is feasible to use shortest path routing in delay-tolerant networks.

Epidemic routing also does not require topology information, and the results show that it performs very well. However, MEED achieves a significant fraction of epidemic routing's delivery ratio using only a single message, instead of one copy for every node. This is much more efficient. It also suggests that it should be possible to use the MEED metric to selectively send a small number of duplicates in order to achieve reliable delivery at a low cost.

We presented the concept of per-contact routing, where the routing tables are recomputed every time a connection is made. This permits the routing to react to topology changes and take advantage of opportunistic contacts. Indeed, the results show that this improves the latency and delivery ratio significantly for shortest path routing protocols. We also showed that hop-by-hop flow control is an essential strategy for dealing with temporary buffer shortages.

The results presented here show that, while the MEED metric is capable of selecting paths that achieve performance comparable to, and sometimes better than, existing techniques, there is certainly room for improvement. In particular, the MEED approach assumes that there is a consistent delay that can be determined, where, in many DTN scenarios, this may be a more complex pattern. For example, in the bus scenario, delay is likely to vary with time of day. There are many opportunities to apply advanced statistical or machine learning techniques here. Additionally, this work has only looked at attempting to minimize end-to-end delay. There may be other metrics that can yield better performance. Our benchmarks show that the protocol overhead is a very small fraction of total contact duration for networks of hundreds of nodes. However, more efficient schemes will be required for very large networks composed of thousands of nodes.

We believe that the most important contribution that can be made to delay-tolerant routing is to build real networks and applications. This is the only way to determine the practical requirements for routing protocols. Protocols that require no configuration, like the one presented here, can facilitate this process by reducing the amount of effort required to deploy and extend these networks.

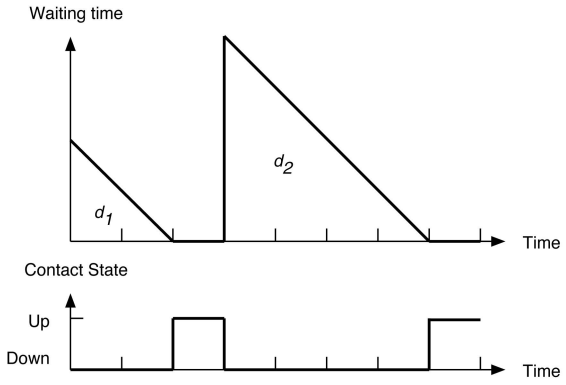

Fig. 19. Example contact waiting time and state.

APPENDIX

DERIVATION OF THE EXPECTED DELAY

The expected delay for a contact is computed assuming that all arrival times are equally likely. When the contact is up, the waiting time is zero. When the contact is down, the waiting time is the time until the contact comes back up again, as shown in Fig. 19. Since the arrival time probability distribution is uniform, to compute the expected value of the waiting time, we can compute the area under the curve and then divide by the length of the time interval. For a single disconnected interval, d_i, the area under the curve is given by $\frac{1}{2}d_i^2$. The area under a connected interval is 0. Thus, the final metric is given by

$$\begin{aligned} \text{MEED} &= \frac{\sum_{i=1}^{n} \frac{1}{2} d_i^2}{t} \\ &= \frac{\sum_{i=1}^{n} d_i^2}{2t}. \end{aligned} \tag{2}$$

ACKNOWLEDGMENTS

This work was supported in part by the Natural Sciences and Engineering Research Council of Canada (NSERC) and the Software Telecommunications Group at the University of Waterloo.

REFERENCES

[1] K. Fall, "A Delay-Tolerant Network Architecture for Challenged Internets," *Proc. ACM SIGCOMM,* pp. 27-34, Aug. 2003.
[2] A. Rabagliati, "Wizzy Digital Courier—How It Works," http://www.wizzy.org.za/article/articleprint/19/1/2/, Apr. 2004.
[3] A.A. Hasson, D.R. Fletcher, and D.A. Pentland, "A Road to Universal Broadband Connectivity," *Proc. Development by Design Second Int'l Conf. Open Collaborative Design for Sustainable Innovation (dyd02),* Dec. 2002.
[4] E. Brewer, M. Demmer, B. Du, M. Ho, M. Kam, S. Nedevschi, J. Pal, R. Patra, S. Surana, and K. Fall, "The Case for Technology in Developing Regions," *Computer,* vol. 38, no. 6, pp. 25-38, June 2005.
[5] T. Small and Z.J. Haas, "The Shared Wireless Infostation Model," *Proc. MobiHoc,* pp. 233-244, June 2003.
[6] J. Ott and D. Kutscher, "Drive-Thru Internet: IEEE 802.11b for 'Automobile' Users," *Proc. INFOCOM,* pp. 362-373, Mar. 2004.
[7] S. Burleigh, A. Hooke, L. Torgerson, K. Fall, V. Cerf, B. Durst, K. Scott, and H. Weiss, "Delay-Tolerant Networking: An Approach to Interplanetary Internet," *IEEE Comm. Magazine,* vol. 41, no. 6, pp. 128-136, June 2003.
[8] E.P.C. Jones, L. Li, and P.A.S. Ward, "Practical Routing in Delay-Tolerant Networks," *Proc. ACM SIGCOMM Workshop Delay-Tolerant Networking (WDTN '05),* pp. 237-243, Aug. 2005.
[9] S. Jain, K. Fall, and R. Patra, "Routing in a Delay Tolerant Network," *Proc. ACM SIGCOMM,* pp. 145-158, Oct. 2004.
[10] A. Vahdat and D. Becker, "Epidemic Routing for Partially-Connected Ad Hoc Networks," Technical Report CS-2000-06, Duke Univ., July 2000.
[11] J.A. Davis, A.H. Fagg, and B.N. Levine, "Wearable Computers as Packet Transport Mechanisms in Highly-Partitioned Ad-Hoc Networks," *Proc. Int'l Symp. Wearable Computers,* pp. 141-148, 2001.
[12] A. Lindgren, A. Doria, and O. Scheln, "Probabilistic Routing in Intermittently Connected Networks," *Lecture Notes in Computer Science,* vol. 3126, pp. 239-254, Jan. 2004.
[13] D. Marasigan and P. Rommel, "MV Routing and Capacity Building in Disruption Tolerant Networks," *Proc. INFOCOM,* pp. 398-408, Mar. 2005.
[14] T. Small and Z.J. Haas, "Resource and Performance Tradeoffs in Delay-Tolerant Wireless Networks," *Proc. ACM SIGCOMM Workshop Delay-Tolerant Networking,* pp. 260-267, Aug. 2005.
[15] K. Tan, Q. Zhang, and W. Zhu, "Shortest Path Routing in Partially Connected Ad Hoc Networks," *Proc. Global Telecomm. Conf. (GLOBECOM '03),* pp. 1038-1042, Dec. 2003.
[16] J. Burgess, B. Gallagher, D. Jensen, and B.N. Levine, "Maxprop: Routing for Vehicle-Based Disruption-Tolerant Networking," *Proc. INFOCOM,* Apr. 2006.
[17] A. Demers, D. Greene, C. Houser, W. Irish, J. Larson, S. Shenker, H. Sturgis, D. Swinehart, and D. Terry, "Epidemic Algorithms for Replicated Database Maintenance," *SIGOPS Operating Systems Rev.,* vol. 22, no. 1, pp. 8-32, Jan. 1988.
[18] K.A. Harras and K.C. Almeroth, "Transport Layer Issues in Delay Tolerant Mobile Networks," *Proc. IFIP-TC6 Networking Conf.,* 2006.
[19] T. Spyropoulos, K. Psounis, and C.S. Raghavendra, "Spray and Wait: An Efficient Routing Scheme for Intermittently Connected Mobile Networks," *Proc. ACM SIGCOMM Workshop Delay-Tolerant Networking (WDTN '05),* pp. 252-259, Aug. 2005.
[20] M. Grossglauser and D.N.C. Tse, "Mobility Increases the Capacity of Ad Hoc Wireless Networks," *IEEE/ACM Trans. Networking,* vol. 10, no. 4, pp. 477-486, Aug. 2002.
[21] R.C. Shah, S. Roy, S. Jain, and W. Brunette, "Data Mules: Modeling a Three-Tier Architecture for Sparse Sensor Networks," *Sensor Network Protocols and Applications,* pp. 30-41, May 2003.
[22] W. Zhao, M. Ammar, and E. Zegura, "A Message Ferrying Approach for Data Delivery in Sparse Mobile Ad Hoc Networks," *Proc. MobiHoc,* pp. 187-198, May 2004.
[23] D. Nain, N. Petigara, and H. Balakrishnan, "Integrated Routing and Storage for Messaging Applications in Mobile Ad Hoc Networks," *Mobile Networks and Applications (MONET),* pp. 595-604, Dec. 2004.
[24] J. Lebrun, C.-N. Chuah, D. Ghosal, and M. Zhang, "Knowledge-Based Opportunistic Forwarding in Vehicular Wireless Ad Hoc Networks," *IEEE Vehicular Technology Conf.,* pp. 2289-2293, 2005.
[25] J. Leguay, F. Friedman, and V. Conan, "Evaluating Mobility Pattern Space Routing for DTNs," *Proc. IEEE INFOCOM,* Apr. 2006.
[26] B. Karp and H.T. Kung, "GPSR: Greedy Perimeter Stateless Routing for Wireless Networks," *Proc. MobiCom,* pp. 243-254, Aug. 2000.
[27] R. Handorean, C. Gill, and G.-C. Roman, "Accommodating Transient Connectivity in Ad Hoc and Mobile Settings," *Lecture Notes in Computer Science,* vol. 3001, pp. 305-322, Jan. 2004.
[28] M.J. Fischer, N.A. Lynch, and M.S. Paterson, "Impossibility of Distributed Consensus with One Faulty Process," *J. ACM,* vol. 32, no. 2, pp. 374-382, Apr. 1985.
[29] F. Cristian and C. Fetzer, "The Timed Asynchronous Distributed System Model," *IEEE Trans. Parallel and Distributed Systems,* vol. 10, no. 6, pp. 642-657, June 1999.
[30] Dartmouth College, "The Dartmouth Wireless Trace Archive," http://crawdad.cs.dartmouth.edu/, 2007.
[31] Univ. of Washington, "Intelligent Transportations Systems," http://www.its.washington.edu/, 2007.
[32] Univ. of Waterloo, "DTNSim2 DTN Simulator," http://shoshin.uwaterloo.ca/dtnsim2/, 2007.

[33] B. Hull, K. Jamieson, and H. Balakrishnan, "Mitigating Congestion in Wireless Sensor Networks," *Proc. ACM Conf. Embedded Networked Sensor Systems (SenSys '04),* Nov. 2004.
[34] R. Gass, J. Scott, and C. Diot, "Measurements of 802.11 In-Motion Networking," *Proc. IEEE Workshop Mobile Computing Systems and Applications (WMCSA '06),* Apr. 2006.

Evan P.C. Jones received the BASc and MASc degrees in computer engineering from the University of Waterloo. He is currently working for Google building large-scale distributed systems.

Lily Li received the BS degree in computer engineering from the Beijing Univerisity of Posts and Telecommunications and the MASc degree in computer engineering from the University of Waterloo. Her research interests include wireless mesh networks and delay-tolerant networks. She has years of software development experience with network management, communication systems, Web applications, and automated testing. She is currently with Engenuity Corporation.

Jakub K. Schmidtke received the BS degree from the Warsaw University of Technology, where he is currently a Master's student in the Department of Electronics and Information Technology. He visited the University of Waterloo in 2005/2006 as an exchange student. Currently, he is working on his thesis on delay-tolerant network simulation. His interests include networks and distributed systems.

Paul A.S. Ward received the BScEng degree in electrical engineering from the University of New Brunswick and the PhD degree in computer science from the University of Waterloo. He is currently an assistant professor in the Department of Electrical and Computer Engineering at the University of Waterloo, where he teaches and researches networks and distributed systems. He is a member of the IEEE.