DTN Routing as a Resource Allocation Problem

Aruna Balasubramanian, Brian Neil Levine and Arun Venkataramani
Dept. of Computer Science, University of Massachusetts Amherst
Amherst, MA, USA
arunab@cs.umass.edu, brian@cs.umass.edu, arun@cs.umass.edu

ABSTRACT

Routing protocols for disruption-tolerant networks (DTNs) use a variety of mechanisms, including discovering the meeting probabilities among nodes, packet replication, and network coding. The primary focus of these mechanisms is to increase the likelihood of finding a path with limited information, and so these approaches have only an *incidental* effect on routing such metrics as maximum or average delivery delay. In this paper, we present RAPID, an *intentional* DTN routing protocol that can optimize a specific routing metric such as the worst-case delivery delay or the fraction of packets that are delivered within a deadline. The key insight is to treat DTN routing as a resource allocation problem that translates the routing metric into per-packet utilities which determine how packets should be replicated in the system.

We evaluate RAPID rigorously through a prototype deployed over a vehicular DTN testbed of 40 buses and simulations based on real traces. To our knowledge, this is the first paper to report on a routing protocol deployed on a real DTN at this scale. Our results suggest that RAPID significantly outperforms existing routing protocols for several metrics. We also show empirically that for small loads RAPID is within 10% of the optimal performance.

Categories and Subject Descriptors

C.2 [**Computer Communication Network**]: Network Protocols— *Routing Protocols*

General Terms

Design, Performance

Keywords

DTN, deployment, mobility, routing, utility

This work was supported in part by NSF awards CNS-0133055 and CNS-0519881.

SIGCOMM'07, August 27–31, 2007, Kyoto, Japan.

1. INTRODUCTION

Disruption-tolerant networks (DTNs) enable transfer of data when mobile nodes are connected only intermittently. Applications of DTNs include large-scale disaster recovery networks, sensor networks for ecological monitoring [36], ocean sensor networks [28, 24], vehicular networks [26, 6], and projects such as TIER [2], Digital Study Hall [15], and One Laptop Per Child [1] to benefit developing nations. Intermittent connectivity can be a result of mobility, power management, wireless range, sparsity, or malicious attacks. The inherent uncertainty about network conditions makes routing in DTNs a challenging problem.

The primary focus of many existing DTN routing protocols is to increase the likelihood of finding a path with extremely limited information. To discover such a path, a variety of mechanisms are used including estimating node meeting probabilities, packet replication, network coding, placement of stationary waypoint stores, and leveraging prior knowledge of mobility patterns. Unfortunately, the burden of finding even one path is so great that existing approaches have only an *incidental* rather than an *intentional* effect on such routing *metrics* as worst-case delivery latency, average delay, or percentage of packets delivered. This disconnect between application needs and routing protocols hinders deployment of DTN applications. Currently, it is difficult to drive the routing layer of a DTN by specifying priorities, deadlines, or cost constraints. For example, a simple news and information application is better served by maximizing the number of news stories delivered before they are outdated, rather than maximizing the number of stories eventually delivered.

In this paper, we present a *resource allocation protocol for intentional DTN* (RAPID) routing, which we designed to explicitly optimize an administrator-specified routing metric. RAPID routes a packet by opportunistically replicating it until a copy reaches the destination. RAPID translates the routing metric to per-packet utilities that determine at every transfer opportunity if the marginal utility of replicating a packet justifies the resources used.

RAPID loosely tracks network resources through a control plane to assimilate a local view of global network state. To this end, RAPID uses an in-band *control channel* to exchange network state information among nodes using a fraction of the available bandwidth. RAPID's control channel builds on insights from previous work, e.g., Jain et al. [19] suggest that DTN routing protocols that use more knowledge of network conditions perform better, and Burgess et al. [6] show that flooding acknowledgments improves delivery rates by removing useless packets from the network. RAPID nodes

use the control channel to exchange additional *metadata* that includes the number and location of replicas of a packet and the average size of past transfers. Even though this information is delayed and inaccurate, the mechanisms in RAPID's control plane combined with its utility-driven replication algorithms significantly improve routing performance compared to existing approaches.

We have built and deployed RAPID on a vehicular DTN testbed, DieselNet [6], that consists of 40 buses covering a 150 square-mile area around Amherst, MA. Each bus carries 802.11b radios and a moderately resourceful computer, and the buses intermittently connect as they pass one another. We collected 58 days of performance traces of the RAPID deployment. To our knowledge, this is the first paper to report on a DTN routing protocol deployed at this scale. Similar testbeds have deployed only flooding as a method of packet propagation [36]. We also conduct a simulation-based evaluation using real traces to stress-test and compare various protocols. To ensure a fair comparison to other DTN protocols (that we did not deploy), we collected traces of the bus-to-bus meeting duration and bandwidth during the 58 days. We then constructed a trace-driven simulation of RAPID, and we show that the simulator provides performance results that are are within 1% of the real measurements with 95% confidence. We use this simulator to compare RAPID to four existing routing protocols [23, 31, 6] and random routing. We also compare the protocols using synthetic mobility models.

To show the generality of RAPID, we evaluate three separate routing metrics: minimizing average delay, minimizing worst-case delay, and maximizing the number of packets delivered before a deadline. Our experiments using trace-driven and synthetic mobility scenarios show that RAPID significantly outperforms four other routing protocols. For example, in trace-driven experiments under moderate-to-high loads, RAPID outperforms the second-best protocol by about 20% for all three metrics, while also delivering 15% more packets for the first two metrics. With *a priori* mobility information and moderate-to-high loads, RAPID outperforms random replication by about 50% for all metrics while delivering 40% more packets. We also compare RAPID to an optimal protocol and show empirically that RAPID performs within 10% of optimal for low loads. All experiments include the cost of RAPID's control channel.

In sum, we demonstrate the feasibility of an intentional routing approach for DTNs. Our contributions include the following.

- A utility-driven DTN routing protocol, RAPID, instantiated with three different routing metrics: minimizing average delay, minimizing maximum delay, and minimizing the number of packets that miss a deadline (Sections 3 and 4).
- Deployment and evaluation of RAPID on a vehicular testbed to show performance in real scenarios and to validate our trace-driven simulator (Section 5).
- Comprehensive experiments using a 58-day trace that show that RAPID not only outperforms four other protocols for each routing metric, but also consistently delivers a larger fraction of packets (Section 6).
- Hardness results to substantiate RAPID's heuristic approach, which prove that online algorithms without complete future knowledge and with unlimited computational power, or computationally limited algorithms with complete future knowledge, can be arbitrarily far from optimal (Section 3 and Appendix).

2. RELATED WORK

Replication versus Forwarding.

We classify related existing DTN routing protocols as those that replicate packets and those that forward only a single copy. *Epidemic* routing protocols replicate packets at transfer opportunities hoping to find a path to a destination. However, naive flooding wastes resources and can severely degrade performance. Proposed protocols attempt to limit replication or otherwise clear useless packets in various ways: *(i)* using historic meeting information [12, 7, 6, 23]; *(ii)* removing useless packets using acknowledgments of delivered data [6]; *(iii)* using probabilistic mobility information to infer delivery [30]; *(iv)* replicating packets with a small probability [35]; *(v)* using network coding [34] and coding with redundancy [18]; and *(vi)* bounding the number of replicas of a packet [31, 30, 25].

In contrast, *forwarding* routing protocols maintain at most one copy of a packet in the network [19, 20, 33]. Jain et al. [19] propose a forwarding algorithm to minimize the average delay of packet delivery using oracles with varying degrees of future knowledge. Our deployment experience suggests that, even for a scheduled bus service, implementing the simplest oracle is difficult; connection opportunities are affected by many factors in practice including weather, radio interference, and system failure. Furthermore, we present formal hardness results and empirical results to quantify the impact of not having complete knowledge.

Jones et al. [20] propose a link-state protocol based on epidemic propagation to disseminate global knowledge, but use a single path to forward a packet. Shah et al. [29] and Spyropoulos et al. [33] present an analytical framework for the forwarding-only case assuming a grid-based mobility model. They subsequently extend the model and propose a replication-based protocol, Spray and Wait [31]. The consensus appears to be [31] that replicating packets can improve performance (and security [5]) over just forwarding, but risk degrading performance when resources are limited.

Incidental versus Intentional.

Our position is that most existing schemes only have an *incidental* effect on desired performance metrics, including commonly evaluated metrics such as average delay or delivery probability. Their theoretical intractability in general makes the effect of a particular protocol design decision on the performance of a given resource constrained network scenario unclear. For example, several existing DTN routing algorithms [31, 30, 25, 6] route packets using the number of replicas as the heuristic, but the effect of replication varies with different routing metrics. *Spray and Wait* [31] routes to reduce delay metric, but it does not take into account bandwidth or storage constraints. In contrast, routing in RAPID is *intentional* with respect to a given performance metric. RAPID explicitly calculates the effect of replication on the routing metric while accounting for resource constraints.

Resource Constraints.

RAPID also differs from most previous work in its assumptions regarding resource constraints, routing policy, and mo-

Problem	Storage	Bandwidth	Routing	Previous work (and mobility)
P1	Unlimited	Unlimited	Replication	Epidemic [25], Spray and Wait [31]: Constraint in the form of channel contention (Grid-based synthetic)
P2	Unlimited	Unlimited	Forwarding	Modified Djikstra's algorithm Jain et al. [19] (simple graph), MobySpace [22] (Powerlaw)
P3	Finite	Unlimited	Replication	Davis et al. [12] (Simple partitioning synthetic), SWIM [30] (Exponential), MV [7] (Community-based synthetic), Prophet [23] (Community-based synthetic)
P4	Finite	Finite	Forwarding	Jones et al. [20] (AP traces), Jain et al. [19] (Synthetic DTN topology)
P5	Finite	Finite	Replication	This paper (Vehicular DTN traces, exponential, and powerlaw meeting probabilities, testbed deployment), MaxProp [6] (Vehicular DTN traces)

Table 1: A classification of some related work into DTN routing scenarios

bility patterns. Table 1 shows a taxonomy of many existing DTN routing protocols based on assumptions about *bandwidth* available during transfer opportunities and the *storage* carried by nodes; both are either *finite* or *unlimited*. For each work, we state in parentheses the mobility model used. RAPID is a replication-based algorithm that assumes constraints on both storage and bandwidth (P5) — the most challenging and most practical problem space.

P1 and P2 are important to examine for valuable insights that theoretical tractability yields but are impractical for real DTNs with limited resources. Many studies [23, 12, 7, 30] analyze the case where storage at nodes is limited, but bandwidth is unlimited (P3). This scenario may happen when the radios used and the duration of contacts allow transmission of more data than can be stored by the node. However, we find this scenario to be uncommon — typically storage is inexpensive and energy efficient. Trends suggest that high bitrate radios will remain more expensive and energy-intensive than storage [13]. We describe how the basic RAPID protocol can be naturally extended to accommodate storage constraints. Finally, for mobile DTNs, and especially vehicular DTNs, transfer opportunities are short-lived [17, 6].

We were unable to find other protocols in P5 except *MaxProp* [6] that assume limited storage and bandwidth. However, it is unclear how to optimize a specific routing metric using *MaxProp*, so we categorize it as an incidental routing protocol. Our experiments indicate that RAPID significantly outperforms *MaxProp* for each metric that we evaluate.

Some theoretical works [37, 32, 30] derive closed-form expressions for average delay and number of replicas in the system as a function of the number of nodes and mobility patterns. Although these analyses contributed to important insights in the design of RAPID, their assumptions about mobility patterns or unlimited resources were, in our experience, too restrictive to be applicable to practical settings.

3. THE RAPID PROTOCOL

3.1 System model

We model a DTN as a set of mobile nodes. Two nodes transfer data packets to each other when within communication range. During a transfer, the sender replicates packets while retaining a copy. A node can deliver packets to a destination node directly or via intermediate nodes, but packets may not be fragmented. There is limited storage and transfer bandwidth available to nodes. Destination nodes are assumed to have sufficient capacity to store delivered packets, so only storage for in-transit data is limited. Node meetings are assumed to be short-lived.

Formally, a DTN consists of a node meeting schedule and a workload. The node meeting schedule is a directed multigraph $G = (V, E)$, where V and E represent the set of nodes and edges, respectively. Each directed edge e between two nodes represents a meeting between them, and it is annotated with a tuple (t_e, s_e), where t is the time of the meeting and s is the size of the transfer opportunity. The workload is a set of packets $P = \{(u_1, v_1, s_1, t_1), (u_2, v_2, s_2, t_2), \ldots\}$, where the ith tuple represents the source, destination, size, and time of creation (at the source), respectively, of packet i. The goal of a DTN routing algorithm is to deliver all packets using a feasible schedule of packet transfers, where *feasible* means that the total size of packets transfered during each opportunity is less than the size of the opportunity, always respecting storage constraints.

In comparison to Jain et al.[19] who model link properties as continuous functions of time, our model assumes discrete short-lived transfers; this makes the problem analytically more tractable and characterizes many practical DTNs well.

3.2 The case for a heuristic approach

Two fundamental reasons make the case for a heuristic approach to DTN routing. First, the inherent uncertainty of DTN environments rules out provably efficient online routing algorithms. Second, computing optimal solutions is hard even with complete knowledge about the environment. Both hardness results formalized below hold even for unit-sized packets and unit-sized transfer opportunities and assume no storage restriction.

THEOREM 1. *Let ALG be a deterministic online DTN routing algorithm with unlimited computational power.*

- *(a) If ALG has complete knowledge of a workload of n packets, but not of the schedule of node meetings, then it is $\Omega(n)$-competitive with an offline adversary with respect to the fraction of packets delivered.*
- *(b) If ALG has complete knowledge of the meeting schedule, but not of the packet workload, then it can deliver at most a third of packets compared to an optimal offline adversary.*

THEOREM 2. *Given complete knowledge of node meetings and the packet workload* a priori*, computing a routing schedule that is optimal with respect to the number of packets delivered is NP-hard with an $\Omega(\sqrt{(n)})$ lower bound on approximability.*

$D(i)$	Packet i's expected delay = $T(i) + A(i)$
$T(i)$	Time since creation of i
$a(i)$	Random variable that determines the remaining time to deliver i
$A(i)$	Expected remaining time = E[$a(i)$]
M_{XZ}	Random variable that determines inter-meeting time between nodes X and Z

Table 2: List of commonly used variables.

The proofs are outlined in the appendix and formal proofs are presented in a technical report [3]. The hardness results naturally extend to the average delay metric for both the online as well as computationally limited algorithms.

Finally, traditional optimization frameworks for routing [14] and congestion control [21] based on fluid models appear difficult to extend to DTNs due to the inherently high feedback delay, uncertainty about network conditions, and the discrete nature of transfer opportunities that are more suited for transferring large "bundles" rather than small packets.

3.3 RAPID design

RAPID models DTN routing as a utility-driven resource allocation problem. A packet is routed by replicating it until a copy reaches the destination. The key question is: given limited bandwidth, how should packets be replicated in the network so as to optimize a specified *routing metric*? RAPID derives a per-packet *utility function* from the routing metric. At a transfer opportunity, it replicates a packet that locally results in the highest increase in utility.

Consider a routing metric such as *minimize average delay of packets*, the running example used in this section. The corresponding utility U_i of packet i is the negative of the expected delay to deliver i, i.e., the time i has already spent in the system plus the additional expected delay before i is delivered. Let δU_i denote the increase in U_i by replicating i and s_i denote the size of i. Then, RAPID replicates the packet with the highest value of $\delta U_i/s_i$ among packets in its buffer; in other words, the packet with the highest marginal utility.

In general, U_i is defined as the expected contribution of i to the given routing metric. For example, the metric *minimize average delay* is measured by summing the delay of packets. Accordingly, the utility of a packet is its expected delay. Thus, RAPID is a heuristic based on locally optimizing marginal utility, i.e., the expected increase in utility per unit resource used. RAPID replicates packets in decreasing order of their marginal utility at each transfer opportunity.

The marginal utility heuristic has some desirable properties. The marginal utility of replicating a packet to a node is low when *(i)* the packet has many replicas, or *(ii)* the node is a poor choice with respect to the routing metric, or *(iii)* the resources used do not justify the benefit. For example, if nodes meet each other uniformly, then a packet i with 6 replicas has lower marginal utility of replication compared to a packet j with just 2 replicas. On the other hand, if the peer is unlikely to meet j's destination for a long time, then i may take priority over j.

RAPID has three core components: a *selection* algorithm, an *inference* algorithm, and a *control channel*. The selection algorithm is used to determine *which* packets to replicate at a transfer opportunity given their utilities. The inference algorithm is used to estimate the *utility* of a packet given the routing metric. The control channel propagates the necessary metadata required by the inference algorithm.

PROTOCOL RAPID(X, Y):

1. *Initialization*: Obtain metadata from Y about packets in its buffer and metadata Y collected over past meetings (detailed in Section 4.2).
2. *Direct delivery*: Deliver packets destined to Y in decreasing order of their utility.
3. *Replication*: For each packet i in node X's buffer
   (a) If i is already in Y's buffer (as determined from the metadata), ignore i.
   (b) Estimate marginal utility, δU_i, of replicating i to Y.
   (c) Replicate packets in decreasing order of $\frac{\delta U_i}{s_i}$.
4. *Termination*: End transfer when out of radio range or all packets replicated.

3.4 The selection algorithm

The RAPID protocol executes when two nodes are within radio range and have discovered one another. The protocol is symmetric; without loss of generality, and describes how node X determines which packets to transfer to node Y (refer to the box marked PROTOCOL RAPID).

RAPID also adapts to storage restrictions for in-transit data. If a node exhausts all available storage, packets with the lowest utility are deleted first as they contribute least to overall performance. However, a source never deletes its own packet unless it receives an acknowledgment for the packet.

3.5 Inference algorithm

Next, we describe how PROTOCOL RAPID can support specific metrics using an algorithm to infer utilities. Table 2 defines the relevant variables.

3.5.1 Metric 1: Minimizing average delay

To minimize the average delay of packets in the network we define the utility of a packet as

$$U_i = -D(i) \tag{1}$$

since the packet's expected delay is its contribution to the performance metric. Thus, the protocol attempts to greedily replicate the packet whose replication reduces the delay by the most among all packets in its buffer.

3.5.2 Metric 2: Minimizing missed deadlines

To minimize the number of packets that miss their deadlines, the utility is defined as the the probability that the packet will be delivered within its deadline:

$$U_i = \begin{cases} P(a(i) < L(i) - T(i)), & L(i) > T(i) \\ 0, & \text{otherwise} \end{cases} \tag{2}$$

where $L(i)$ is the packet life-time. A packet that has missed its deadline can no longer improve performance and is thus assigned a value of 0. The marginal utility is the improvement in the probability that the packet will be delivered within

ALGORITHM ESTIMATE_DELAY(X, Q, Z):
Node X with a set of packets Q to destination Z estimates the time, $A(i)$, until packet $i \in Q$ is delivered to Z as follows:

1. Sort packets in Q in decreasing order of $T(i)$. Let $b(i)$ be the sum of sizes of packets that precede i, and B the expected transfer opportunity in bytes between X and Z (refer Figure 1).
2. X by itself requires $\lceil b(i)/B \rceil$ meetings with Z to deliver i. Compute the random variable $M_X(i)$ for the corresponding delay as
$$M_X(i) = M_{XZ} + M_{XZ} + \ldots \lceil b(i)/B \rceil \text{ times} \quad (4)$$
3. Let $X_1, \ldots, X_k \supseteq X$ be the set of nodes possessing a replica of i. Estimate remaining time $a(i)$ as
$$\mathrm{a}(i) = \min(M_{X_1}(i), \ldots, M_{X_k}(i)) \quad (5)$$
4. Expected delay $D(i) = T(i) + E[a(i)]$

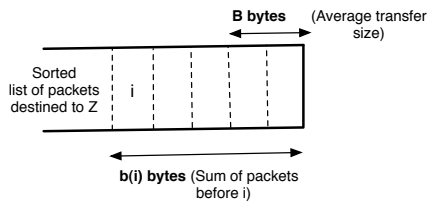

Figure 1: Position of packet i in a queue of packets destined to Z.

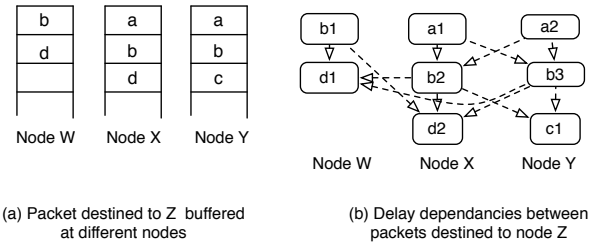

Figure 2: Delay dependencies between packets destined to Z buffered in different nodes.

its deadline, so the protocol replicates the packet that yields the highest improvement among packets in its buffer.

3.5.3 Metric 3: Minimizing maximum delay

To minimize the maximum delay of packets in the network, we define the utility U_i as

$$U_i = \begin{cases} -D(i), & D(i) \geq D(j) \quad \forall j \in S \\ 0, & \text{otherwise} \end{cases} \quad (3)$$

where S denotes the set of all packets in X's buffer. Thus, U_i is the negative expected delay if i is a packet with the maximum expected delay among all packets held by Y. So, replication is useful only for the packet whose delay is maximum. For the routing algorithm to be work conserving, RAPID computes utility for the packet whose delay is currently the maximum; i.e., once a packet with maximum delay is evaluated for replication, the utility of the remaining packets is recalculated using Eq. 3.

4. ESTIMATING DELIVERY DELAY

How does a RAPID node estimate expected delay in Eqs. 1 and 3, or the probability of packet delivery within a deadline in Eq. 2? The expected delivery delay is the minimum expected time until any node with the replica of the packet delivers the packet; so a node needs to know which other nodes possess replicas of the packet and when they expect to meet the destination.

To estimate expected delay we assume that the packet is delivered directly to the destination, ignoring the effect of further replication. This estimation is nontrivial even with an accurate global snapshot of system state. For ease of exposition, we first present RAPID's estimation algorithm as if we had knowledge of the global system state, and then we present a practical distributed implementation.

4.1 Algorithm Estimate_Delay

Algorithm ESTIMATE_DELAY works as follows. Each node X maintains a separate queue of packets Q destined to each node Z sorted in decreasing order of $T(i)$ or time since creation — the order in which they would be delivered directly (in Step 2 of PROTOCOL RAPID). Step 2 in ESTIMATE_DELAY computes the delay distribution for delivery of the packet by X, as if X were the only node carrying a replica of i. Step 3 computes the minimum across all replicas of the corresponding delay distributions, as the remaining time $a(i)$ is the time until any one of those nodes meets Z.

ESTIMATE_DELAY makes a simplifying independence assumption that does not hold in general. Consider Figure 2(a), an example showing the positions of packet replicas in the queues of different nodes; packets with the same letter and different indices are replicas. All packets have a common destination Z and each queue is sorted by $T(i)$. Assume that the size of each transfer opportunity is one packet.

Packet b may be delivered in two ways: *(i)* if W meets Z; *(ii)* one of X and Y meets Z and then one of X and Y meet Z again. These delay dependencies can be represented using a dependency graph as illustrated in Fig 2(b). A vertex corresponds to a packet replica. An edge from one node to another indicates a dependency between the delays of the corresponding packets. Recall that M_{XY} is the random variable that represents the meeting time between X and Y.

ESTIMATE_DELAY ignores all the non-vertical dependencies. For example, it estimates b's delivery time distribution as

$$\min(M_{WZ}, M_{XZ} + M_{XZ},\ M_{YZ} + M_{YZ}),$$

whereas the distribution is actually

$$\min(M_{WZ}, \min(M_{XZ}, M_{YZ}) + \min(M_{XZ}, M_{YZ})).$$

Although, in general, the independence assumption can arbitrarily inflate delay estimates[1], it makes our implementation *(i) simple* — computing an accurate estimate is much more complex especially when transfer opportunities are not unit-sized as above — and *(ii) distributed* — in practice, RAPID does not have global view, but ESTIMATE_DELAY can be implemented using a thin in-band control channel.

[1]Pathological cases are discussed in our technical report [3].

4.1.1 Exponential distributions

We walk through the distributed implementation of ESTIMATE_DELAY for a scenario where the inter-meeting time between nodes is exponentially distributed. Further, suppose all nodes meet according to a uniform exponential distribution with mean time $1/\lambda$. In the absence of bandwidth restrictions, the expected delivery delay when there are k replicas is the mean meeting time divided by k, i.e., $\mathrm{P}(a(i) < t) = 1 - e^{-k\lambda t}$ and $A(i) = \frac{1}{k\lambda}$. (Note that the minimum of k i.i.d. exponentials is also an exponential with mean $1/k$ of the mean of the i.i.d exponentials [8].)

However, when transfer opportunities are limited, the expected delay depends on the packet's position in nodes' buffers. In Step 2 of ESTIMATE_DELAY, the time for some node X to meet the destination $\lceil b(i)/B \rceil$ times is described by a gamma distribution with mean $\frac{1}{\lambda} \cdot \lceil b(i)/B \rceil$.

If packet i is replicated at k nodes, Step 3 computes the delay distribution $a(i)$ as the minimum of k gamma variables. We do not know of a closed form expression for the minimum of gamma variables. Instead, if we assume that the time taken for a node to meet the destination $b(i)/B$ times is exponential with the same mean $\frac{1}{\lambda} \cdot \lceil b(i)/B \rceil$, we can again estimate $a(i)$ as the minimum of k exponentials as follows.

Let $n_1(i), n_2(i), \ldots, n_k(i)$ be the number of times each of the k nodes respectively needs to meet the destination to deliver i directly. Then $A(i)$ is computed as:

$$\mathrm{P}(a(i) < t) = 1 - e^{-(\frac{\lambda}{n_1(i)} + \frac{\lambda}{n_2(i)} + \ldots + \frac{\lambda}{n_k(i)})t} \quad (6)$$

$$A(i) = \frac{1}{\frac{\lambda}{n_1(i)} + \frac{\lambda}{n_2(i)} + \ldots + \frac{\lambda}{n_k(i)}} \quad (7)$$

When the meeting time distributions between nodes are non-uniform, say with means $1/\lambda_1, 1/\lambda_2 \ldots 1/\lambda_k$ respectively, then $A(i) = (\frac{\lambda_1}{n_1(i)} + \frac{\lambda_2}{n_2(i)} + \ldots + \frac{\lambda_k}{n_k(i)})^{-1}$.

4.1.2 Unknown mobility distributions

To estimate mean inter-node meeting times in the DieselNet testbed, every node tabulates the average time to meet every other node based on past meeting times. Nodes exchange this table as part of metadata exchanges (Step 1 in PROTOCOL RAPID). A node combines the metadata into a meeting-time adjacency matrix and the information is updated after each transfer opportunity. The matrix contains the expected time for two nodes to meet directly, calculated as the average of past meetings.

Node X estimates $E(M_{XZ})$, the expected time to meet Z, using the meeting-time matrix. $E(M_{XZ})$ is estimated as the expected time taken for X to meet Z in at most h hops. (Unlike uniform exponential mobility models, some nodes in the trace never meet directly.) For example, if X meets Z via an intermediary Y, the expected meeting time is the expected time for X to meet Y and then Y to meet Z in 2 hops. In our implementation we restrict $h = 3$. When two nodes never meet, even via three intermediate nodes, we set the expected inter-meeting time to infinity. Several DTN routing protocols [6, 23, 7] use similar techniques to estimate meeting probability among peers.

Let replicas of packet i destined to Z reside at nodes $X_1, \ldots, X_k$. Since we do not know the meeting time distributions, we simply assume they are exponentially distributed. Then from Eq. 7, the expected delay to deliver i is

$$A(i) = [\sum_{j=1}^{k} \frac{1}{E(M_{X_j Z}) \cdot n_j(i)}]^{-1} \quad (8)$$

We use an exponential distribution because bus meeting times in the testbed are very difficult to model. Buses change routes several times in one day, the inter-bus meeting distribution is noisy, and we found them hard to model even using mixture models.Approximating meeting times as exponentially distributed makes delay estimates easy to compute and performs well in practice.

4.2 Control channel

Previous studies [19] have shown that as nodes have the benefit of more information about global system state and future from oracles, they can make significantly better routing decisions. We extend this idea to practical DTNs where no oracle is available. To this end, RAPID nodes gather knowledge about the global system state by disseminating metadata using a fraction of the transfer opportunity.

RAPID uses an in-band *control channel* to exchange acknowledgments for delivered packets as well as metadata about every packet learnt from past exchanges. For each encountered packet i, RAPID maintains a list of nodes that carry the replica of i, and for each replica, an estimated time for direct delivery. Metadata for delivered packets is deleted when an ack is received.

For efficiency, a RAPID node maintains the time of last metadata exchange with its peers. The node only sends information about packets whose information changed since the last exchange, which reduces the size of the exchange considerably. A RAPID node sends the following information on encountering a peer.

- Average size of past transfer opportunities;
- Expected meeting times with nodes;
- List of packets delivered since last exchange;
- For each of its own packets, the updated delivery delay estimate based on current buffer state;
- Information about other packets if modified since last exchange with the peer.

When using the control channel, nodes have only an imperfect view of the system. The propagated information may be stale due to change is number of replicas, changes in delivery delays, or if the packet is delivered but acknowledgments have not propagated. Nevertheless, our experiments confirm that *(i)* this inaccurate information is sufficient for RAPID to achieve significant performance gains over existing protocols and *(ii)* the overhead of metadata itself is minimal.

5. IMPLEMENTATION ON A VEHICULAR DTN TESTBED

We implemented and deployed RAPID on our vehicular DTN testbed, DieselNet [6] (`http://prisms.cs.umass.edu/dome`), consisting of 40 buses, of which a subset is on the road each day. The implementation allowed us to meet the following two objectives. First, the routing protocol is a first step towards deploying realistic DTN applications on the testbed. Second, the deployment is subject to some events that are not perfectly modeled in the simulation, including delays caused by computation or the wireless channel.

Each bus in DieselNet carries a small-form desktop computer, 40 GB of storage, and a GPS device. The buses operate a 802.11b radio that scans for other buses 100 times a second and an 802.11b access point (AP) that accepts incoming connections. Once a bus is found, a connection is created to the remote AP. (It is likely that the remote bus then creates a connection to the discovered AP, which our software merges into one connection event.) The connection lasts until the radios are out of range. Burgess et al. [6] describes the DieselNet testbed in more detail.

5.1 Deployment

Buses in DieselNet send messages using PROTOCOL RAPID in Section 3, computing the metadata as described in Section 4.2. We generated packets of size 1 KB periodically on each bus with an exponential inter-arrival time. The destinations of the packets included only buses that were scheduled to be on the road, which avoided creation of many packets that could never be delivered. We did not provide the buses information about the location or route of other buses on the road. We set the default packet generation rate to 4 packets per hour generated by each bus for every other bus on the road; since the number of buses on the road at any time varies, this is the simplest way to express load. For example, when 20 buses are on the road, the default rate is 1,520 packets per hour.

During the experiments, the buses logged packet generation, packet delivery, delivery delay, meta-data size, and the total size of the transfer opportunity. Buses transfered random data after all routing was complete in order to measure the capacity and duration of each transfer opportunity. The logs were periodically uploaded to a central server using open Internet APs found on the road.

5.2 Performance of deployed RAPID

We measured the routing performance of RAPID on the buses from Feb 6, 2007 until May 14, 2007[2]. The measurements are tabulated in Table 3. We exclude holidays and weekends since almost no buses were on the road, leaving 58 days of experiments. RAPID delivered 88% of packets with an average delivery delay of about 91 minutes. We also note that overhead due to meta-data accounts for less than 0.02% of the total available bandwidth and less than 1.7% of the data transmitted.

5.3 Validating trace-driven simulator

In the next section, we evaluate RAPID using a trace-driven simulator. The simulator takes as input a schedule of node meetings, the bandwidth available at each meeting, and a routing algorithm. We validated our simulator by comparing simulation results against the 58-days of measurements from the deployment. In the simulator, we generate packets under the same assumptions as the deployment, using the same parameters for exponentially distributed inter-arrival times.

Figure 3 shows the average delay characteristics of the real system and the simulator. Delays measured using the simulator were averaged over the 30 runs and the error-bars show a 95% confidence interval. From those results and further analysis, we find with 95% confidence that the simulator results are within 1% of the implementation measurement of average delay. The close correlation between system measurement and simulation increases our confidence in the accuracy of the simulator.

[2]The traces are available at `http://traces.cs.umass.edu`.

Avg. buses scheduled per day	19
Avg. total bytes transfered per day	261.4 MB
Avg. number of meetings per day	147.5
Percentage delivered per day	88%
Avg. packet delivery delay	91.7 min
Meta-data size/ bandwidth	0.002
Meta-data size/ data size	0.017

Table 3: Deployment of Rapid: Average daily statistics

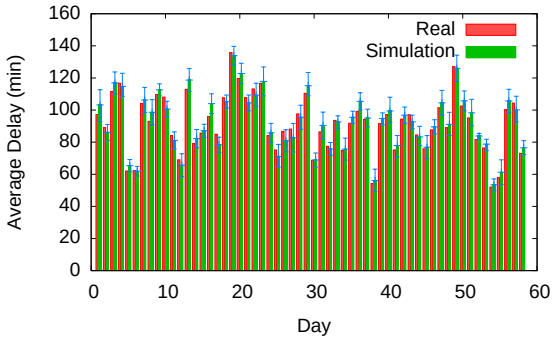

Figure 3: Trace: Average delay for 58 days of rapid real deployment compared to simulation of rapid using traces

6. EVALUATION

The goal of our evaluations is to show that, unlike existing work, RAPID can improve performance for customizable metrics. We evaluate RAPID using three metrics: minimize maximum delay, minimize average delay, and minimize missed deadlines. In all cases, we found that RAPID significantly outperforms existing protocols and also performs close to optimal for our workloads.

6.1 Experimental setup

Our evaluations are based on a custom event-driven simulator, as described in the previous section. The meeting times between buses in these experiments are not known *a priori*. All values used by RAPID, including average meeting times, are learned during the experiment.

We compare RAPID to five other routing protocols: *MaxProp* [6], *Spray and Wait* [31], *Prophet* [23], *Random*, and *Optimal*. In all experiments, we include the cost of RAPID's in-band control channel for exchanging metadata.

MaxProp operates in a storage- and bandwidth-constrained environment, allows packet replication, and leverages delivery notifications to purge old replicas; of recent related work, it is closest to RAPID's objectives. *Random* replicates randomly chosen packets for the duration of the transfer opportunity. *Spray and Wait* restricts the number of replications of a packets to L, where L is calculated based on the number of nodes in the network. For our simulations, we implemented the binary *Spray and Wait* and set[3] $L = 12$. We implemented

[3]We set this value based on consultation with authors and

	Power law	Trace-driven
Number of nodes	20	max of 40
Buffer size	100 KB	40 GB
Average transfer opp. size	100 KB	given by real transfers among buses
Duration	15 min	19 hours each trace
Size of a packet	1 KB	1 KB
Packet generation rate	50 sec mean	1 hour
Delivery deadline	20 sec	2.7 hours

Table 4: Experiment parameters

Prophet with parameters $P_{init} = 0.75$, $\beta = 0.25$ and $\gamma = 0.98$ (parameters based on values used in [23]).

We also compare RAPID to *Optimal*, the optimal routing protocol that provides an upper bound on performance. We also perform experiments where mobility is modeled as a power law distribution. Previous studies [9, 22] have suggested that DTNs among people have a skewed, power law inter-meeting time distribution. The default parameters used for all the experiments are tabulated in Table 4. The parameters for power law mobility model is different from the trace-driven model because the performance between the two models are not comparable.

Each data point is averaged over 10 runs; in the case of trace-driven results, the results are averaged over 58 traces. Each of the 58 days is a separate experiment. In other words, packets that are not delivered by the end of the day are lost. In all experiments, *MaxProp*, RAPID and *Spray and Wait* performed significantly better than *Prophet*, and the latter is not shown in the graphs for clarity. In all trace experiments, *Prophet* performed worse than the three routing protocols for for all loads and all metrics.

6.2 Results based on testbed traces

6.2.1 Comparison with existing routing protocols

Our experiments show that RAPID consistently outperforms *MaxProp*, *Spray and Wait* and *Random*. We increased the load in the system up to 40 packets per hour per destination, when *Random* delivers less than 50% of the packets.

Figure 4 shows the average delay of delivered packets using the four protocols for varying loads when RAPID's routing metric is set to minimize average delay (Eq. 1). When using RAPID, the average delay of delivered packets is significantly lower than *MaxProp*, *Spray and Wait* and *Random*. Moreover, RAPID also consistently delivers a greater fraction of packets as shown in Figure 5.

Figure 6 shows RAPID's performance when the routing metric is set to minimize maximum delay (Eq. 3) and similarly Figure 7 shows results when the metric is set to maximize the number of packets delivered within a deadline (Eq. 2).

We note that among *MaxProp*, *Spray and Wait* and *Random*, *MaxProp* delivers the most number of packets, but *Spray and Wait* has marginally lower average delay than *MaxProp*. RAPID significantly outperforms the three protocol for all metrics because of its intentional design.

Standard deviation and similar measures of variance are not appropriate for comparing the mean delays as each bus takes a different geographic route. So, we performed a paired t-test [8] to compare the average delay of every source-destination pair using RAPID to the average delay of the same source-destination pair using *MaxProp* (the second best performing protocol). In our tests, we found p-values always less than 0.0005, indicating the differences between the means reported in these figures are statistically significant.

6.2.2 Metadata exchange

We allow RAPID to use as much bandwidth at the start of a transfer opportunity for exchanging metadata as it requires. To see if this approach was wasteful or beneficial, we performed experiments where we limited the total metadata exchanged. Figure 8 shows the average delay performance of RAPID when metadata is limited as a percentage of the total bandwidth. The results show that performance increases as the limit is removed and that the best performance results when there is no restriction on metadata at all. The performance of RAPID with complete metadata exchange improves by 20% compared to when no metadata is exchanged. The metadata in this experiment is represented as a percentage of available bandwidth.

In the next experiment, we analyze total metadata as a percentage of data. In particular, we increase the load to 75 packets per destination per hour to analyze the trend in terms of bandwidth utilization, delivery rate and metadata. Figure 9 shows this trend as load increases. The bandwidth utilization is about 35% for the load of 75 packets per hour per destination, while delivery rate is only about 65%. This suggests that the performance drops even though the network is under-utilized, and it is because of the bottleneck links in the network. The available bandwidth varies significantly across transfer opportunities in our bus traces [6].

We also observe that metadata increases to about 4% of data for high loads. This is an order of magnitude higher than the metadata observed as a fraction of bandwidth, again because of the poor channel utilization. The average metadata exchange per contact is proportional to the load and the channel utilization. However, metadata enables efficient routing and helps remove copies of packets that are already delivered, increasing the overall performance of RAPID. Moving from 1-KB to 10-KB packets will reduce RAPID's metadata overhead by another order of magnitude.

6.2.3 Hybrid DTN with thin continuous connectivity

In this section, we compare the performance of RAPID using an instant *global* control channel for exchanging metadata as opposed to the default (delayed) *in-band* control channel.

Figure 10 shows the average delay of RAPID when using an in-band control channel compared to a global channel. We observe that the average delay decreases by up to 20 minutes when using a global channel. Similarly, from Figure 11 we observe that the percentage packets delivered within a deadline increases by an average of 20% using a global channel. This observation suggests that RAPID's performance can benefit further by using more control information.

One interpretation of the global channel is the use of RAPID as a hybrid DTN where all control traffic goes over a low-bandwidth, long-range radio such as XTEND [4]. A hybrid DTN will use a high-cost, low-bandwidth channel for control whenever available and low-cost high-bandwidth delayed channel for data. In our experiments, we assumed that the global channel is instant. While this may not be feasible in practice, the results give an upper bound on RAPID's performance when accurate channel information is available.

using LEMMA 4.3 in [31] with $a = 4$.

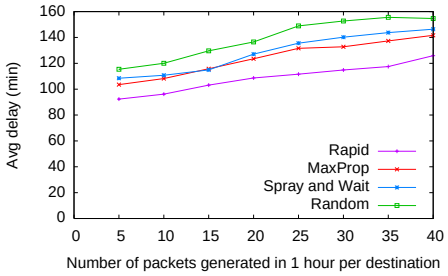

Figure 4: (Trace) Average Delay: RAPID has up to 20% lower delay than *MaxProp* and up to 35% lower delay than *Random*

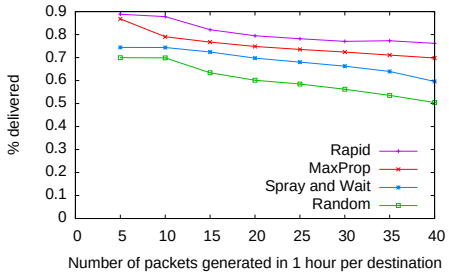

Figure 5: (Trace) Delivery Rate: RAPID delivers up to 14% more than *MaxProp*, 28% than *Spray and Wait* and 45% than *Random*

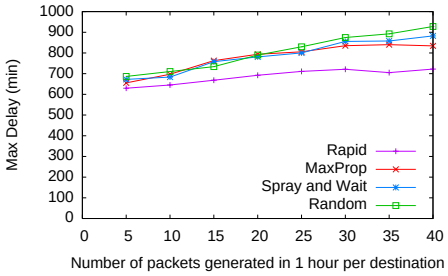

Figure 6: (Trace) Max Delay: Maximum delay of RAPID is up to 90 min lower than *MaxProp*, *Spray and Wait*, and *Random*

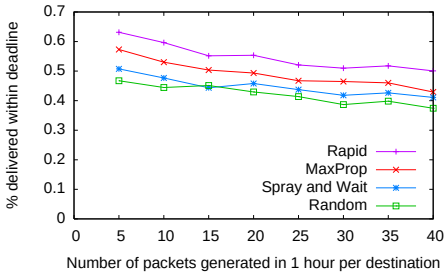

Figure 7: (Trace) Delivery within deadline: RAPID delivers up to 21% more than *MaxProp*, 24% than *Spray and Wait*, 28% than *Random*

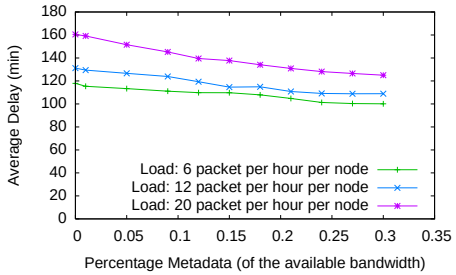

Figure 8: (Trace) Control channel benefit: Average delay performance improves as more metadata is allowed to be exchanged

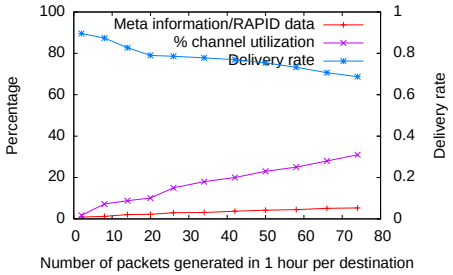

Figure 9: (Trace) Channel utilization: As load increases, delivery rate decreases to 65% but channel utilization is only about 35%

6.2.4 *Comparison with Optimal*

We compare RAPID to *Optimal*, which is an upper bound on the performance. To obtain the optimal delay, we formulate the DTN routing problem as an Integer Linear Program (ILP) optimization problem when the meeting times between nodes are precisely know. The optimal solution assumes that the propagation delay of all links are equal and that node meetings are known in advance. We present a formulation of this problem in our technical report [3]. Our evaluations use the CPLEX solver [11]. Because the solver grows in complexity with the number of packets, these simulations are limited to only 6 packets per hour per destination. Jain et al. [19] solve a more general DTN routing problem by allowing packets to be fragmented across links and assigning non-zero propagation delays on the links, however, this limited the size of the network they could evaluate even more. Our ILP objective function minimizes delay of all packets, where the delay of undelivered packets is set to time the packet spent in the system. Accordingly, we add the delay of undelivered packets when presenting the results for RAPID and *MaxProp*.

Figure 12 presents the average delay performance of *Optimal*, RAPID, and *MaxProp*. We observe that for small loads, the performance of RAPID using the in-band control channel is within 10% of the optimum performance, while using *MaxProp* the delays are about 22% from the optimal. RAPID using a global channel performs within 6% of optimal.

6.2.5 *Evaluation of* RAPID *components*

RAPID is comprised of several components that all contribute to performance. We ran experiments to study the value added by each component. Our approach is to compare subsets of the full RAPID, cumulatively adding components from *Random*. The components are *(i) Random with acks*: propagation of delivery acknowledgments; and *(ii)* RAPID-LOCAL: using RAPID but nodes exchange metadata about only packets in their own buffers.

Figure 13 shows the performance of different components of RAPID when the routing metric is set to minimize average delay. From the figure we observe that using acknowledgments alone improves performance by an average of 8%. In our previous work, *MaxProp* [6], we show empirically that propagating acknowledgments clears buffers, avoids exchange of already delivered packets and improving performance. In addition, RAPID-LOCAL provides a further improvement of 10% on average even though metadata exchange is restricted to packets in the node's local buffer. Allowing all metadata to flow further improves the performance by about 11%.

6.3 Results from synthetic mobility models

Next, we use a power law mobility model to compare the performance of RAPID to *MaxProp*, *Random*, and *Spray and Wait*. When mobility is modeled using power law, two nodes meet with an exponential inter-meeting time, but the mean of the exponential distribution is determined by the popularity of the nodes. For the 20 nodes, we randomly set a popularity value of 1 to 20, with 1 being most popular. The mean of the power law mobility model is set to 0.3 seconds and is skewed for each pair of nodes according to their popularity.

Figure 14 shows the maximum delay of packets when the load is varied (i.e., RAPID is set to use Eq. 3 as a metric).

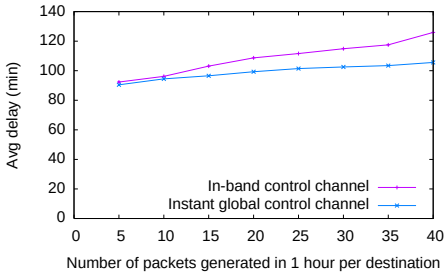

Figure 10: (Trace) Global channel: Average delay of RAPID decreases by up to 20 minutes using instant global control channel

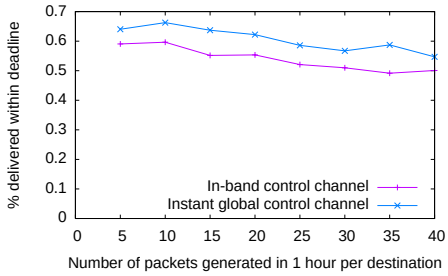

Figure 11: (Trace) Global channel: Packets delivered within deadline increases by about 15% using instant global control channel

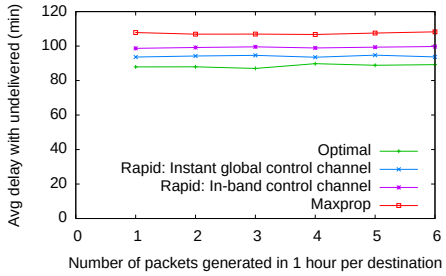

Figure 12: (Trace) Comparison with *Optimal*: Average delay of RAPID is within 10% of *Optimal* for small loads

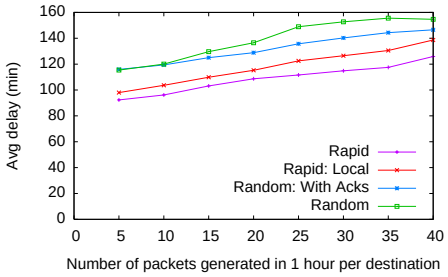

Figure 13: (Trace) RAPID Components: Flooding acks decreases average delay by about 8% and RAPID further decreases average delay by about 30% over *Random*

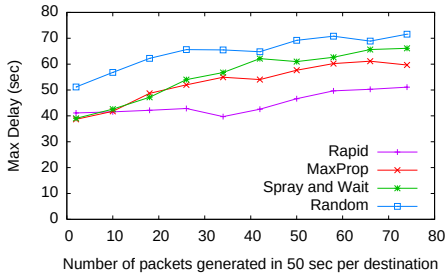

Figure 14: (Powerlaw) Max delay: RAPID's max delay is about 30% lower than *MaxProp*, 35% lower than *Spray and Wait* and 45% lower than *Random*

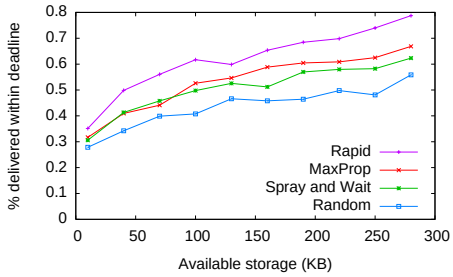

Figure 15: (Powerlaw) Constrained buffer size: RAPID delivers about 20% more packets than *MaxProp*, and 45% more packets than *Spray and Wait* and *Random*

RAPID reduces maximum delay by over 30% compared to the other protocols. For both the traces and the synthetic mobility, the performance of RAPID is significantly higher than *MaxProp*, *Spray and Wait*, and *Random* for the maximum delay metric. The reason is *MaxProp* prioritizes new packets; older, undelivered packets will not see service as load increases. Similarly, *Spray and Wait* does not give preference to older packets. However, RAPID specifically prioritizes older packets to reduce maximum delay.

Figure 15 shows how constrained buffers varied from 10 KB to 280 KB affect the delivery deadline metric for a fixed load of 20 packets per node per destination every 50 seconds. RAPID is able to best manage limited buffers to deliver packets within a deadline. When storage is restricted, *MaxProp* deletes packets that are replicated most number of times, while *Spray and Wait* and *Random* deletes packets randomly. RAPID, when set to maximizing number of packets delivered within a deadline, deletes packets that are most likely to miss the deadline and is able to improve performance significantly.

We observed similar trends for other routing metrics for increasing load and decreasing buffer size and when the node meeting time was modeled as an exponential distribution. Those results are presented in our technical report [3].

6.4 Limitations

The above experiments show that RAPID performs well from many viewpoints. However, there are limitations to our approach. The heuristics we use are sub-optimal solutions and although they seek to maximize specific utilities, we can offer no performance guarantees. Our estimations of delay are based on simple, tractable distributions. Finally, we note that our implementation of RAPID shows that the protocol can be deployed efficiently and effectively; however, in other DTN scenarios or testbeds, mobility patterns may be more difficult to learn. In future work, we believe a more sophisticated estimation of delay will improve our results, perhaps bringing us closer to guarantees of performance. The release of an implementation of RAPID will enable us to enlist others to deploy RAPID on their DTNs, diversifying results to other scenarios.

7. CONCLUSIONS

Previous work in DTN routing protocols has seen only incidental performance improvement from various routing mechanisms and protocol design choices. In contrast, we have proposed a routing protocol for DTNs that intentionally maximizes the performance of a specific routing metric. Our protocol, RAPID, treats DTN routing as a resource allocation problem, making use of an in-band control channel to propagated metadata. Although our approach is heuristic, we have proven that the general DTN routing protocol lacks sufficient information in practice to solve optimally. Moreover, we have shown that even when complete knowledge is available, solving the DTN routing problem optimally is NP-hard. Our deployment of RAPID in a DTN testbed illustrates that our approach is realistic and effective. We have shown through trace-driven simulation using 58 days of testbed measurements that RAPID yields significant performance gains over many existing protocols.

Acknowledgments

We thank Mark Corner, John Burgess, and Brian Lynn for helping build and maintain DieselNet, Ramgopal Mettu for helping develop the NP-hardness proof, and Erik Learned-Miller and Jérémie Leguay for feedback on earlier drafts.

8. REFERENCES

[1] One laptop per child. `http://www.laptop.org`.
[2] TIER Project, UC Berkeley. `http://tier.cs.berkeley.edu/`.
[3] A. Balasubramanian, B. N. Levine, and A. Venkataramani. DTN Routing as a Resource Allocation Problem. Technical Report 07-37, UMass Amherst, 2007.
[4] N. Banerjee, M. D. Corner, and B. N. Levine. An Energy-Efficient Architecture for DTN Throwboxes. In *Proc. IEEE Infocom*, May 2007.
[5] J. Burgess, G. Bissias, M. D. Corner, and B. N. Levine. Surviving Attacks on Disruption-Tolerant Networks without Authentication. In *Proc. ACM Mobihoc*, September 2007.
[6] J. Burgess, B. Gallagher, D. Jensen, and B. N. Levine. MaxProp: Routing for Vehicle-Based Disruption- Tolerant Networks. In *Proc. IEEE Infocom*, April 2006.
[7] B. Burns, O. Brock, and B. N. Levine. *MV* Routing and Capacity Building in Disruption Tolerant Networks. In *Proc. IEEE Infocom*, pages 398–408, March 2005.
[8] G. Casella and R. L. Berger. *Statistical Inference*. Second Edition. Duxbury, 2002.
[9] A. Chaintreau, P. Hui, J. Crowcroft, C. Diot, R. Gass, and J. Scott. Impact of Human Mobility on the Design of Opportunistic Forwarding Algorithms. In *Proc. IEEE Infocom*, May 2006.
[10] C. Chekuri, S. Khanna, and F. B. Shepherd. An $O(\sqrt{(}n))$ Approximation and Integrality Gap for Disjoint Paths and Unsplittable Flow. *Theory of Computing*, 2(7):137–146, 2006.
[11] CPLEX. `http://www.ilog.com`.
[12] J. Davis, A. Fagg, and B. N. Levine. Wearable Computers and Packet Transport Mechanisms in Highly Partitioned Ad hoc Networks. In *Proc. IEEE ISWC*, pages 141–148, October 2001.
[13] P. Desnoyers, D. Ganesan, H. Li, M. Li, and P. Shenoy. PRESTO: A Predictive Storage Architecture for Sensor Networks. In *Proc. USENIX HotOS*, June 2005.
[14] R. Gallager. A Minimum Delay Routing Algorithm Using Distributed Computation. In *IEEE Trans. on Communications*, volume 25, pages 73–85, Jan 1977.
[15] N. Garg, S. Sobti, J. Lai, F. Zheng, K. Li, A. Krishnamurthy, and R. Wang. Bridging the Digital Divide. *ACM Trans. on Storage*, 1(2):246–275, May 2005.
[16] V. Guruswami, S. Khanna, R. Rajaraman, B. Shepherd, and M. Yannakakis. Near-Optimal Hardness Results and Approximation Algorithms for Edge-Disjoint Paths and Related Problems. In *Proc. ACM STOC*, pages 19–28, 1999.
[17] B. Hull et al. CarTel: A Distributed Mobile Sensor Computing System. In *Proc. ACM SenSys*, pages 125–138, Oct. 2006.
[18] S. Jain, M. Demmer, R. Patra, and K. Fall. Using Redundancy to Cope with Failures in a Delay Tolerant Network. In *Proc. ACM Sigcomm*, pages 109–120, 2005.
[19] S. Jain, K. Fall, and R. Patra. Routing in a Delay Tolerant Network. In *Proc. ACM Sigcomm*, pages 145–158, Aug. 2004.
[20] E. Jones, L. Li, and P. Ward. Practical Routing in Delay-Tolerant Networks. In *Proc. ACM Chants Workshop*, pages 237–243, Aug. 2005.
[21] F. Kelly, A. Maulloo, and D. Tan. Rate Control for Communication Networks: Shadow Prices, Proportional Fairness and Stability. In *J. Op. Res. Society*, volume 49, pages 237–252, 1998.
[22] J. Leguay, T. Friedman, and V. Conan. DTN Routing in a Mobility Pattern Space. In *Proc. ACM Chants Workshop*, pages 276–283, Aug. 2005.
[23] A. Lindgren, A. Doria, and O. Schelén. Probabilistic Routing in Intermittently Connected Networks. In *Proc. SAPIR Workshop*, pages 239–254, Aug. 2004.
[24] A. Maffei, K. Fall, and D. Chayes. Ocean Instrument Internet. In *Proc. AGU Ocean Sciences Conf.*, Feb 2006.
[25] W. Mitchener and A. Vadhat. Epidemic Routing for Partially Connected Ad hoc Networks. Technical Report CS-2000-06, Duke Univ., 2000.
[26] J. Ott and D. Kutscher. A Disconnection-Tolerant Transport for Drive-thru Internet Environments. In *Proc. IEEE INFOCOM*, pages 1849–1862, Mar. 2005.
[27] C. Papadimitriou. *Computational Complexity*. Addison Wesley, 1994.
[28] J. Partan, J. Kurose, and B. N. Levine. A Survey of Practical Issues in Underwater Networks. In *Proc. ACM WUWNet*, pages 17–24, Sept. 2006.
[29] R. C. Shah, S. Roy, S. Jain, and W. Brunette. Data MULEs: Modeling a Three-tier Architecture for Sparse Sensor Networks. In *Proc. IEEE SNPA*, pages 30–41, May 2003.
[30] T. Small and Z. Haas. Resource and Performance Tradeoffs in Delay-Tolerant Wireless Networks. In *Proc. ACM WDTN*, pages 260–267, Aug. 2005.
[31] T. Spyropoulos, K. Psounis, and C. S. Raghavendra. Spray and Wait: An Efficient Routing Scheme for Intermittently Connected Mobile Networks. In *Proc. ACM WDTN*, pages 252–259, Aug. 2005.
[32] T. Spyropoulos, K. Psounis, and C. S. Raghavendra. Performance analysis of mobility-assisted routing. In *ACM MobiHoc*, pages 49–60, May 2006.
[33] T. Spyropoulos and K. Psounis and C. Raghavendra. Single-copy Routing in Intermittently Connected Mobile Networks. In *IEEE SECON*, October 2004.
[34] J. Widmer and J.-Y. Le Boudec. Network Coding for Efficient Communication in Extreme Networks. In *Proc. ACM WDTN*, pages 284–291, Aug. 2005.
[35] Y.-C. Tseng and S.-Y. Ni and Y.-S. Chen and J.-P. Sheu. The Broadcast Storm Problem in a Mobile Ad hoc Network. *Springer Wireless Networks*, 8(2/3):153–167, 2002.
[36] P. Zhang, C. M. Sadler, S. A. Lyon, and M. Martonosi. Hardware Design Experiences in ZebraNet. In *Proc. ACM SenSys*, pages 227–238, Nov. 2004.
[37] X. Zhang, G. Neglia, J. Kurose, and D. Towsley. Performance Modeling of Epidemic Routing. In *Proc. IFIP Networking*, May 2006.

APPENDIX

Packets and transfer opportunities are unit-sized and storage is unlimited in all proofs outlined below. Formal proofs are presented in our technical report [3].

THEOREM 1(a). *If ALG has complete knowledge of a workload consisting of n packets, but not of the schedule of node meetings, then it is $\Omega(n)$-competitive with an offline adversary with respect to the fraction of packets delivered.*

PROOF OUTLINE. We construct an offline adversary, ADV, that incrementally generates a node meeting schedule based on the actions of the online algorithm ALG at each step. We show how ADV can construct a node meeting schedule such that ADV can deliver all packets while ALG, without prior knowledge of the schedule, can deliver at most 1 packet. Consider the schedule illustrated in Figure 16 where arrows represent meetings between the corresponding nodes. There are a total of $2n$ meetings — the source S at time T_1 meets n intermediate nodes $u_1, \ldots, u_n$ that each subsequently meet a unique destination from among $v_1, \ldots, v_n$ at time T_2. S

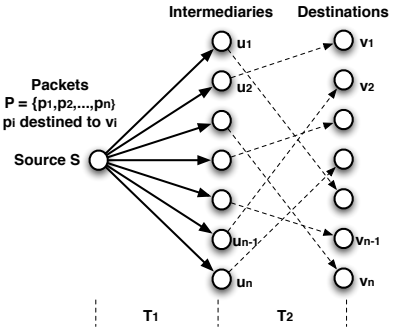

Figure 16: Theorem 1(a) construction: Solid arrows represent node meetings ALG known a priori while dotted arrows represent meetings ADV generates and reveals subsequently.

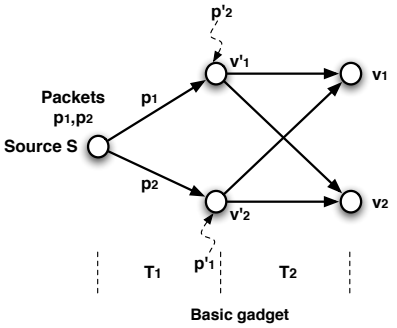

Figure 17: Theorem 1(b) basic gadget: Solid arrows represent node meetings ALG knows a priori while dotted squiggly arrows represent packets ADV generates and reveals subsequently.

must deliver n packets $p_1, \ldots, p_n$ destined respectively to $v_1, \ldots, v_n$.

ALG is effectively forced to guess the permutation corresponding to the latter set of meetings between $u_1, \ldots, u_n$ and $v_1, \ldots, v_n$. The best strategy for ALG is to replicate one packet to all nodes in $u_1, \ldots, u_n$, and this strategy allows it to deliver exactly one packet. ADV on the other hand knows the latter n meetings a priori and can therefore route all n packets to their respective destinations. □

THEOREM 1(b). *If ALG has complete knowledge of the meeting schedule, but not of the packet workload, then it can deliver at most a third of packets compared to an optimal offline adversary.*

PROOF OUTLINE. We construct an offline adversary, ADV, that incrementally generates a packet workload by observing ALG's transfers at each step. Consider the scenario in Figure 17, which we refer to as the *basic gadget*. The source S initially has two packets p_1, p_2 that it must deliver respectively to destinations v_1 and v_2 via the intermediate nodes v_1' and v_2'. Each solid arrow represents a unit-sized transfer opportunity between the corresponding nodes.

ADV can use this gadget to force ALG to drop two packets by creating two additional packets p_1' and p_2' while ADV itself delivers all four packets as follows. If ALG transfers p_1 to v_1' and p_2 to v_2' at time T_1, then ADV generates two more packets at time T_2: p_2' at v_1' destined to v_2 and p_1' at v_2' destined to v_1. ALG is forced to drop one of the two packets at both v_1' and v_2'. Instead, if ALG chooses to transfer p_1 to v_2' and p_2 to v_1', ADV simply chooses the opposite strategy putting ALG in the same predicament.

The basic gadget can be extended by composing two additional basic gadgets to force ALG to deliver at most $\frac{2}{5}$ of the packets. We show in the full proof that by constructing a gadget of depth i, ADV can force ALG to deliver at most a $i/(3i-1)$ fraction of packets. □

It is an open question whether there exists a constant-competitive online algorithm that knows the complete node meeting schedule but not the packet workload. The proofs above suggest, but do not prove, that not knowing the schedule is more damning than not knowing the workload.

THEOREM 2. *Given complete knowledge of node meetings and the packet workload* a priori*, computing a routing schedule that is optimal with respect to the number of packets delivered is NP-hard with an $\Omega(\sqrt(n))$ lower bound on approximability.*

PROOF OUTLINE. We show that the DTN routing problem given complete knowledge (of both the node meeting schedule and packet workload) is NP-hard by reducing the edge-disjoint paths (EDP) problem to the DTN routing problem.

The EDP problem for a DAG is known to be NP-hard [10]. The EDP problem is: given a DAG $G = (V, E)$ and source-destination pairs $\{(r_1, d_1)...(r_s, d_s)\}$, compute the largest subset of the source-destination pairs with edge-disjoint paths between them. The decision version of the EDP problem is: is there a subset of k source-destination pairs with edge-disjoint paths between them? We reduce this problem to the decision version of the DTN routing problem.

To reduce the EDP problem, $G = (V, E)$, to the DTN routing problem (refer Section 3.1), we topologically sort the DAG G in polynomial time and label the edges of the DAG such that the labels strictly increase along the edges. The label function l maps every edge to a natural number. We map the vertices V to the nodes in the DTN network. We map each edge $e = (u, v)$ to the transfer opportunity $(u, v, 1, l(e))$. I.e., nodes u and v have a unit-sized transfer opportunity at time $l(e)$. We map the source destination pairs (r_i, d_i) to a packet p_i from source r_i to destination d_i, with unit size and created at time 0. Since the transfer opportunities are unit-sized, at most one packet can be transfered using each opportunity. The solution to the DTN routing problem transfers p_i from u_i to v_i using a series of opportunities; the path formed by the transfers is a valid edge-disjoint path between the corresponding source-destination pairs r_i and d_i in the DAG G of the EDP problem. Using this mapping, we reduce EDP to the DTN routing problem.

The reduction above is a true reduction in the following sense: each successfully delivered DTN packet corresponds to an edge-disjoint path and vice-versa. Thus, the optimal solution for one exactly corresponds to an optimal solution for the other. We can show formally that this reduction is an L-reduction [27]. Consequently, the lower bound $\Omega(n^{1/2-\epsilon})$ known for the hardness of approximating the EDP problem [16] holds for the DTN routing problem as well. □