2007

DTN Routing as a Resource Allocation Problem

Aruna Balasubramanian
University of Massachusetts - Amherst

Brian Neil Levine
University of Massachusetts - Amherst

Arun Venkataramani
University of Massachusetts - Amherst

DTN Routing as a Resource Allocation Problem

Aruna Balasubramanian Brian Neil Levine Arun Venkataramani
Department of Computer Science, University of Massachusetts, Amherst, USA 01003
{arunab, brian, arun}@cs.umass.edu

revised: June 23, 2007

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.

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 [35], ocean sensor networks [27, 23], vehicular networks [25, 5], and projects such as TIER [2], Digital Study Hall [14], 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. [18] suggest that DTN routing protocols that use more knowledge of network conditions perform better, and Burgess et al. [5] 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 [5], 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 [35]. 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 [22, 30, 5] 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 high packet loads. 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 [11, 6, 5, 22]; *(ii)* removing useless packets using acknowledgments of delivered data [5]; *(iii)* using probabilistic mobility information to infer delivery [29]; *(iv)* replicating packets with a small probability [34]; *(v)* using network coding [33] and coding with redundancy [17]; and *(vi)* bounding the number of replicas of a packet [30, 29, 24].

In contrast, *forwarding* routing protocols maintain at most one copy of a packet in the network [18, 19, 32]. Jain et al. [18] 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. [19] 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. [28] and Spyropoulos et al. [32] 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 [30]. The consensus appears to be [30] that replicating packets can improve performance (and security [4]) 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 [30, 29, 24, 5] route packets using the number of replicas as the heuristic,

Problem	Storage	Bandwidth	Routing	Previous work (and mobility)
P1	Unlimited	Unlimited	Replication	Epidemic [24], Spray and Wait [30]: Constraint in the form of channel contention (Grid-based synthetic)
P2	Unlimited	Unlimited	Forwarding	Modified Djikstra's algorithm Jain et al. [18] (simple graph), MobySpace [21] (Powerlaw)
P3	Finite	Unlimited	Replication	Davis et al. [11] (Simple partitioning synthetic), SWIM [29] (Exponential), MV [6] (Community-based synthetic), Prophet [22] (Community-based synthetic)
P4	Finite	Finite	Forwarding	Jones et al. [19] (AP traces), Jain et al. [18] (Synthetic DTN topology)
P5	Finite	Finite	Replication	This paper (Vehicular DTN traces, exponential, and powerlaw meeting probabilities, testbed deployment), MaxProp [5] (Vehicular DTN traces)

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

but the effect of replication varies with different routing metrics. *Spray and Wait* [30] 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 mobility 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 [22, 11, 6, 29] 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 [12]. 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 [16, 5].

We were unable to find other protocols in P5 except *MaxProp* [5] 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 [36, 31, 29] 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.[18] 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.*

The formal proofs are presented in the Appendix. 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 [13] and congestion control [20] 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.

$D(i)$	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.

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.

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

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.

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 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

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

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

Algorithm 2. Node n_j storing a set of packets S to destination n_x performs the following steps to estimate the time until packet $i \in S$ is delivered

1. n_j sorts all packets $s \in S$ in the descending order of $m_{n_j}(s) + T(s)$.
2. Let $b_j(i)$ be the sum sizes of packets that precede i in the sorted list of n_j. Figure 1 illustrates a sorted buffer containing packet i.
3. Let B_j be the expected transfer opportunity in bytes between n_j and i's destination. (For readability, we drop subscript i.) Nodes locally compute the expected transfer opportunity with every other node as a moving average of past transfers.
4. *Assumption 1: Suppose only n_j could deliver the packet directly to the destination n_x.*

   Then, n_j requires $\lceil b_j(i)/B_j \rceil$ meetings with that node. Let r be a distribution that models the inter-meeting times between nodes, and let r_{jx} be the random variable that represents the time taken for n_j and n_x to meet. We transform r_{jx} to random variable r'_{jx} that represents the time until n_j and n_x meet $\lceil b_j(i)/B_j \rceil$ times. Then, by definition
   $$a_{n_j}(i) = r'_{jx} \quad (4)$$
5. *Assumption 2: Suppose the k random variables $a_{n_y}, y \in [1,k]$ were independent.*

   Then, the probability of delivering i within time t, i.e., the minimum of the k random variables $a_{n_y}, y \in [1,k]$ is:
   $$\mathrm{P}(a(i) < t) = 1 - \prod_{y=1}^{k} (1 - \mathrm{P}(a_{n_y}(i) < t) \quad (5)$$
6. Accordingly:
   $$A(i) = \mathrm{E}[a(i)] \quad (6)$$

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

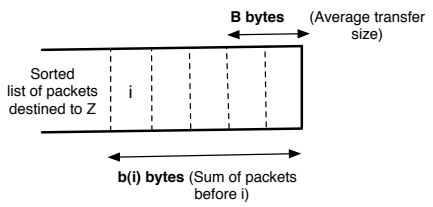

Figure 1: Position of packet i in a queue of packets destined to Z.

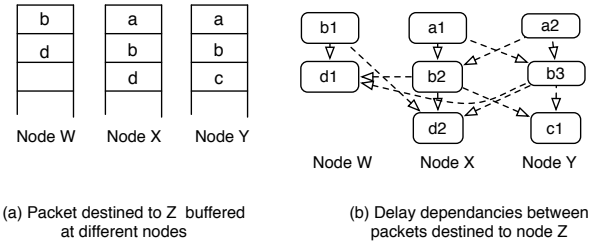

Figure 2: Delay dependencies between packets destined to Z buffered in different nodes.

distribution as

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

whereas the distribution is actually

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

In the Appendix we detail an algorithm to estimate delay without ignoring non-vertical edges. Although, in general, the independence assumption can arbitrarily inflate delay estimates (detailed in the Appendix), 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.

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 [7].)

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:

$$\mathrm{P}(a(i) < t) = 1 - e^{-(\frac{\lambda}{n_1(i)} + \frac{\lambda}{n_2(i)} + \ldots + \frac{\lambda}{n_k(i)})t} \quad (7)$$

$$A(i) = \frac{1}{\frac{\lambda}{n_1(i)} + \frac{\lambda}{n_2(i)} + \ldots + \frac{\lambda}{n_k(i)}} \quad (8)$$

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 [5, 22, 6] 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. 8, the expected delay to deliver i is

$$A(i) = [\sum_{j=1}^{k} \frac{1}{E(M_{X_j Z}) \cdot n_j(i)}]^{-1} \quad (9)$$

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 [18] 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 [5] (`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. [5] 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[1]. 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

[1]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

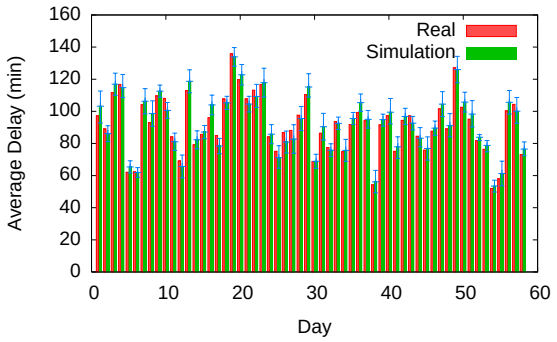

Figure 3: **Trace:** Average delay for 58 days of RAPID real deployment compared to simulation of RAPID using traces

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.

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

	Exponential/ 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

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* [5], *Spray and Wait* [30], *Prophet* [22], *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[2] $L = 12$. We implemented *Prophet* with parameters $P_{init} = 0.75$, $\beta = 0.25$ and $\gamma = 0.98$ (parameters based on values used in [22]).

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 [8, 21] 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.

[2]We set this value based on consultation with authors and using LEMMA 4.3 in [30] with $a = 4$.

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 [7] 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.

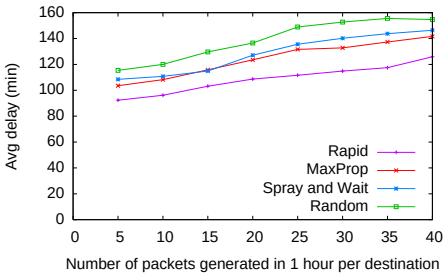

Figure 4: (Trace) Average Delay: RAPID has up to 20% lower delay than *MaxProp* and up to 35% lower delay than *Random*

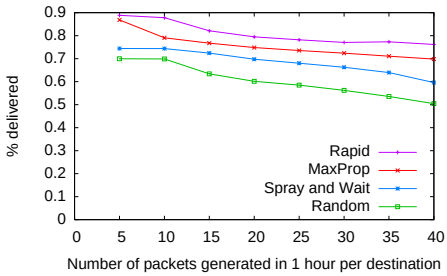

Figure 5: (Trace) Delivery Rate: RAPID delivers up to 14% more than *MaxProp*, 28% than *Spray and Wait* and 45% than *Random*

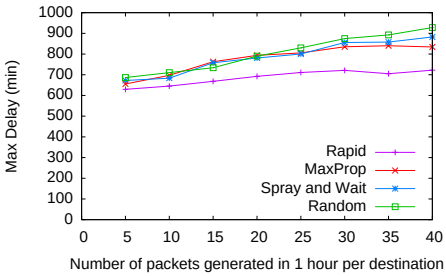

Figure 6: (Trace) Max Delay: Maximum delay of RAPID is up to 90 min lower than *MaxProp*, *Spray and Wait*, and *Random*

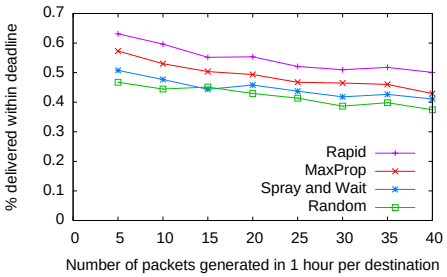

Figure 7: (Trace) Delivery within deadline: RAPID delivers up to 21% more than *MaxProp*, 24% than *Spray and Wait*, 28% than *Random*

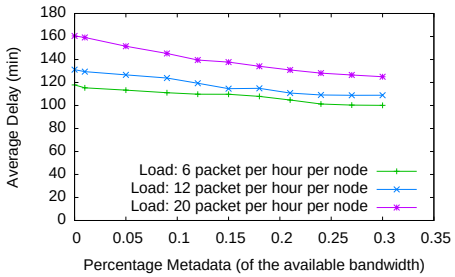

Figure 8: (Trace) Control channel benefit: Average delay performance improves as more metadata is allowed to be exchanged

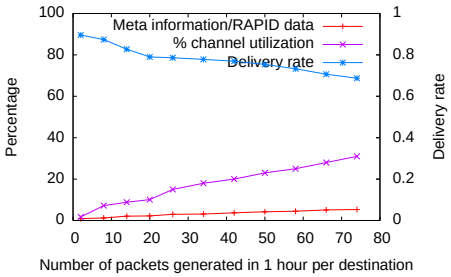

Figure 9: (Trace) Channel utilization: As load increases, delivery rate decreases to 65% but channel utilization is only about 35%

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 underutilized, and it is because of the bottleneck links in the network. The available bandwidth varies significantly across transfer opportunities in our bus traces [5].

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 of delivered packets decreases by up to 20 minutes when using a global channel. For the same experiments, the delivery rate when using an instant global channel increases by up to 12% (shown in Figure 11). Similarly, Figure 12 shows 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 [3]. 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

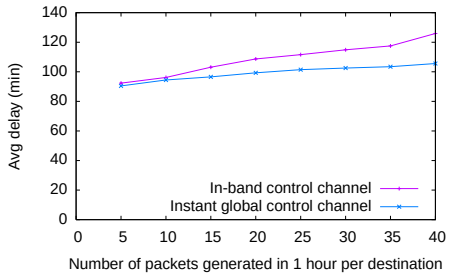

Figure 10: (Trace) Global channel: Average delay of RAPID decreases by up to 20 minutes using instant global control channel

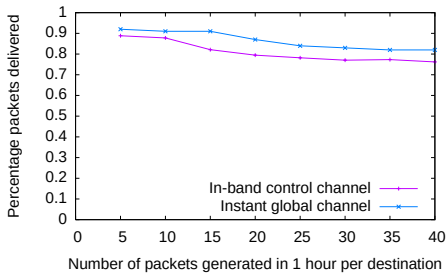

Figure 11: (Trace) Global channel: Delivery rate increases by up to 12% using an instant global control channel, for the average delay metric

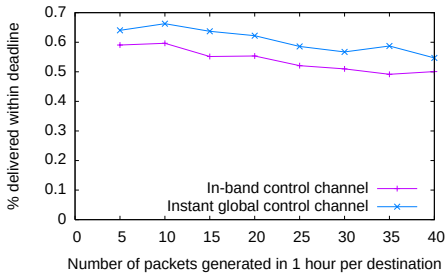

Figure 12: (Trace) Global channel: Packets delivered within deadline increases by about 15% using instant global control channel

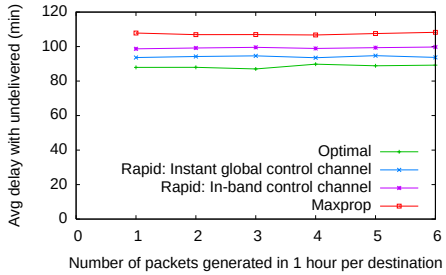

Figure 13: (Trace) Comparison with *Optimal*: Average delay of RAPID is within 10% of *Optimal* for small loads

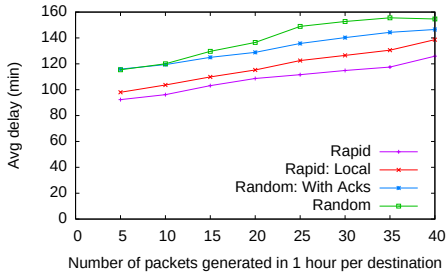

Figure 14: (Trace) RAPID Components: Flooding acks decreases average delay by about 8% and RAPID further decreases average delay by about 30% over *Random*

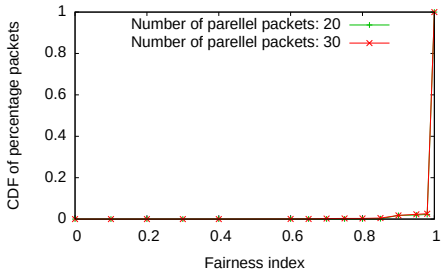

Figure 15: (Trace) RAPID Fairness: When 30 packets are created in parallel, computing Jain's fairness index indicates that RAPID is fair to parallel flows in terms of delay

when accurate channel information is available.

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 the Appendix. Our evaluations use the CPLEX solver [10]. 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. [18] 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 13 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 *Fairness*

Figure 15 presents the results of a fairness experiment on RAPID. In this experiment, we generate 20 to 30 parallel packets and compared the delay of packets that were created in parallel to analyze the fairness of RAPID. We use Jain's fairness index given by $\frac{d_i^2}{n.(\sum_i d_i)^2}$, where d_i is the delay of packet i and n is the number of parallel flows. A high fairness index indicates that the protocol is fair to the parallel flows. In Figure 15 the fairness index is 1 over 98% of the time even when for every packet, 29 others packets are generated in parallel. The number of packets generated per hour per node was set to 60 to ensure that there is contention for resources. This experiment indicates that RAPID's resource allocation is fair with respect to delays seen by packets created in parallel.

6.2.6 *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 14 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* [5], 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 an exponential and 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. All other parameters for exponential and powerlaw are identical.

6.3.1 Powerlaw mobility model: increasing load

Figure 16 shows the average delay for packets to be delivered (i.e., RAPID is set to use Eq. 1 as a metric). The average delay of packets quickly increase to 20 seconds as load increases in the case of *MaxProp*, *Spray and Wait* and *Random*. In comparison, RAPID's delay does not increase rapidly with increasing load, and is on an average 20% lower than all the three protocols.

Figure 17 shows the performance with respect to minimizing the maximum delay of packets (using Eq. 3 as a metric). RAPID reduces maximum delay by an average of 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.

We observe similar trends in Figure. 18, that shows the performance of the different routing protocols with respect to maximizing the number of packet delivered within an average deadline of 20 sec (RAPID uses Eq. 2).

6.3.2 Powerlaw mobility model: decreasing storage constraint

In this set of experiments we varied available buffer from from 10 KB to 280 KB and compared the performance of the four routing protocols. The load is fixed to 20 packets per destination and generated every 50 seconds. When the buffer size is increased to greater than 280 KB, the average delay of all four protocols tend to be stable.

Figure **??** shows how the average delay of all four protocols vary with increase storage availability. RAPID is able to maintain low delays even when only 10 KB space is available at each node. In comparison, *MaxProp*, *Spray and Wait* and *Random* have an average 23% higher delay.

Figure 20 shows a similar performance trend in terms of minimizing maximum delay. Similar to other experiments, the difference in performance between RAPID and the other three protocols is more marked for the maximum delay metric.

Figure 21 shows how constrained buffers varied affect the delivery deadline metric. 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. These experiments suggest that RAPID's utility-driven approach adapts well to storage restrictions as well.

6.3.3 Exponential mobility model

In the final set of experiments using synthetic mobility model, we measured the performance of the routing protocols using exponential mobility model. Figure 22 shows that average delay of RAPID is on an average 20% lower than the other three protocols. The average delay value for exponential mobility model is similar to the average delay value when the mobility model is power law. When mobility model is power law, the meeting times between nodes is skewed and may result in high delays or low delays. But on an average, the delay values are similar to when mobility is uniform exponential.

Figure 23 shows the maximum delay of all four protocols as load increases. The maximum delay of RAPID is the lowest and is on an average 25% lower than *MaxProp*, the second best performing protocol. Similarly,

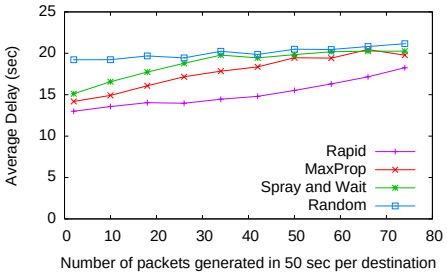

Figure 16: (Powerlaw) Avg Delay: RAPID reduces delay by about 20% compared to *MaxProp*, and 23% than *Spray and Wait* and 25% than *Random*

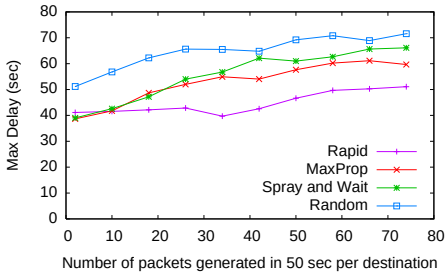

Figure 17: (Powerlaw) Max delay: RAPID's max delay is about 30% lower than *MaxProp*, 35% lower than *Spray and Wait* and 45% lower than *Random*

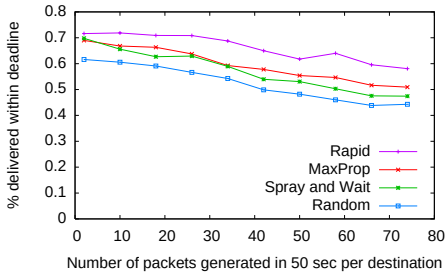

Figure 18: (Powerlaw) Delivery Deadline: RAPID delivers about 20% more packets within deadline when buffer size is constrained, compared to *MaxProp*, and 45% more packets compared to *Spray and Wait* and *Random*

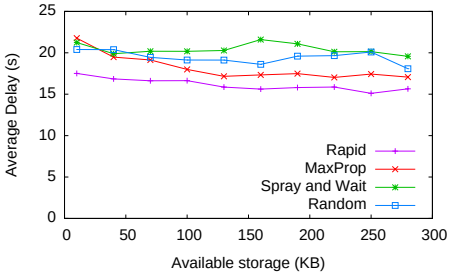

Figure 19: (Powerlaw) Avg Delay with constrained buffer: RAPID reduces average delay by about 23%when buffer size is constrained compared to *MaxProp*, *Spray and Wait* and *Random*

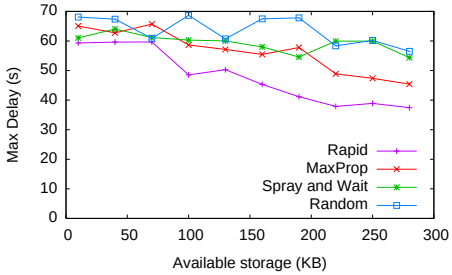

Figure 20: (Powerlaw) Max delay with constrained buffer: RAPID's max delay is about 22% lower than *MaxProp*, 35% lower than *Spray and Wait* and 38% lower than *Random* when buffer is constrained

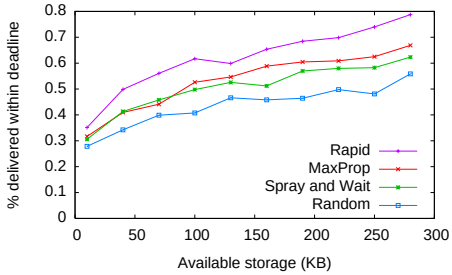

Figure 21: (Powerlaw) Delivery Deadline with constrained buffer: RAPID delivers about 20% more packets within deadline when buffer size is constrained compared to *MaxProp*, and 45% more than *Spray and Wait* and *Random*

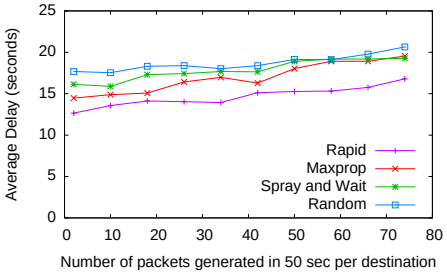

Figure 22: (Exponential) Avg Delay: RAPID decreases delay by about 20% compared to *MaxProp*, *Spray and Wait* and *Random*

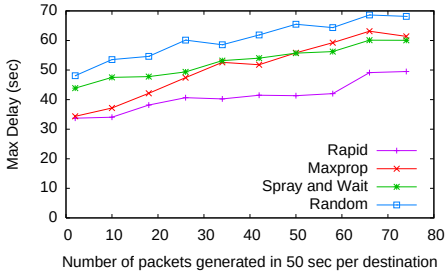

Figure 23: (Exponential) Max delay: RAPID's max delay is about 25% lower than *MaxProp* and *Spray and Wait* and 40% lower than *Random*

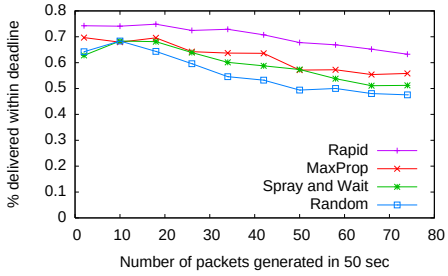

Figure 24: (Exponential) Delivery Deadline: RAPID delivers about 12% more packets within deadline compared to *MaxProp*, about 23% than *Spray and Wait* and 25% than *Random*

Figure 24 shows that RAPID is able to deliver about 12% more packets than *MaxProp* and about 23% more packets than *Spray and Wait* and *Random*.

We observed similar trends for increasing storage restrictions when using exponential mobility model (not shown in figure). The above experiments indicate that RAPID is able to adapt well to bandwidth and storage constraints, and is able to perform better than existing routing protocols for all three routing metrics in different mobility scenarios.

6.4 Discussion

The above experiments show that RAPID performs well from many viewpoints. However, there are limitations to our approach. The heuristics we use are suboptimal 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. Our resource allocation formulation was inspired by the utility-theoretic framework [20] for Internet-like low-feedback-delay networks pioneered by Kelly. However, we have not shown the ability of RAPID's local utility maximization approach to achieve the global optima for different routing performance objectives; unlike [20], our benefit (cost) function is not always strictly concave (convex) and smooth. 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 our java implementation of RAPID will enable us to enlist others to deploy RAPID on their DTNs, diversifying our results to other scenarios. We will also investigate encoding other application-specific metrics, including consistency requirements and power management.

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 metrics. Our protocol, RAPID, treats DTN routing as a resource allocation problem, making use of casually propagated meta-data that reports on network delay and capacity. Although our approach is heuristical, we have proven that the general DTN routing protocol lacks sufficient information in practice to solve optimally. Moreover, we have shown an optimal solution 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 65 days of testbed measurements that RAPID yields significant performance gains over previous work.

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. We thank Karthik Thyagarajan for his help in formulating the Integer Linear Program. This research was supported in part by National Science Foundation awards NSF-0133055 and CNS-0519881.

8. REFERENCES

[1] One laptop per child. `http://www.laptop.org`.
[2] TIER Project, UC Berkeley. `http://tier.cs.berkeley.edu/`.
[3] N. Banerjee, M. D. Corner, and B. N. Levine. An Energy-Efficient Architecture for DTN Throwboxes. In *Proc. IEEE Infocom*, May 2007.
[4] 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.
[5] J. Burgess, B. Gallagher, D. Jensen, and B. N. Levine. MaxProp: Routing for Vehicle-Based Disruption- Tolerant Networks. In *Proc. IEEE Infocom*, April 2006.
[6] 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.
[7] G. Casella and R. L. Berger. *Statistical Inference*. Second Edition. Duxbury, 2002.
[8] 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.
[9] 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.
[10] CPLEX. `http://www.ilog.com`.
[11] 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.
[12] 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.
[13] R. Gallager. A Minimum Delay Routing Algorithm Using Distributed Computation. In *IEEE Trans. on Communications*, volume 25, pages 73–85, Jan 1977.
[14] 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.
[15] 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.
[16] B. Hull et al. CarTel: A Distributed Mobile Sensor Computing System. In *Proc. ACM SenSys*, pages 125–138, Oct. 2006.
[17] 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.
[18] S. Jain, K. Fall, and R. Patra. Routing in a Delay Tolerant Network. In *Proc. ACM Sigcomm*, pages 145–158, Aug. 2004.

[19] E. Jones, L. Li, and P. Ward. Practical Routing in Delay-Tolerant Networks. In *Proc. ACM Chants Workshop*, pages 237–243, Aug. 2005.
[20] 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.
[21] J. Leguay, T. Friedman, and V. Conan. DTN Routing in a Mobility Pattern Space. In *Proc. ACM Chants Workshop*, pages 276–283, Aug. 2005.
[22] A. Lindgren, A. Doria, and O. Schelén. Probabilistic Routing in Intermittently Connected Networks. In *Proc. SAPIR Workshop*, pages 239–254, Aug. 2004.
[23] A. Maffei, K. Fall, and D. Chayes. Ocean Instrument Internet. In *Proc. AGU Ocean Sciences Conf.*, Feb 2006.
[24] W. Mitchener and A. Vadhat. Epidemic Routing for Partially Connected Ad hoc Networks. Technical Report CS-2000-06, Duke Univ., 2000.
[25] J. Ott and D. Kutscher. A Disconnection-Tolerant Transport for Drive-thru Internet Environments. In *Proc. IEEE INFOCOM*, pages 1849–1862, Mar. 2005.
[26] C. Papadimitriou. *Computational Complexity*. Addison Wesley, 1994.
[27] 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.
[28] 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.
[29] T. Small and Z. Haas. Resource and Performance Tradeoffs in Delay-Tolerant Wireless Networks. In *Proc. ACM WDTN*, pages 260–267, Aug. 2005.
[30] 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.
[31] T. Spyropoulos, K. Psounis, and C. S. Raghavendra. Performance analysis of mobility-assisted routing. In *ACM MobiHoc*, pages 49–60, May 2006.
[32] T. Spyropoulos and K. Psounis and C. Raghavendra. Single-copy Routing in Intermittently Connected Mobile Networks. In *IEEE SECON*, October 2004.
[33] J. Widmer and J.-Y. Le Boudec. Network Coding for Efficient Communication in Extreme Networks. In *Proc. ACM WDTN*, pages 284–291, Aug. 2005.
[34] 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.
[35] 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.
[36] X. Zhang, G. Neglia, J. Kurose, and D. Towsley. Performance Modeling of Epidemic Routing. In *Proc. IFIP Networking*, May 2006.

APPENDIX

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 each tuple represents the source, destination, size, and time of creation (at the source), respectively, of a packet.

A DTN routing algorithm computes a feasible schedule of packet transfers, where *feasible* means that, within the constraint that at each transfer opportunity, the total size of packets transferred is less than the size of the transfer opportunity.

A. COMPETITIVE HARDNESS OF ONLINE DTN ROUTING

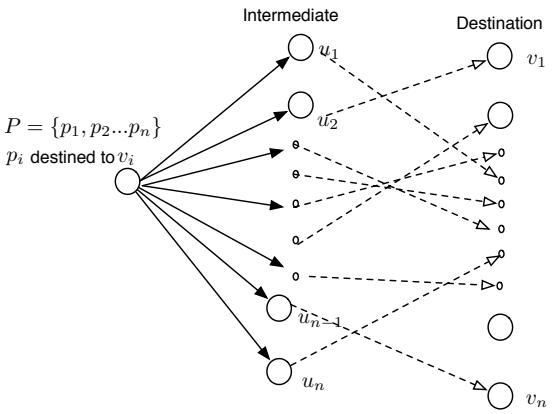

Figure 25: DTN node meetings for Theorem 7. Solid arrows represent node meetings known a priori to the online algorithm while dotted arrows represent meetings revealed subsequently by an offline adversary.

Let ALG be any deterministic online DTN routing algorithm with unlimited computational power.

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

PROOF. We prove the theorem by constructing an offline adversary, ADV, that incrementally generates a node meeting schedule after observing the actions of 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 node meetings, can deliver at most 1 packet.

Consider a DTN with n nodes and n packets as illustrated in Fig. 25, where $P = \{p_1, p_2, \ldots, p_n\}$ denotes a set of unit-sized packets; $I = \{u_1, u_2, \ldots, u_n\}$ denotes a set of intermediate nodes; and $D = \{v_1, v_2, \ldots, v_n\}$ denotes a set of nodes to which the packets are respectively destined, i.e. p_i is destined to v_i for all $i \in [1, n]$. The following procedure describes ADV's actions given ALG as input.

PROCEDURE FOR ADV:

- **Step 1**: ADV generates a set of node meetings

involving unit-size transfer opportunities at time $t = 0$ between A and each of $i_1, \ldots, i_n$ respectively (refer to Figure 25).

- **Step 2**: At time $t_1 > 0$, ADV observes the set of transfers X made by ALG. Without loss of generality, $X : P \to I$ is represented as a (one-to-many) mapping where $X(p_i)$ is the set of intermediate nodes to which ALG replicates packet p_i.
- **Step 3**: ADV generates the next set of node meetings $(i_1, Y(i_1)), (i_2, Y(i_2)), \ldots, (i_n, Y(i_n))$ at time t_1, where $Y : I \to D$ is a bijective mapping from the set of intermediate nodes to the destination nodes.

ADV uses the following procedure to generate the mapping Y given X in Step 3.
PROCEDURE GENERATE_Y(X):

1. Initialize $Y(p_i)$ to null for all $i \in [1, n]$;
2. for each $i \in [1, n]$ do
3. if $\exists j : u_j \notin X(p_i)$ and $Y(u_j) =$ null, then
4. Map $Y(u_j) \to v_i$ for the smallest such j;
5. else
6. Pick a j: $Y(u_j) =$ null, and map $Y(u_j) \to v_i$
7. endif

LEMMA 1. *ADV executes Line 6 in* GENERATE_Y(X) *at most once.*

PROOF. We first note that the procedure is well defined at Line 6: each iteration of the main loop map exactly one node in I to a node in D, therefore a suitable j such that $Y(u_j) =$ null exists. Suppose ADV first executes Line 6 in the i'th iteration. By inspection of the code, the condition in Line 3 is false, therefore each intermediate node u_k, $k \in [1, n]$, either belongs to $X(p_i)$ or is mapped to some destination node $Y(u_k) \neq$ null. Since each of the $i - 1$ previous iterations must have executed Line 4 by assumption, exactly $i - 1$ nodes in I have been mapped to nodes in D. Therefore, each of the remaining $n - i + 1$ unmapped nodes must belong to $X(p_i)$ in order to falsify Line 3. Line 6 maps one of these to v_i leaving $n - i$ unmapped nodes. None of these $n - i$ nodes is contained in $X(p_k)$ for $k \in [i + 1, \ldots, n]$. Thus, in each of the subsequent $n - i$ iterations, the condition in Line 3 evaluates to true. □

LEMMA 2. *The schedule of node meetings created by Y allows ALG to deliver at most one packet to its destination.*

PROOF. For ALG to deliver any packet p_i successfully to its destination v_i, it must be the case that some node in $X(p_i)$ maps to v_i. Such a mapping could not have occurred in Line 3 by inspection of the code, so it must have occurred in Line 6. By Lemma 1, Line 6 is executed exactly once, so ALG can deliver at most one packet. □

LEMMA 3. *The schedule of node meetings created by Y allows ADV to deliver all packets to their respective destinations.*

PROOF. We first note that, by inspection of the code, Y is a bijective mapping: Line 4 and 6 map an unmapped node in I to v_i in iteration i and there are n such iterations. So, ADV can route p_i by sending it $Y^{-1}(v_i)$ and subsequently to v_i. □

Theorem 1(a) follows directly from Lemmas 2 and 3.

COROLLARY 1. *ALG can be arbitrarily far from ADV with respect to average delivery delay.*

PROOF. The average delivery delay is unbounded for ALG because of undelivered packets in the construction above while it is finite for ADV. If we assume that that ALG can eventually deliver all packets after a long time T (say, because all nodes connect to a well-connected wired network at the end of the day), then ALG is $\Omega(T)$-competitive with respect to average delivery delay using the same construction as above. □

We remark that it is unnecessary in the construction above for the two sets of n node meetings to occur simultaneously at $t = 0$ and $t = t_1$, respectively. The construction can be easily modified to not involve any concurrent node meetings.

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

PROOF. We prove the theorem by constructing a procedure for ADV to incrementally generate a packet workload by observing ALG's transfers at each step. As before, we only need unit-sized transfer opportunities and packets for the construction.

Consider the basic DTN "gadget" shown in Fig. 26(a) involving just six node meetings. The node meetings are known in advance and occur at times T_1 and $T_2 > T_1$. respectively. The workload consists of just two packets $P = \{p_1, p_2\}$ destined to v_1 and v_2, respectively.

LEMMA 4. *ADV can use the basic gadget to force ALG to drop half the packets while itself delivering all packets.*

PROOF. The procedure for ADV is as follows. If ALG transfers p_1 to v_1' and p_2 to v_2', then ADV generates two more packets: 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'. ADV can deliver all four packets by transferring p_1 and p_2 to v_2' and v_1' respectively at time T_1, which is the exact opposite of ALG's choice.

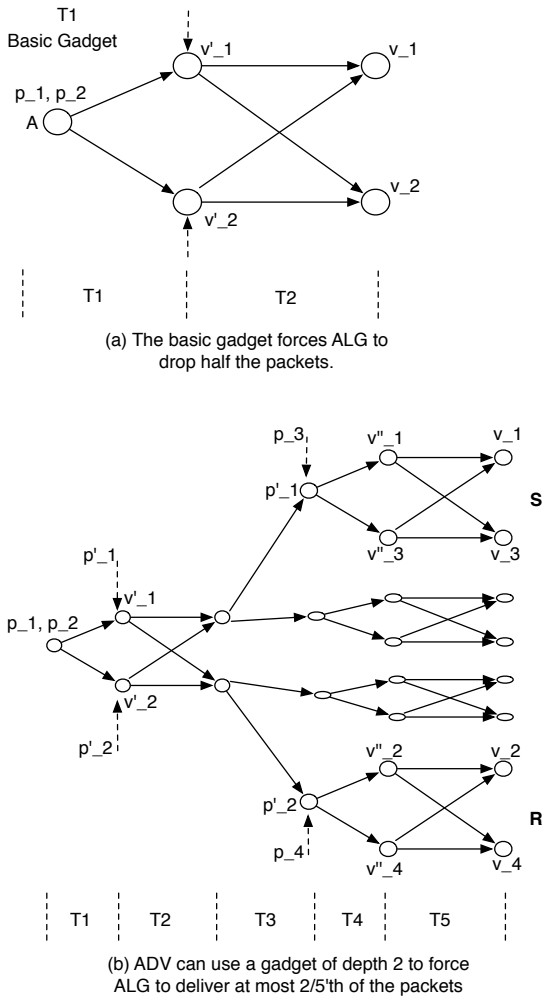

Figure 26: DTN construction for Theorem 7. Solid arrows represent node meetings known a priori to ALG while vertical dotted arrows represent packets created by ADV at the corresponding node.

If ALG instead chooses to transfer p_1 to v'_2 and p_2 to v'_1, ADV chooses the opposite strategy.

If ALG chooses to replicate one of the two packets in both transfer opportunities at time T_1 while dropping the other packet, ADV simply deliver both packets. Hence the lemma. □

Next, we extend the basic gadget to show that ALG can deliver at most a third of the packets while ADV delivers all packets. The corresponding construction is shown in Figure 26(b).

The construction used by ADV composes the basic gadget repeatedly. Consider the gadget S attached to v''_1. Without loss of generality, suppose ALG dropped p_1 at T_2 and is left with p'_1 at v''_1.

Suppose ALG replicates p'_1 to the head of gadget S at the top right. In response, ADV introduces a packet p_3 at S's head destined to v_3 resulting in another instance of the basic gadget. By Lemma 4, ALG is forced to drop (w.l.o.g.) p'_1 and p_3 and proceed with p''_1 and p'_3. The process at gadget R at the bottom right is similar. Thus, at time T_3, ALG has dropped the 4 packets $p_1, p_2, p'_1, p_3, p'_2, p_4$ while hoping to deliver p''_1, p'_3, p''_2, p'_4, i.e., ALG has dropped 6 out 10 packets achieving a potential delivery rate of at most 2/5. Even if ALG replicates p_1 on both edges adjacent to v''_1, ADV can ensure that ALG delivers at most 2/5'th of the packets by creating another basic gadget for each replica.

In contrast, we can show that ADV can deliver all packets it creates by following the same strategy as in the basic gadget in Lemma 4 throughout the course.

Similarly, by creating a gadget of depth 3, we can show that ADV can force ALG to deliver at most 4/11'th of the packets. Effectively, each new basic gadget introduces 3 more packets and forces ALG to drop 2 more packets. In particular, with a gadget of depth i, ADV can limit ALG's delivery rate to $i/(3i-1)$. Thus, by composing a sufficiently large number of basic gadgets, ADV can limit the delivery rate of ALG to a value arbitrarily close to 1/3.

Hence, Theorem 1(b).

B. COMPUTATIONAL HARDNESS OF THE DTN ROUTING PROBLEM

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 and has a lower bound of $\Omega(n^{1/2-\epsilon})$ on the approximation ratio.

PROOF. Consider a DTN routing problem with n nodes that have complete knowledge of node meetings and workload *a priori*. The input to the DTN problem is the set of nodes $1, \ldots, n$; a series of transfer opportunities $\{(u_1, v_1, s_1, t_1), (u_2, v_2, t_2, s_2, t_2), \ldots\}$ such that $u_i, v_i \in [1, n]$, s_i is the size of the transfer opportunity, and t_i is the time of meeting; and a packet workload $\{p_1, p_2, \ldots p_s\}$, where $p_i = (u'_i, v'_i, s'_i, t'_i)$, where $u', v' \in [1, n]$ are the source and destination, s' the size, and t' the time of creation of the packet, respectively. The goal of a DTN routing algorithm is to compute a feasible schedule of packet transfers, where *feasible* means that the total size of transferred packets in any transfer opportunity is less than the size of the transfer opportunity.

The decision version $O_{n,k}$ of this problem is: Given a DTN with n nodes such that nodes have complete knowledge of transfer opportunities and the packet workload, is there a feasible schedule that delivers at least k packets?

LEMMA 5. *$O(n, k)$ is NP-hard.*

PROOF. We show that $O(n,k)$ is a NP-hard problem using a polynomial-time reduction from the edge-disjoint path (EDP) problem for a directed acyclic graph (DAG) to $O(n,k)$. The EDP problem for a DAG is known to be NP-hard [9].

The decision version of EDP problem is: Given a DAG $G=(V,E)$. where $|V|=n$, $E \in V \times V$: $e_i = (u_i, v_i) \in E$, if e_i is incident on u_i and v_i and direction is from u_i to v_i. If given source-destination pairs $\{(s_1,t_1),(s_2,t_2)...(s_s,t_s)\}$, do a set of edge-disjoint paths $\{c_1, c_2...c_k\}$ exist, such that c_i is a path between s_i and t_i, where $1 \leq i \leq k$.

Given an instance of the EDP problem, we generate a DTN problem $O(n,k)$ as follows:

As the first step, we topologically order the edges in G, which is possible given G is a DAG. The topological sorting can be performed in polynomial-time.

Next, we label edges using natural numbers with any function $l : E \rightarrow \mathbb{N}$ such that if $e_i = (u_i, u_j)$ and $e_j = (u_j, u_k)$, then $l(e_i) < l(e_j)$. There are many ways to define such a function l. One algorithm is:

1. label = 0
2. For each vertex v in the decreasing order of the topological sort,
   (a) Choose unlabeled edge $e = (v,x) : x \in V$,
   (b) label = label + 1
   (c) Label e; l(e) = label.

Since vertices are topologically sorted, if $e_i = (u_i, u_j)$ then $u_i < u_j$. Since the algorithm labels all edges with source u_i before it labels edges with source u_j, if $e_j = (u_j, u_k)$, then $l(e_i) < l(e_j)$.

Given a G, we define a DTN routing problem by mapping V to the nodes $(1,..,n)$ in the DTN. The edge $(e = \{u,v\} : u,v \in V)$ is mapped to the transfer opportunity $(u,v,1,l(e))$, assuming transfer opportunities are unit-sized. Source and destination pairs $\{(s_1,t_1),(s_2,t_2),\ldots,(s_s,t_s)\}$ are mapped to packets $\{p_1,p_2,\ldots,p_s\}$, where $p_i = (s_i,t_i,1,0)$. In other words, packet p is created between the corresponding source-destination pair at time 0 and with unit size. A path in graph G is a valid route in the DTN because the edges on a path are transformed to transfer opportunities of increasing time steps. Moreover, a transfer opportunity can be used to send no more than one packet because all opportunities are unit-sized. If we solve the DTN routing problem of delivering k packets, then there exists k edge-disjoint paths in graph G, or in other words we can solve the EDP problem. Similarly, if the EDP problem has a solution consisting of k edge-disjoint paths in G, at least k packets can be delivered using the set of transfer opportunities represented by each path. Using the above polynomial-time reduction, we show that a solution to EDP exists if and only if a solution to $O(n,k)$ exists. Thus, $O(n,k)$ is NP-hard. □

COROLLARY 2. *The DTN routing problem has a lower bound of $\Omega(n^{1/2-\epsilon})$ on the approximation ratio.*

PROOF. The reduction given 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. Therefore, this reduction is an L-reduction [26]. Consequently, the lower bound $\Omega(n^{1/2-\epsilon})$ known for the hardness of approximating the EDP problem [15] holds for the DTN routing problem as well. □

Hence, Theorem 2.

C. DELAY ESTIMATION BASED ON DEPENDENCY GRAPHS

In Section 3, we presented ESTIMATE_DELAY that estimates expected delays of packets based on their position in node buffers. The algorithm ignores some dependencies between delivery delay distributions of packets across node buffers. In this section, we first present an idealized algorithm to accurately estimate expected delays assuming that a global channel is available to track global state exactly.

To understand the simplifying assumption in ESTIMATE_DELAY, we introduce some notation. Let $G = (V,E)$ be a graph representing a markov network with vertices $V = \{V_1 \cup V_2 \cup \ldots \cup V_m\}$ where $V_i = \{x_{i,1}, x_{i,2}, \ldots, x_{i,k_i}\}$ is the set of k_i replicas of packet i. All packets in V are destined to the same DTN node[3] — recall that we wish to estimate expected delays of packets based on the current state of the network assuming no further replication, so packets destined to other DTN nodes do not affect the delays of packets in V. An edge (or a path) from one node to another indicates a dependency between the delivery time distributions of the corresponding packets. The edges are constructed as follows.

- Each replica is connected to its successor, i.e., the replica immediately ahead of it in the current buffer.
- Each replica is connected to all the replicas of its successor at other DTN node buffers.

Refer to an example dependancy graph shown earlier in Figure 2.

ESTIMATE_DELAY ignored the dependancies created by non-vertical edges in the dependancy graph. We present an idealized algorithm DAG_DELAY, to compute expected delays of packets given the complete dependency graph without ignoring any dependancies. We call this algorithm *idealized* because its implementation

[3]We distinguish a "DTN node" from a node in the dependency graph; the latter represents a packet replica.

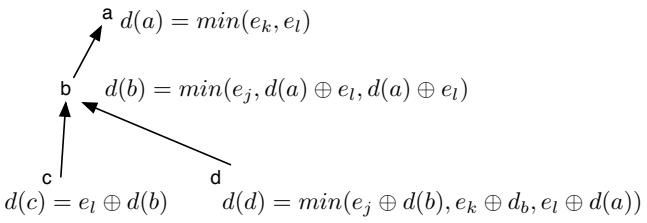

Figure 27: A topologically sorted dependancy graph.

requires a global control channel such as that introduced in section 6.2.3.

Let the delay distribution of a packet in buffer x be e_x. Let $\oplus$ represent the addition of two distributions (e.g., adding two identical exponential distributions yields a gamma distribution with twice the mean). Assume unit-sized transfer opportunity and packet. We topologically sort the dependancy graph. The topological sort of the example dependency graph in Figure 2 is shown in Figure 27. DAG_DELAY computes the delay of the packets in the graph in the topologically order starting from the top. The information maintained for each of the k replicas $p_1, \ldots, p_k$ of packet p is $\{succ(p_j), e_{node(p_j)}\}$, $1 \leq j \leq k$, where $succ(p_j)$ is the successor of the replica p_j, and $node(p_j)$ is the DTN node buffer where p_j exists.

PROCEDURE DAG_DELAY(p):

1. for each replica $p_j, 1 \leq j \leq k$ of p, do
   (a) Let $s = succ(p_j)$, and $n = node(p_j)$
   (b) if $d(s)$ is not defined, then
      i. $d(s) =$DAG_DELAY(s)
   (c) $d'(p_j) = d(s) \oplus e_n$
2. return $d(p) = min(d'(p_1), \ldots, d'(p_k))$

Figure 27 presents the delay of each packet as computed using DAG_DELAY. Although the algorithm is recursive, sequentially computing the delay of packets top down in the DAG and storing the delay values of already computed packets ensures that the delay of each packet is computed exactly once. And since the DAG has no cycles, DAG_DELAY will converge.

When the transfer opportunities are not unit-sized, the delivery delay cannot be estimated using the above algorithm. For example, refer to Figure 2. If the transfer opportunity and packets are unit-sized, then the delay of packet b depends on the delay of packet a. However, if the transfer opportunities are greater than one unit, then the delay of b may not depend on the delay of a, and the dependancy graph is no longer valid.

Our implementation of RAPID uses ESTIMATE_DELAY instead of DAG_DELAY to estimate expected delays. ESTIMATE_DELAY estimates expected delays by ignoring non-vertical edges in the dependency DAG. The delays estimated by ESTIMATE_DELAY is not order-preserving manner. i.e., if the expected delay d_i for packet i is less than the expected delay d_j of packet j as estimated by DAG_DELAY, ESTIMATE_DELAY may estimate the delay d'_i of i to be greater than the delay d'_j of j. So the estimation error can be arbitrarily large for certain pathological cases. However, ESTIMATE_DELAY is simple, local, and computationally efficient heuristic to estimate expected delays. Also, it works well in practice.

D. ILP FORMULATION:

We formulate the DTN routing problem as an Integer Linear Program (ILP) to minimize average delay. Jain et al. [18] solve a similar DTN routing problem but allow packets to be fragmented across links and mapped non-zero propagation delays on the links. This severely limited the size of the network and the number of packets they could evaluate. In comparison, our formulation lets us obtain the optimal solution for realistic DTNs with small to moderate workloads.

First, we divide time into discrete intervals so every node meets at most one other node in an interval. The inputs to the problem are as follows.

- The set of time intervals $I = 1, 2, \ldots, h$. The function b returns the beginning of the interval. e returns the end of an interval and variable h represents the last interval
- The set of nodes in the network N
- The set of edges E. An edge is defined when two nodes meeting in an interval. We define functions f and s to return the first and the second node that meet respectively, d returns the interval in which the edge is defined and b represents the bandwidth available during the meeting. When two nodes i and j meet, they are represented two edges e and e' on either direction. $E_{(x,y)}$ represents an edge with source x and destination y.
- The set of packets P. Function st return the source of the packet, dt return the destination of the packet, c returns the interval in which the packet was created, t returns time the packet was created and $size()$ returns the size of the packet.

The variables are

- $X(p \in P, e \in E) = 1$ if j is forwarded over the edge e and is 0 otherwise
- $N(p \in P, n \in N, i \in I) = 1$ if node n has packet p in the interval i and is 0 otherwise
- $D(p \in P, i \in I) = 1$ if packet p is delivered before interval i and is 0 otherwise

X can be used to construct the optimal path taken by

a packet.

$$min \sum_{p \in P} \sum_{i \in I} \sum_{e \in E_{(dt(p),i)}} (b(i) - t(p)) \cdot X(p,e) + \sum_{p \in P} (1 - D(p, e(h)) \cdot (b(h) - t(p))$$

All constraints use notations $\forall p, n, i, e$ to mean $\forall p \in P, \forall n \in N, \forall i \in I$ and $\forall e \in E$. The constraints are

Initialization constraints

$N(p,n,i) = 0 \ if \ i < c(p) \ \forall p, n, i$

$N(p,n,i) = 1 \ if \ st(p) = n$ and $c(p) = i \ \forall p$

Bandwidth constraint

$$\sum_{p \in P} (X(p,e) * size(p) \leq b(e) \ \forall e$$

Transfer constraints

$$N(p,n,i-1) - \sum_{e \in E_{(i,n)}} X(p,e) \ \forall \ p,n,i$$

$$\sum_{e \in E_{(n,i)}} X(p,e) - N(p,n,i) = 0 \ \forall p,n,i$$

$N(p, f(e), d(e) - 1) - X(p,e) >= 0 \ \forall p, e$

Conservation constraint

$$1 - \sum_{n \in N} N(p,n,i) = 0 \ if \ i > c(p) \ \forall p, e$$

Delivery Constraint

$$D(p,i) - \sum_{e \in E_{(dt(p),.)} : d(e) < i} X(p,e) = 0 \ \forall \ p, i$$