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

ABSTRACT

Many DTN routing protocols use a variety of mechanisms including discovering the meeting probabilities among peers, packet replication and network coding. The primary focus of these mechanisms is to increase the likelihood of finding a path with limited information, so these approaches have only an “incidental” effect on various routing metrics such as delivery latency. In this paper, we present RAPID, an “intentional” DTN routing protocol that can optimize a specific routing metric such as worst-case delivery latency or the fraction of packets that meet their deadlines. The key insight is to treat DTN routing as a resource allocation problem that translates the routing metric into per-packet utilities that determine how packets should be replicated in the system.

We evaluate RAPID rigorously through an implemented prototype of RAPID over a vehicular DTN testbed of 40 nodes and simulations based on real traces. To our knowledge this is the first routing protocol deployed in a real DTN at scale. Our results suggest that RAPID significantly outperforms existing routing protocols for several routing 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 [18], ocean sensor networks [24, 27, 22], vehicular networks [26, 3], and projects such as Tier-Store [2], Digital Study Hall [12], 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. These characteristics make routing difficult in DTNs because of inherent uncertainty about the state of the network.

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 peers meeting probabilities, packet replication, network coding, placement of stationary waypoint stores, and using 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 throughput, 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 if the DTN maximizes the number of news stories delivered before they are outdated, rather than maximizing the number of stories that are eventually delivered.

In this paper, we present the Resource Allocation Protocol for Intentional DTN routing (RAPID), a protocol 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 defined per-packet utilities that determine if the marginal utility of replicating a packet justifies the resources used. DTNs are resource-constrained networks, in terms of transfer bandwidth, energy, and storage; allocating resources to replicas without careful attention to available resources can cause more harm than good. RAPID loosely tracks network resources through a distributed control plane to assimilate a local view of global state.

The design of RAPID’s control plane, that we call a causal channel, is based on information that is exchanged by peers, an approach that builds on insights from previous work. For example, Jain et al [16] suggest that DTN routing protocols that use more knowledge of network conditions perform better. Burgess et al [3] show that flooding acknowledgments is an effective method of improving delivery rates by removing useless packets from the system, freeing resources. RAPID reports on additional metadata, including the number and location of extant 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 result in significantly improved routing performance compared to existing approaches.

We have built and deployed RAPID on a vehicular DTN testbed that intermittently connects 40 buses in a 150 square-mile area carrying 802.11b radios and moderately resourceful computers. To our knowledge, this
paper reports on the first DTN routing algorithm deployed at this scale. We also conduct a simulation-based evaluation based on 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 two weeks. 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 three existing routing protocols [21, 31, 3] and random routing using a larger, 65-day trace of our testbed’s transfer opportunities. We also compare the protocols using synthetic mobility models.

To show the generality of our RAPID approach, we evaluate three separate routing metrics: minimizing average delay, minimizing worst-case delay, and maximizing the number of packets that are delivered before a deadline. Our evaluations using trace-driven and synthetic mobility scenarios demonstrate that RAPID significantly outperforms four other routing protocols. For example, in trace-driven evaluations, RAPID outperforms the second-best performing protocol by at least 20% for all three metrics. With a priori information about the mobility model, RAPID outperforms the second-best performing protocol by up to 50% for high loads. We also compare RAPID to an optimal protocol for low packet loads and show empirically that RAPID performs within 10% of optimal. All experiments include the cost of RAPID’s control plane. Furthermore, the “second-best” protocol is different for different metrics demonstrating RAPID’s intentional design.

Contributions. Our main contribution is demonstrating the feasibility of an intentional routing approach for DTNs. To this end,

- We present 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 (Section 3.3).
- We present results from a deployment of RAPID on a vehicular testbed that show performance in real scenarios, and we use the results to validate a trace-driven simulator (Section 5).
- We show using a 65-day trace of the testbed that in terms of reducing delay RAPID not only outperforms existing approaches with respect to specific metrics for delivered packets, but also delivers a larger fraction of packets (Section 6).
- We evaluate the feasibility of hybrid DTNs that combine a high-delay moderate-bandwidth DTN with a low-delay, low-bandwidth channel such as a cell-phone or long-range radio networks to improve RAPID’s distributed control plane.
- We substantiate RAPID’s heuristic approach with hardness results that prove that online algorithms without complete knowledge and unlimited computational power, or computationally limited algorithms with complete future knowledge can be arbitrarily far from optimal (Section 3.3).

2. RELATED WORK

Replication vs. 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 to avoid replication along paths with little chance of reaching the destination [9, 18, 4, 3, 20, 21]; (ii) removing useless packets using acknowledgments of delivered data [3]; (iii) using probabilistic mobility information to infer delivery [29]; (iv) replicating packets with a small probability [37]; (v) using network coding [36] and coding with redundancy [15]; and (vi) bounding the number of replicas of a packet [31, 29, 35, 24, 23].

In contrast, forwarding routing protocols maintain at most one copy of a packet in the network [16, 17, 28, 33]. Jain et al. [16] evaluates using oracles with varying degrees of future knowledge to minimize the average delay of forwarded packets. 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 bus speed, weather, radio interference, vehicular traffic, and system failure. Moreover, our interests lie in developing an algorithm that is as general as possible. Jones et al. [17] propose a link-state protocol based on epidemic propagation to disseminate global knowledge, but use a single path to forward each packet. Shah et al. [28] and Spyropoulos et al. [33] present an analytical framework for the forwarding-only case assuming a grid-based mobility model. They subsequently extend the model [34] and propose a replication-based protocol, Spray and Wait [31]. The consensus [16, 34, 24] appears to be that replicating packets can potentially improve performance over forwarding, but at a greater risk of degrading performance.

Incidental vs. Intentional.

Our position is that existing schemes only have an incidental effect on desired performance metrics, including commonly evaluated metrics like average delay or
Problem | Storage | Bandwidth | Routing | Previous work (and mobility model)  
---|---|---|---|---  
P1 | Unlimited | Unlimited | Replication | MobiSpace [20] (AP traces)  
P2 | Unlimited | Unlimited | Replication | Modified Djikstra’s algorithm Jain et al. [16] (simple graph)  
P3 | Finite | Unlimited | Replication | Davis et al. [9] (Simple partitioning synthetic), SWIM [29] (Exponential), Spray and Wait [31] (Exponential), MV [4] (Community-based synthetic), Prophet [21] (Community-based synthetic)  
P4 | Finite | Finite | Forwarding | Jones et al. [17] (AP traces), Jain et al. [16] (Synthetic DTN topology)  
P5 | Finite | Finite | Replication | This paper (Vehicular DTN traces, exponential, and powerlaw meeting probabilities, testbed deployment), MaxProp [3] (Vehicular DTN traces)  

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

delivery probability. Their theoretical intractability in general makes the effect of a particular mechanism or a protocol design decision on a given performance metric unclear. For example, several existing DTN routing algorithms [31, 29, 35, 24, 23, 3] route packets using the number of replicas as the heuristic, but the effect of replication varies with different routing metrics. In contrast, routing in RAPID is intentional with respect to a given performance metric. RAPID explicitly calculates the effect of replication on the specific routing metric it seeks to maximize. Furthermore, it removes delivered packets using acknowledgements and prevents useless packets from overloading the system.

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 model used. RAPID is a replication-based algorithm that assumes constraints on both storage and bandwidth (P5) — the most challenging but practical problem space. In addition, to our knowledge, RAPID is the only protocol deployed and evaluated on a real testbed.

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 [31, 21, 9, 4, 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 [10]. Finally, for mobile DTNs, and especially vehicular DTNs, transfer opportunities are short-lived [14, 3].

Storage at a mobile node can become a bottleneck when it accumulates in-transit data from other nodes much faster than it can deliver the data to its destination. We describe a natural extension of the basic RAPID protocol (Section 6) to accommodate storage constraints.

We were unable to find other protocols in P5 except MaxProp [3], which assumes 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 (Section 6) indicate that RAPID performs significantly better than MaxProp for each considered metric.

Some theoretical works [38, 32, 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, so limited bandwidth means limited size of transfers.

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 $(t, s)$ where $t$ is the time of the meeting and $s$ is the size of the transfer opportunity. The workload is a set of packets $P = \{ (t_1, s_1), (t_2, s_2), \ldots \}$ where the $i$th tuple represents the source, destination, size, and time
of creation (at the source), respectively, of packet $i$.

### 3.2 The case for a heuristic approach

Two fundamental reasons make a 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. Both hardness results hold even for unit-sized packets and unit-sized transfer opportunities.

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

- (a) If ALG has complete knowledge of the workload, 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, where $n$ is the number of packets in the workload.
- (b) If ALG has complete knowledge of the meeting schedule, but not of the packet workload, then it can deliver a most a third of packets in the presence of an optimal offline adversary compared to an offline algorithm.

**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 $\Omega(\sqrt{n})$ on polynomial-time approximability.

The proofs are outlined in the appendix. Furthermore, with respect to average delay, both the online as well as computationally limited algorithms can be shown to be arbitrarily far from optimal.

Finally, we note that traditional utility-theoretic optimization frameworks developed for congestion control [19] or routing [11] based on fluid models are difficult to extend to DTNs both due to the inherently high feedback delay and the discrete nature of transfer opportunities that are more suited for transferring large “bundles” as opposed to small packets.

### 3.3 RAPID Design

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

Consider a routing metric such as “minimize average delay of packets”, the running example used in this section. The corresponding utility $U_i$ 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 $\delta U_i$ 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 $\frac{\delta U_i}{s_i}$ among packets in its buffer.

In general, $U_i$ is defined as the expected contribution of $i$ to the given routing metric. 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 — it lowers the replication priority of packets that (i) already have many replicas, (ii) the peer node is “bad” 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 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 $j$.

Estimating the expected utility of a packet in a distributed manner is challenging. RAPID uses meta-data propagated via a control channel to determine the utility of a packet (Section 4.3). RAPID has two core components: a selection algorithm and an inference algorithm. 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.

### 3.4 The Selection Algorithm

The key steps in RAPID take place when two nodes are within radio range and have discovered one another. The protocol is symmetric; without loss of generality, we describe how node $Y$ determines which packets to transfer to node $X$.

RAPID also adapts to storage restrictions for in-transit data. If a node exhausts all available storage, packets with the highest utility are deleted first. RAPID removes packets with high utility to make room for new packets that have not been allocated enough resources. However, a source never deletes its own packet unless it receives an acknowledgment for the packet.

### 3.5 Inference algorithm

Next, we describe how this canonical version can support specific metrics using an inference algorithm. In Section 4, we present an algorithm to calculate the utility using meta-data. Table 2 summarizes all variables used in these two sections.
\[ D(i) = E[T(i)] + A(i) \]
\[ T(i) = \text{Time since creation of } i \]
\[ A(i) = \text{Expected delivery time} \]
\[ a(i) = \text{Random variable representing the time taken for the first replica to deliver } i \]
\[ n_1, \ldots, n_k = \text{Nodes known to have a copy of } i \]
\[ a_{n_j}(i) = \text{Random variable representing the time taken by } n_j \text{ to deliver } i \]
\[ k = \text{Estimated number of replicas of } i \text{ existing in network} \]
\[ m_{n_j}(i) = \text{Inter-meeting time between } n_j \text{ and the destination of } i \]

Table 2: Variables used by routing metric descriptions and estimation of utility of packet \( i \)

Algorithm 1.

1. **Initialization**: Nodes \( X \) and \( Y \) exchange metadata about packets in their buffer and metadata collected over past meetings. (Detailed in Section 4.3.)
2. **Delivery**: Nodes \( X \) and \( Y \) deliver packets destined to \( Y \) and \( X \) respectively.
3. **Per-packet Operation**: For each packet \( i \) in node \( Y \)'s buffer:
   (a) \( Y \) estimates the benefit of replicating \( i \) to \( X \), \( \delta U_i \)
   (b) \( Y \) sorts packets in the decreasing order of \( \frac{\delta U_i}{\delta} \) and starts the transfer.
4. **Termination**: Transfer ends when the other node is out of radio range or all eligible packets are transmitted.

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
\[ U_i = -D(i) \quad (1) \]
since the packet’s expected delay is its contribution to the performance metric. Essentially, the protocol attempts to greedily replicate the packet whose replication reduces the delay by the most.

3.5.2 Metric 2: Minimizing missed deadlines

\textsc{rapid} can also be set to minimize the number of packets that miss delivery deadlines. Let \( L(i) \) be the deadline of packet \( i \) and \( T(i) \) be the time since the packet was created. Let \( P(a(i) < L(i) - T(i)) \) be the probability that the packet will be delivered within time \( L(i) - T(i) \). Then the utility is defined as
\[ U_i = \begin{cases} P(a(i) < L(i) - T(i)), & L(i) > T(i) \\ 0, & \text{otherwise} \end{cases} \quad (2) \]

A packet that has missed its deadline can no longer improve performance and is thus assigned a value of 0. For other packets, the utility is the probability that the packet will be delivered within the deadline. The protocol replicates packet whose probability of delivery within deadline most improves.

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 follows.
\[ U_i = \begin{cases} -D(i), & D(i) \geq D(j) \quad \forall j \in S_Y \\ 0, & \text{otherwise} \end{cases} \quad (3) \]
where \( S_Y \) denote the the set of packets in the buffer of \( Y \). Thus, \( U_i \) as defined above is the negative expected delay if \( i \) is a packet with the maximum expected delay among all packets held by \( Y \). Using this definition, replication is useful only for the packet whose delay is maximum. For the routing algorithm to be work conserving, \textsc{rapid} computes utility for the packet whose delay is currently the maximum; i.e., once a packet with maximum delay is considered for replication, the utility of the remaining packets is recalculated using Eq. 3. In other words, \textsc{rapid} sorts the packets and replicates in the order of decreasing delay if the utility improvement is positive.

4. ESTIMATING UTILITIES

\textsc{rapid} estimates utilities as a function of expected delivery delay in Eqs. 1 and 3, or the probability of packet delivery within a deadline, in Eq. 2. Estimating the expected delivery delay is challenging. The exact answer requires knowledge of the minimum expected time until any node with the replica of the packet delivers the packet. Thus, a node must know which other nodes carry replicas of the packet and the delivery delay distribution of each replica.

Recall that \textsc{rapid} estimates delivery delay as \( D(i) = T(i) + E(a(i)) \), where \( T(i) \) is the time since packet creation and \( E(a(i)) \) is the expected time remaining for the first replica to be delivered (refer Table 2). \( T(i) \) is simply the difference between the current time and the creation timestamp carried in the packet. Given the delays of real DTNs, loose clock synchronization suffices.

Below, we first present an algorithm to estimate \( a(i) \) that uses all control information for a packet, including the number of replicas \( k \) and the nodes that carry the replica \( n_1, n_2, \ldots, n_k \). We then describe the sequence of meta-information exchanges between nodes to approximate this control information necessary to estimate \( a(i) \) in practice.
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 $[b_j(i)/B_j]$ 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 $[b_j(i)/B_j]$ times. Then, by definition
   \[ a_{n_j}(i) = r'_{jx} \] (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:
   \[ P(a(i) < t) = 1 - \prod_{y=1}^{k} (1 - P(a_{n_y}(i) < t)) \] (5)
6. Accordingly:
   \[ A(i) = E[a(i)] \] (6)

4.1 Estimating delivery delay

Assume that we know the distribution of inter-meeting times $m_{n_j}(i)$, i.e., the time node $n_j$ takes to meet $i$’s destination. Note that $n_j$ may not be able to deliver the packet when it meets the destination if the transfer opportunity is limited. The RAPID algorithm to calculate the delivery delay $a(i)$ is shown in Algorithm 2.

4.1.1 Simplifying Assumptions in Algorithm 2

Algorithm 2 makes a simplifying assumption (Steps 4 and 5) that the expected delay, $E(a(i))$, can be computed by treating the meetings of each node $n_j$ holding a replica of $i$ as an independent process. This assumption does not hold in general as exemplified below.

Figure 2 shows the position of packet replicas in the buffers of different nodes; packets with the same number are replicas. Packet $b$ may be delivered in two ways: (i) if node $j$ meet’s $b$’s destination, or (ii) one of nodes $k$ and $l$ meet a’s destination and then one of the them meet $b$’s destination. We can represent the delay dependencies as a graph as shown in Fig 2. The graph $G = (V,E)$ represents a markov network with vertices $V = \{ V_1 \cup V_2 \cup \ldots \cup V_m \}$ where $V_i = \{ x_{i,1}, x_{i,2}, \ldots, x_{i,k} \}$ is the set of $k_i$ replicas of packet $i$. An edge from one node to another indicates a dependency between the delivery time distributions of the corresponding packets. More specifically, we construct the edges such that each packet replica is connected to its immediate successor in the current buffer and all replicas of the successor in other buffers.

Algorithm 2 estimates expected meeting times ignoring all but vertical edges when estimated delays, which results in higher estimates of expected delivery delay. For example, Algorithm 2 estimates the delivery time distribution of $b$ as $\min(m_{n_j}(b), \min(m_{n_k}(a) \oplus m_{n_j}(b)), m_{n_i}(a) \oplus m_{n_j}(b))$, whereas the distribution is actually $\min(m_{n_j}(b), \min(m_{n_k}(a), m_{n_l}(a)) \oplus \min(m_{n_k}(b), m_{n_l}(b)))$ where the $\oplus$ operator represents the addition of two distributions.

In general, this approximation can arbitrarily inflate estimates of expected delays of packets. In the Appendix, we present an extension, DAG_DELAY, to Algorithm 2 that computes a provably accurate estimate of packet delays assuming that global network state is instantaneously available. We also present an implementation of DAG_DELAY using a more realistic causal channel that builds and maintains a a partial view of global state based on information exchanged in past meetings (Section 4.3).

The computational complexity of estimating delays
using $\text{DAG\_DELAY}$ is significantly higher than Algorithm 2. Ignoring non-vertical edges makes the estimation of Algorithm 2 simple, local, and computationally efficient — our primary design criteria. Our experience suggests that in practice, Algorithm 2 well-approximates the expected delay of packets assuming no further packet replication. We do not formally prove this claim. The evidence of the benign nature of this approximation is in our experimental results based on simulations as well as real deployment. We refer the interested reader to the Appendix for a detailed analysis of the impact of the independence assumption in Algorithm 2.

### 4.1.2 Exponential distributions

We walk through Algorithm 2 for a scenario where nodes meet with a known exponentially distributed meeting times. Few scenarios may have such clean distributions, but we examine this case for expository benefit.

Let the meeting time between nodes be described by a uniform exponential distribution with parameter $\frac{1}{\lambda}$. In the absence of bandwidth restrictions, the expected delivery delay when there are $k$ replicas is the mean meeting time factored by $k$, i.e., $P(a(i) < t) = 1 - e^{-\frac{t}{\lambda}}$ and $A(i) = \frac{t}{\lambda}$.

However, when transfer opportunities are limited, the expected delivery delay depends on the contents of $n_j$’s buffer. From Step 4 of Algorithm 2, let $b_j(i)$ be the size of packets ahead of a packet $i$ in a node’s buffer and let the size of the transfer opportunity available be $B_j$. Then the distribution for meeting the destination $[b_j(i)/B_j]$ times is described by a gamma distribution with mean $\frac{1}{\lambda} \cdot \left[\frac{b_j(i)}{B_j}\right]$.

Given $k$ replicas, the random variable $a(i)$ is described by the minimum of $k$ gamma variables. Calculating an accurate estimate of the expected delivery delay becomes complex because we cannot obtain a closed form expression for $a(i)$. Instead, if we assume that the time taken for a node to meet the destination $b_j(i)/B_j$ times is exponential with mean $\frac{1}{\lambda} \cdot \left[\frac{b_j(i)}{B_j}\right]$. This assumption estimates $a(i)$ as the minimum of $k$ exponential variables. Let

$$s_y(i) = \left[b_y(i)/B_y\right], \ \forall y \in [1, k]$$

Then $A(i)$ is calculated as

$$P(a(i) < t) = 1 - (e^{-\frac{s_1(i)}{\mu_1} + \frac{s_2(i)}{\mu_2} + \ldots + \frac{s_k(i)}{\mu_k}})$$

$$A(i) = \frac{1}{\frac{s_1(i)}{\mu_1} + \frac{s_2(i)}{\mu_2} + \ldots + \frac{s_k(i)}{\mu_k}}$$

When the meeting time between nodes is described by a powerlaw distribution, the result is similar. The meetings times are drawn from an exponential distribution, but the mean meeting times between each pair of nodes is drawn from a powerlaw distribution. Assume that there are $k$ nodes with the copy of the packet and the mean meeting time between the $k$ nodes and the destination is $\mu_1, \mu_2 \ldots \mu_k$. Then $A(i)$ is calculated as

$$P(a(i) < t) = 1 - (e^{-\frac{s_1(i)}{\mu_1} + \frac{s_2(i)}{\mu_2} + \ldots + \frac{s_k(i)}{\mu_k}})$$

$$A(i) = \frac{1}{\frac{s_1(i)}{\mu_1} + \frac{s_2(i)}{\mu_2} + \ldots + \frac{s_k(i)}{\mu_k}}$$

### 4.2 Unknown mobility distributions

To estimate mean inter-node meeting times in the vehicular DTN testbed, every node tabulates the average time to meet every other node based on past meeting times. Nodes exchange this table as part of meta-data exchanges (Section 3.4). A node combines the meta-data into a meeting-time matrix and the information is updated after each transfer opportunity. The $(x, y)$ entry of the meeting-time matrix $M_{x,y}$ is the expected time for $x$ to meet $y$ directly, calculated as the average of past meetings.

Node $n_j$ calculates $m_{n_j}(i)$, the expected time for $n_j$ to meet $i$’s destination, using the meeting-time matrix. $m_{n_j}(i)$ is estimated as the minimum time taken for $n_j$ to meet the destination of $i$ in at most $h$ hops. (Unlike exponential mobility models, some nodes in the trace never meet directly.) For example, if a node $A$ meets another node $C$ via an intermediary $B$, the expected meeting time is $M_{A,B} + M_{B,C}$ for $h = 2$. When two nodes never meet, the inter-meeting time is infinity. In our implementation we restrict $h = 3$. Several DTN routing protocols [3, 21, 5] use similar techniques to estimate meeting probability among peers.

RAPID used this peer meeting times to estimate delivery delay. Since we do not know the meeting time distribution, we make a simplifying assumption and let the meeting times approximate the exponential distribution. Using this assumption and from Eq. 8, the expected delivery delay is

$$m'_{n_j}(i) = \left[b_j(i)/B_j\right] \cdot m_{n_j}(i)$$

$$A(i) = \frac{1}{\sum_{y=1}^{k} m'_{n_y}(i)}$$

The distribution of bus meeting times in the DTN testbed is 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 trace meeting times as an exponential distribution makes delivery delay estimates easy to compute and performs well in practice (Section 6). Moreover, implementing Eq. 11 is simple and incurs low overhead making them practical for extremely resource-constrained DTNs such as wildlife sensornets [18] or OLPC-like environments [1].

### 4.3 Control channel

Previous studies [16] have shown that as nodes have more available information about the global system state
from oracles, they can make better decisions and improve routing performance significantly. We extend this notion for practical DTNs where no oracle is available. To this end, RAPID nodes gather knowledge about the global system state by disseminating information using a small fraction of the transfer opportunity.

RAPID uses a causal channel to exchange acknowledgments for delivered packets as well as meta-data about every packet learnt from past exchanges. To estimate utilities, for each packet encountered causally, RAPID maintains a list of nodes that carry the replica of the packet, and for each replica an estimated time to delivery. Meta-data of delivered packets are deleted when an ack is received.

For efficiency, a RAPID node maintains the time of last meta-data exchange with its peers. The node only sends information about causal packets whose content 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 transfer opportunities in the past;
- Expected meeting times with peers;
- List of packets delivered since last exchange;
- For each packet in its own buffer, 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 causal channel, nodes have only an imperfect view of the system. Information propagated using the causal channel may be stale due to change in number of replicas, changes in delivery delays, or if the packet is delivered and acknowledgments have not propagated. However, we show in Section 6 that this in-accurate information does not prevent RAPID from performing significantly better than related protocols.

We observe in our implementation that for 1 KB packets, the average meta-data exchange is about 1% of transfer opportunities. Packets of 10 KB would incur about a 0.1% overhead overall. We present more detailed measurements of overhead in RAPID in Section 6. The average meta-data exchange per contact is proportional to the size and frequency of transfer opportunities. As more packets are exchanged, meta-data size increases. But as packets are delivered, meta-data about the packet is removed from the system decreasing the total volume.

5. IMPLEMENTATION ON A VEHICULAR DTN TESTBED

We implemented and deployed RAPID on our vehicular DTN testbed consisting of 40 buses, of which a subset is on the road each day. The implementation allowed us to meet the following objectives – (i) the routing infrastructure that we created is the first step in deployment of DTN applications on the testbed (ii) the deployment is subject to some events that are not perfectly modeled in the simulation, including delays caused by sorting and utility comparisons, interruptions and disconnections that can occur at any point in the code, and operating system and file system. Therefore, we use this measured performance to validate our simulation results for RAPID; we use the same simulator in Section 6 to compare RAPID to other protocols.

5.1 Deployment Environment

Each bus 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 discover AP, which our software merges into one connection event.) The connection lasts until the radios are out of range. The buses send messages according to the protocol described in Algorithm 1 in Section 3.3, including the meta-data as described in Section 4.3. The protocol is symmetric and when two buses are connected, they are simultaneously the sender and the receiver.

During the deployment experiments, we generated packets periodically on each bus with an exponential inter-arrival time. The destinations of the 1 KB-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 buses information about the location or route of other buses on the road. We set the packet generation to 4 packets were generated by each node for every other destination per hour; since the number of buses on the road at any time varies, this is the simplest way to express load. For example, that when the load is 4 packets/hour/bus/destination and there are 20 buses on the road, then 1,520 packets are generated 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 transferred 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 in the field.

5.2 Performance of Deployed Rapid

We measured the routing performance of RAPID on the buses from December 18, 2006 until January 8, 2007. The measurements are tabulated in Table 3. We exclude holidays and weekends since almost no buses were on the road, leaving 15 days of experiments. On December 23, the buses operated on a reduced schedule
while our local University was on winter break; thus the last 10 days have fewer buses and transfer opportunities to route packets and also fewer packets generated.

During full schedule days, RAPID delivered 91% of packets with an average delivery delay of about 100 minutes. During a reduced schedule, when the offered load was cut by half and the number of contact opportunities was cut from 100 to 31, the delivery rate is down only to 69% of packets. Delays are lower at 60 minutes, but this may be simply because only packets that traverse shorter paths are being delivered. Though the delivery rate is lower, the experiment shows that DTNs can be used in very sparse vehicular network of just 8 buses. We also note that overhead due to meta-data accounts for less than 0.01% of the total transfer bandwidth and less than 1.1% of the data size during both the reduced and the full schedule.

### 5.3 Trace-driven Simulation vs. Measured Performance

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 experimental results against the 15-days of measurements from the deployment. In the simulator, we generate packets under the same assumptions as the deployment, using the same parameters for an exponentially distributed inter-arrival time.

#### Table 4: Experiment parameters

<table>
<thead>
<tr>
<th>Schedule type</th>
<th>Full</th>
<th>Reduced</th>
</tr>
</thead>
<tbody>
<tr>
<td>Avg buses scheduled per day</td>
<td>19</td>
<td>8</td>
</tr>
<tr>
<td>Avg total bytes transferred per day</td>
<td>2.476 MB</td>
<td>575 MB</td>
</tr>
<tr>
<td>Avg number of meetings per day</td>
<td>101</td>
<td>31</td>
</tr>
<tr>
<td>Avg packets delivered per day</td>
<td>91%</td>
<td>69%</td>
</tr>
<tr>
<td>Average packet delivery delay</td>
<td>101.9 min</td>
<td>60.4 min</td>
</tr>
<tr>
<td>meta-data size / bandwidth</td>
<td>0.0009</td>
<td>0.00064</td>
</tr>
<tr>
<td>meta-data size / data size</td>
<td>0.011</td>
<td>0.004</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Number of nodes</th>
<th>Power law</th>
<th>Trace-driven</th>
</tr>
</thead>
<tbody>
<tr>
<td>40</td>
<td>max of 40</td>
<td>40 GB</td>
</tr>
<tr>
<td>100 KB</td>
<td>100 KB</td>
<td>given by real transfers among buses</td>
</tr>
<tr>
<td>15 min</td>
<td>19 hours each trace</td>
<td></td>
</tr>
<tr>
<td>1 KB</td>
<td>1 KB</td>
<td></td>
</tr>
<tr>
<td>50 sec mean</td>
<td>1 hour</td>
<td></td>
</tr>
<tr>
<td>20 sec</td>
<td>2.7 hours</td>
<td></td>
</tr>
</tbody>
</table>

Fig. 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 simulator. We build other protocols over the simulator and compare routing performance with varying network parameters using 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 using RAPID in three cases: minimize maximum delay, minimize average delay, and minimize missed deadlines. In all cases, we found RAPID performs significantly better than previous work. We also compare the performance of RAPID to an optimal solution.

#### 6.1 Assumptions

Our evaluations are based on a custom event-driven simulator, as described in the previous section. We used traces from our vehicular DTN testbed from September 15, 2007 until December 22, 2007. When excluding weekends and holidays, we collected 65 days of measurements at full schedule. In these traces, buses exchanged as much random data as possible over TCP at each connection opportunity until they are out of wireless range.

In these tests, the meeting times between buses are not known a priori. All values used by RAPID, including average inter-meeting times, are learned during the experiment assuming initial infinite meeting time.

We compare RAPID to four other routing protocols: MaxProp [3], Spray and Wait [31], Random, and Optimal. We also implement PROPHET [21], that uses historical information to calculate meeting probabilities. In all evaluations, RAPID uses a causal channel for exchanging meta-data, and we include the cost of such exchanges.

MaxProp operates in a storage- and bandwidth-con-
strained environments, allows for 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 implement the binary Spray and Wait and set $L = 12$. We implemented PROPHET with parameters $P_{init} = 0.75$, $\beta = 0.25$ and $\gamma = 0.98$ (parameters based on values used in [21]).

We also compare RAPID to Optimal, the optimal routing protocol that provides an upper bound on performance. Optimal is computed using an ILP formulation. The formulation is presented in the appendix.

To model stochastic mobility patterns, we use uniform exponential and power law distributions. Song et al. [30] suggests that meeting times between people in a busy public place follow a uniform exponential model, i.e., pairs of nodes meet each other once every $t$ seconds where $t$ is exponentially distributed with a mean that is common for all pairs of nodes. Other have suggested that a DTN formed among people have been shown to have a skewed, power law inter-meeting time distribution [6, 20]. The default parameters used for the experiments are tabulated in Table 4. The parameters for synthetic mobility model are different from the trace-driven model because the performances from 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 65 traces. Moreover, each of the 65 days of traces 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 is not shown in the graphs for clarity).

### 6.2 Results Based on Testbed Traces

We set this value based on consultation with authors and using LEMMA 4.3 in [31] with $a = 4$.  

#### 6.2.1 Comparison with existing routing protocols

Our experiments show that RAPID consistently outperforms MaxProp, Spray and Wait and Random. We increase the load in the system up to 40 packets per hour per destination until Random delivers less than 50% of the packets. Figure 4 shows the average delay of delivered packets using the four protocols under varied load when RAPID’s routing metric is set to minimize average delay (Eq. 1). On average, when using RAPID packets are delivered 20 minutes (20%) earlier than MaxProp, 30 minutes (25%) earlier than Spray and Wait, and 40 minutes (30%) earlier than Random. In the same experiment, the fraction of packets delivered by RAPID is consistently greater than other protocols (shown in Figure 5). We observe that at load 40, RAPID still delivers 80% of the packets compared to 63% by MaxProp and 50% by Random.

Figure 6 shows results when the metric is set to minimize maximum delay (Eq. 3). and similarly, Figure 7 shows the performance of RAPID when the routing metric is set to maximize the number of packets delivered within deadlines (Eq. 2). The figures show similar trends for all four protocols. RAPID reduces maximum delay by an average of 90 minutes compared to MaxProp, the second-best performing protocol, and similarly, RAPID increases packets delivered within a deadline by an average of 20% compared to MaxProp. We have results for the fraction of packets delivered for these two metrics, but we exclude them due to space restrictions; the results are similar to Fig. 5.

Because each bus takes a different geographic route, standard deviation and similar measures of variance are not appropriate for comparing the mean delays. Accordingly, we performed a paired $t$-test [25] 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 Meta-data Exchange

We allow RAPID to use as much bandwidth at the start of a transfer opportunity as it requires. To see if this approach was wasteful or beneficial, we performed experiments where we limited the total meta-data exchanged as a fixed fraction of the transfer opportunity.

Figure 8 shows the average delay performance of RAPID when meta-data is limited as a percentage of the total data transferred. The results show that performance increases as the limitation is removed, and the best performance results when there is no restriction on meta-data at all. Particularly, we observe that the performance of RAPID with complete meta-data exchange improves by 20% compared to when no meta-data is exchanged.

Fig. 9 shows the trade-off between meta-data as a percentage of total data transferred and the percentage of total data transferred and the percentage of data delivered. The results show that performance increases as the limitation is removed, and the best performance results when there is no restriction on meta-data at all. Particularly, we observe that the performance of RAPID with complete meta-data exchange improves by 20% compared to when no meta-data is exchanged.

6.2.3 Evaluation of RAPID components

We increase the load to 75 packets per destination per hour. The bandwidth utilization of the system increases from 5% to 30% when load is increased from 5 to 75 packets per hour per destination. The available bandwidth varies significantly across transfer opportunities in our bus traces. Small bandwidth opportunities cause bottlenecks along a path, resulting in a low utilization. When the load is small, the bandwidth utilization is small and delivery rate is high. For a small load, meta-data accounts for about 1% of the data bandwidth. But as the load in the system increases, more packets are exchanged but delivery rate decreases. Hence, when the load is is 75, we observe the the meta-data cost is 5% of the data bandwidth.
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; (ii) RAPID-LOCAL: Using RAPID metrics to exchange packets but peers exchange meta-data about only packets in their own buffers; (iv) RAPID.

Figure 13 shows performance of the different components of RAPID when the routing metric is average delay. From the figure we observe that using acknowledgments alone improves performance by an average of 8%. The authors of Maxprop [3] also show empirically that propagating acknowledgments clears buffers, avoids exchange of already delivered packets, improving performance. RAPID-LOCAL provides a further improvement of 10% on average even though meta-data exchange is restricted to packets in the peer’s local buffer. But allowing all meta-data to flow further improves the performance by about 11%.

6.2.4 Fairness

Figure 14 presents the results of a fairness experiment on RAPID. In this experiment, we generate 20 or 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 \( J = \frac{d_i^2}{n \cdot \left( \sum d_i \right)^2} \), 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. From Figure 14 we observe that the fairness index of over 98% of the packets is 1 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 delay.

6.2.5 Hybrid DTN with thin continuous connectivity

In this section, we measure the performance of RAPID using an instant, unlimited bandwidth global channel for exchange information. These results show us how inaccurate meta-data reduces the performance of RAPID. One interpretation of the global channel is the use of RAPID as a hybrid DTN where all control traffic went over a low bandwidth, long-range radio, such as WiMax or XTEND. A hybrid DTN will use a high-cost low bandwidth channel for control whenever available and low-cost high bandwidth channel to send data.

Figure 10 shows the average delay performance of RAPID when using a causal channel in comparison to a global channel. We observe that the average delay decreases by 20 minutes when using a global channel for high delays. Similarly, from Fig. 11 we observe that the percentage packets delivered within a deadline increases by an average of 15% using a global channel.

This observation suggests that performance improvements to RAPID with a causal channel can come from allowing control information to propagate more easily. Since control information is small compared to data, RAPID can make efficient use of a low-bandwidth, long-range radio to propagate control information, such as WiMax or XTEND.

6.2.6 Comparison with Optimal

In our final trace experiment, we compare RAPID to Optimal which is an upper bound on the performance. We have assumed in experiments above that RAPID has used a limited, causal channel between for both causal channels. 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 does not use replication when there are no failures in the system, propagation delay of all links are equal, and when node meetings are known in advance. We present a formulation of this problem in the Appendix. Our evaluations use the CPLEX solver [8] to show specific results for the Optimal algorithm. Because the solver grows in complexity with the number of packets, these simulations are limited to only 6 packets per hour per destination which is why we present them separately. Jain et al. [16] solve a more general DTN routing problem by allowing packets to be fragmented across links and assigning non-zero propagation delays on the links. This limited the size of the network they could evaluate.

Figs. 12 presents the average delay performance of Optimal, RAPID, and MaxProp. 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 delay of undelivered packets when presenting the results for RAPID and MaxProp. We observe that for small loads, the performance of RAPID using the causal channel is within 10% of the optimum performance, while MaxProp is greater than 28% from the optimal performance. RAPID using global channel performs within 6% of optimal.

6.3 Results from synthetic mobility models

In this section, we use synthetic mobility models to compare the performance of RAPID to MaxProp, Random, and Spray and Wait. We set the mean of exponentially distributed meeting times to 100 sec, and we set the exponent of power law meeting times to 0.3 sec.

Figure 15 shows the maximum delay of packets using the four routing protocols when the intermeeting times are powerlaw and the load is varied (i.e., RAPID is set to use Eq. 3 as a metric). RAPID reduces maximum delay by up to 30% compared to MaxProp, which is the next-best performing protocol. The reason MaxProp
performs worse is that it prioritizes new packets; older, undelivered packets won’t see service as load increases, which increases maximum delay in the system.

Figure 16 show the performance of the different routing protocols with respect to maximizing the number of packet delivered within an average deadline of 20 sec. From Fig. 16 we observe that the percentage packets delivered is improved by 20% over MaxProp. This experiment shows that an intentional routing protocol can achieve a better allocation of resources even in a completely decentralized setting.

We observe similar trends when the mobility model is assumed to be uniform exponential. Figure 17 shows that the average delay of packets is reduced by about 25% when using RAPID over MaxProp. Similarly, Figure 18 shows that the max delay of packets is reduced by up to 50% when using RAPID. We also observe that when the metric is to minimize average delay, MaxProp and Spray and Wait have comparable performance to Random for high loads, while RAPID maintains lower delays.

6.3.1 Storage constraint

Figure 20 shows how constrained buffers varied from 10 KB to 280 KB affect the delivery deadline metric (for power law meeting times and fixed load of 20 packets per node per destination every 50 sec). RAPID is able to best manage limited buffers to deliver packets within a deadline of 20 sec and delivers up to 20% more packets within deadline compared to MaxProp even when storage is restricted.

Figure 19 shows the performance of the four protocol with respect to the max delay metric. RAPID achieves about 35% reduction in delay compared to MaxProp, and about 50% reduction in delay compared to Spray and Wait. We note that all protocols perform poorly when the buffer size is small.

Similar trends were observed for other combinations of routing metric and synthetic mobility pattern (not shown in the graphs), both for increasing load and decreasing buffer size.

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 [19] 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 [19], 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 metric. 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 heuristic, 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.

8. REFERENCES

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 \( t_e \) is the time of the meeting and \( s_e \) 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

Let \( \text{ALG} \) be any deterministic online DTN routing algorithm with unlimited computational power.

**Theorem 1(a).** If \( \text{ALG} \) has complete knowledge of the workload, but not of the schedule of node meetings, then \( \text{ALG} \) is \( \Omega(n) \)-competitive with an offline adversary.

**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. 21, 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 ALG replicates packet \( p_i \). Let \( X(p_i) \) be the set of intermediate nodes to which ALG delivers packet \( p_i \). Let \( D(p_i) \) denote the set of nodes to which ALG delivers packet \( p_i \).

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.
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. \qed

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 22(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 $p_3$ 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 $p_1, p_2, p'_1, p_3$ 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).

\section*{B. COMPUTATIONAL HARDNESS OF THE DTN ROUTING PROBLEM}

\textbf{Theorem 2}: Given complete knowledge of node meetings and the packet workload \textit{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. \medskip

\textbf{Proof}. Consider a DTN routing problem with $n$ nodes that have complete knowledge of node meetings and workload \textit{a priori}. The input to the DTN problem is the set of nodes $1, \ldots, n$; a series of transfer opportunities $\{(u_1, v_1, s_1, t_1), (u_2, v_2, s_2, t_2), \ldots\}$ such that
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 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 [7].

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 $\{(s_1, t_1), (s_2, t_2), ..., (s_n, t_n)\}$, do a set of edge-disjoint paths $\{e_1, e_2, ..., e_k\}$ exist, such that $e_i$ 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 $(u, v, l(e))$, assuming transfer opportunities are unit-sized. Source and destination pairs $\{(s_1, t_1), (s_2, t_2), ..., (s_n, t_n)\}$ are mapped to packets $\{p_1, p_2, ..., p_n\}$, 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 corollary follows from the lower bound of $\Omega(n^{1/2-\epsilon})$ on the approximation ratio of the EDP problem [13] and the above reduction.

Hence, Theorem 2.

### C. DELAY ESTIMATION BASED ON DEPENDENCY GRAPHS

In Section 3.3, we presented Algorithm 2 that estimates expected delays of packets based on their position in node buffers. The algorithm is a simple, local, and computationally efficient heuristic to estimate expected delays. However, it 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, and then we present an algorithm, DAG\_DELAY, based on RAPID’s causal channel to approximate these expected delays with a partial view of global state.

To understand the simplifying assumption in Algorithm 2, 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 $V_i = \{x_{i,1}, x_{i,2}, ..., x_{i,k_i}\}$ is the set of $k_i$ replicas of packet $i$. All packets in $V$ are destined to the same DTN node — 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.

An example is the graph shown in Figure 23.

![Packet positions in the buffer and dependency graph](image)

**Figure 23:** An example dependency graph. All packets have the same destination. Nodes correspond to packet replicas and represent their delay distribution. Edges represent dependencies between delay distributions.

**Lemma 6.** \( G \) is a directed, acyclic graph (DAG) and there is no path from a replica of any packet \( i \) to itself.

**Proof.** Let the position of packet \( i \) in the buffer of a DTN node \( n_x \) be denoted by \( pos_x(i) \). The packet is in position 1 if it is in the top of the buffer. We first prove the following claim.

\[
\begin{align*}
d(a) &= \min(e_k, e_l) \\
d(b) &= \min(e_j, d(a) \oplus e_i, d(a) \oplus e_l) \\
d(c) &= e_i \oplus d(b) \\
d(d) &= \min(e_j \oplus d(b), e_k \oplus d_i, e_i \oplus d(a))
\end{align*}
\]

**Figure 24:** A topologically sorted dependancy graph.

**Claim.** If \( pos_y(i) < pos_y(j) \), then \( pos_y(i) < pos_y(j) \) at any DTN node \( n_y \) where replicas of both \( i \) and \( j \) exist.

**Proof.** From Algorithm 2, the position of the packets \( i \) and \( j \) is determined as \( m_{n_y}(i) + T(i) \) and \( m_{n_y}(j) + T(j) \) respectively by \( n_x \). The creation time of a packet is same across all of its replicas. Also, since \( i \) and \( j \) are destined to the same node, \( m_{n_y}(i) = m_{n_y}(j) \). Therefore if \( (m_{n_y}(i) + T(i)) < (m_{n_y}(j) + T(j)) \), then \( (m_{n_y}(i) + T(i)) < (m_{n_y}(j) + T(j)) \).

**Corollary 3.** The creation timestamps of packets along the DAG are in nondecreasing order.

We prove the lemma itself by contradiction. The construction of \( G \) ensures that a packet \( i \) in buffer \( n_x \) is directly dependent on its successor \( j \) and all replicas of \( j \). Assume that there is a cycle in graph \( G \) involving node \( i \). Then, there exist dependencies such as (w.l.o.g) \( (i \rightarrow j, j \rightarrow k, k \rightarrow i) \), where \( i \rightarrow j \) means there is an edge from some replica of \( i \) to a replica of \( j \). This implies that \( pos_y(i) > pos_y(j) \), \( pos_y(j) > pos_y(k) \), and \( pos_y(k) > pos_y(i) \) for some nodes \( n_y, n_x \) and \( n_z \). This contradicts the claim above. Hence Lemma 6.

**C.1 An idealized global algorithm**

Next, we present an idealized algorithm to compute expected delays of packets given the complete dependency graph. We call this algorithm idealized because its implementation requires a global control channel such as that introduced in section 6.2.5.

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). We topologically sort the dependency graph. The topological sort of the example dependency graph in Fig 23 is shown in Fig 24. \( \text{dag\_delay} \) computes the delay of the packets in the graph in the topologically order starting from the root. The information maintained for each of the \( k \) replicas \( p_1, \ldots, p_k \) of packet \( p \) is \( \{ \text{succ}(p_j), e_{\text{node}(p_j)} \} \), \( 1 \leq j \leq k \), where \( \text{succ}(p_j) \) is the successor of the replica \( p_j \) and \( \text{node}(p_j) \) is the DTN node buffer where \( p_j \) exists.

**Procedure \( \text{dag\_delay}(p) \):**

1. for each replica \( p_j, 1 \leq j \leq k \) of \( p \), do

   (a) Let \( s = \text{succ}(p_j) \), and \( n = \text{node}(p_j) \)

   (b) if \( d(s) \) is not defined, then

      i. \( d(s) = \text{dag\_delay}(s) \)

   (c) \( d'(p_j) = d(s) \oplus e_n \)

2. return \( d(p) = \min(d'(p_1), \ldots, d'(p_k)) \)

**Fig. 24** presents the delay of each packet as computed using \( \text{dag\_delay} \). 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, \( \text{dag\_delay} \) will converge.

**C.2 A causal channel based implementation**

The \( \text{dag\_delay} \) algorithm described above assumes a global channel that instantaneously broadcasts information about packet replication to all DTN nodes. However, in practice, a global channel may not be available
in a DTN. In this section, we describe an implementation of DAG\_DELAY based on a causal channel that may only have a partial view of global state.

First, consider how one may implement DAG\_DELAY above using a global channel. At time 0, each DTN node constructs a DAG using packets in its buffer. The DAG is updated on replication of a packet \( p \) to node \( x \) (or creation of \( p \) at node \( x \)) as follows:

1. Place \( p \) in the buffer and sorted by creation timestamps of packets.
2. Flood information about \( e_x \) and the successor of \( p \) in \( x \)'s buffer to all other nodes.
3. At each node, update DAG by adding edges from \( p \) to \( \text{succ}(p) \) and record \( e_x \).
4. At each node, re-calculate the delay for all packets that are below \( p \) in the topological ordering.

The implementation can be optimized by updating the DAG for all events (packet creation, replication, and delivery) during a transfer opportunity in a batched manner and subsequently estimating the delays using DAG\_DELAY.

A causal channel replaces step 2 above with pairwise exchanges of global state information. Thus, a causal channel may have an incomplete view of global state. However, the causal approximation of the DAG preserves the topological ordering of packets in the system in the following sense. Let \( D' \) be the DAG maintained by a causal channel with stale information and \( D \) be the DAG maintained by a global channel. We say that packet \( p \) depends on packet \( q \) if there is a path from some replica of \( p \) to some replica of \( q \).

**Lemma 7.** \( D' \) is an order-preserving approximation of \( D \) in the following sense: for any two packets \( p \) and \( q \), it cannot be the case that \( p \) depends on \( q \) in \( D' \) and \( q \) depends on \( p \) in \( D \).

**Proof.** A causal channel obtains only a partial, stale view of global state — it may not know about the creation, replication, and removal (upon delivery) of some packets in the system. By construction, the creation of a packet does not affect the delays of existing packets in the DAG as it introduces only outgoing edges from itself.

Replicating a packet can affect the delays of existing packets by introducing incoming edges, however, it can not invert the dependency between an existing pair of packets. We show this by contradiction. Suppose \( p \) depends on \( q \) before the replication of a packet \( r \), and \( q \) depends on \( p \) just after. By Corollary 3, \( p \) and \( q \) must have the same creation timestamp as the replication of packet \( r \) has no effect on the creation timestamps of existing packets. Furthermore, the new replica \( r \) must lie on a path from \( q \) to \( p \) as there was no such path before the replication. This means that \( r \) has the same creation timestamp as \( p \) and \( q \), a contradiction.

Removing a packet upon delivery can not create a dependency that did not already exist as the removal operation only deletes edges from the DAG. Hence, the lemma.

**Corollary 4.** \( D' \) eventually converges to \( D \) assuming no further packet replication if all DTN nodes meet all other DTN nodes directly or indirectly.

Note that Lemma 7 does not imply that DAG\_DELAY estimates expected delays in an order-preserving manner, i.e., if the expected delay \( d_i \) for packet \( i \) is less than the expected delay \( d_j \) of packet \( j \) in the DAG, then DAG\_DELAY with a causal channel may estimate the delay \( d'_i \) of \( i \) to be greater than the delay \( d'_j \) of \( j \). In fact, we have been able to construct pathological scenarios where the estimation error is arbitrarily large.

### C.3 Comparing Algorithm 2 and DAG\_DELAY

Our implementation of rapid uses Algorithm 2 instead of DAG\_DELAY to estimate expected delays. Algorithm 2 estimates expected delays by ignoring non-vertical edges in the dependency DAG. Although ignoring these dependencies can arbitrarily inflate delay estimates in some pathological scenarios and we find that in practice Algorithm 2 performs close to optimal for small to moderate workloads for which we were able to compute the optimal. We also note that in practice, DAG\_DELAY is computationally heavyweight even when a global channel is available. For every replication, delays need to be recomputed and propagated down the DAG. In contrast, Algorithm 2 makes the estimation simple, local, and computationally efficient, in line with our primary design goal of developing a practical DTN routing algorithm.

### D. ILP FORMULATION:

We divide time into discrete intervals so every node meets at most one other node in an interval. Jain et al. [16] 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.

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. 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 bandwidth for each meeting is a constant and is $B$.

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.

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

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 $s(t(p)) = n$ and $c(p) = i$ $\forall p$

**Bandwidth constraint**

$$
\sum_{p \in P} (X(p, e) \cdot \text{size}(p)) \leq B \forall e
$$

**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**

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

**Delivery Constraint**

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