Routing in a Delay Tolerant Network

Sushant Jain
University of Washington
sushjain@cs.washington.edu

Kevin Fall
Intel Research, Berkeley
kfall@intel.com

Rabin Patra
University of California, Berkeley
rkpatra@cs.berkeley.edu

ABSTRACT

We formulate the delay-tolerant networking routing problem, where messages are to be moved end-to-end across a connectivity graph that is time-varying but whose dynamics may be known in advance. The problem has the added constraints of finite buffers at each node and the general property that no contemporaneous end-to-end path may ever exist. This situation limits the applicability of traditional routing approaches that tend to treat outages as failures and seek to find an existing end-to-end path. We propose a framework for evaluating routing algorithms in such environments. We then develop several algorithms and use simulations to compare their performance with respect to the amount of knowledge they require about network topology. We find that, as expected, the algorithms using the least knowledge tend to perform poorly. We also find that with limited additional knowledge, far less than complete global knowledge, efficient algorithms can be constructed for routing in such environments. To the best of our knowledge this is the first such investigation of routing issues in DTNs.

Categories and Subject Descriptors:
C.2.2: Routing Protocols
General Terms: Algorithms, Performance
Keywords: Routing, Delay Tolerant Network

1. INTRODUCTION

In this work, we look at the problem of routing in a *delay tolerant network* (DTN)[8]. Such networks are assumed to experience frequent, long-duration partitioning and may never have an end-to-end contemporaneous path. This problem contrasts with routing in conventional data networks which typically selects a shortest policy-compliant path in a connected graph without considering availability of intermediate buffering and bandwidth capacity.

In graph theoretic terms, our problem is a form of the "quickest transshipment problem" in which both edge capacities and transit delays along an edge can vary (down to zero) as a function of time and nodes have finite buffers [12]. In practical terms, DTNs arise in networks with known connectivity patterns such as Low-Earth Orbiting Satellites (LEO) or those with unpredicted, opportunistic connectivity (e.g., communication among PDAs when brought into close proximity [5]). Here, we focus on the former case.

The routing problem in a DTN may at first appear as the standard problem of dynamic routing but with extended link failure times. This is not the case. For the standard dynamic routing problem, the topology is assumed to be connected (or partitioned for very short intervals), and the objective of the routing algorithm is to find the best currently-available path to move traffic end-to-end. In a DTN, however, an end-to-end path may be unavailable at all times; routing is performed *over time* to achieve *eventual delivery* by employing long-term storage at the intermediate nodes. The DTN routing problem amounts to a constrained optimization problem where edges may be unavailable for extended periods of time and a storage constraint exists at each node. This formulation reveals DTN routing to be a considerably different and more challenging problem.

In this paper, we make several contributions: we first motivate and formulate the DTN routing problem when the connectivity patterns are known, then provide a framework for evaluating various routing algorithms, and finally show a simulation-based comparison of several of our own algorithms. We also include an optimal algorithm based on a linear programming approach to serve as a basis for comparison with the simulations. Finally, we outline the future work to be accomplished in the area.

2. EXAMPLE: CONNECTING A REMOTE VILLAGE

The problem of providing data communications to remote and rural areas is beginning to attract the attention of the computer systems research community [23]. While many rural connectivity projects involve attempts to provide conventional Internet access to remote areas, a small number of projects are taking an alternative approach which focuses on asynchronous messaging in order to greatly reduce the cost of connectivity [25, 20, 23]. For example, the Wizzy Digital Courier service provides asynchronous (disconnected) Internet access to schools in remote villages of South Africa [25]. In this system, a courier on a motorbike, equipped with a USB storage device, travels from a village school to a large city which has permanent (reasonably high-speed) Internet connectivity. Typically, it takes a few hours for the courier to travel from the village to the city.

In consideration of this scenario, we realize that several other connectivity options may be available (e.g. satellites, either LEO or GEO, possibly telephone), but are not likely to be cost

SIGCOMM'04, Aug. 30–Sept. 3, 2004, Portland, Oregon, USA.

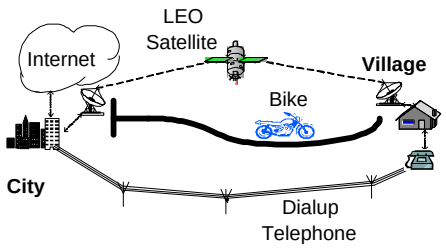


Figure 1: Scenario illustrating a variety of connectivity options between a remote village and a city. Even in this simple scenario, many route choices are possible.

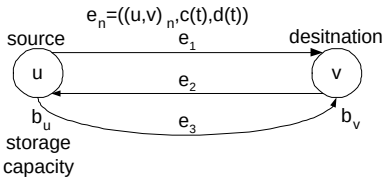


Figure 2: Edges in a DTN graph. Nodes may be connected by multiple edges, representing different physical links. Each node j performs store-and-forward routing, and has finite storage capacity (b_j). An edge is parameterized by its source and destination nodes plus a capacity ($c(t)$) and delay function ($d(t)$).

effective or of sufficient capacity to handle all of the traffic. Conversely, for some traffic, such as high-priority alerts, low latency may be sufficiently important to justify using a higher-cost communication system offering smaller delay. Thus, we consider a simple extended scenario, based on this real-world example, that motivates the DTN routing problem.

Figure 1 shows a hypothetical village served by a digital courier, a wired dialup Internet connection, and a store-and-forward LEO satellite (e.g. PACSAT). These satellites have low to moderate bandwidth (around 10 Kbps) and are visible for 4-5 short periods of time ("passes") per day (lasting around 10 minutes per pass, depending on the orbit inclination and location on Earth). We call the opportunity to communicate a *contact* (as in [8]), which is characterized by a duration of time, a capacity, and a propagation delay (assumed to remain constant during the contact duration). In addition, depending on the type of connection used, buffering constraints may also need to be considered.[1] The digital courier service represents a high-bandwidth, high-latency contact, the dialup represents a low-bandwidth, low-latency contact, and the LEO satellite represents a moderate-bandwidth, moderate-latency contact. The problem of selecting which contacts to carry messages and when represents an instance of the DTN routing problem. Route selection may depend on a variety of factors including message source and destination, size, time of request, available contacts, traffic in the system, or other factors (e.g. cost, delay, etc.).

In the next sections we develop a set of definitions and a framework for evaluating DTN routing algorithms. We then propose several of our own routing algorithms and use the framework in conjunction with simulations to evaluate their performance in the context of this village scenario.

3. DTN NETWORK MODEL

Nodes and Edges The DTN graph is a directed multi-graph, in which more than one edge (also called link) may exist between a pair of nodes (see Figure 2). The reason for using a multigraph is straightforward: it may be possible to select between two distinct (physical) connection types to move data between the same pair of nodes. Furthermore, the link capacities (and to a lesser extent, propagation delay) are *time-dependent* (capacity is zero at times when the link is unavailable). Thus, the set of edges in the graph must capture both time-varying capacity and propagation delay as well as multiple parallel edges.

A simple example of an edge captured by this description involves a ground station and a LEO satellite rising, passing directly overhead, and setting at the opposite horizon. As it rises, its channel capacity will generally increase until it is directly overhead and will decrease for the remaining time of the pass. This is because noise is minimal when the satellite is directly overhead but increases at lower elevations. Another example would be a bus (carrying a wireless access point) passing by a village. The throughput of the wireless link would depend upon the distance of the bus from the village. When no communication is possible, the edge is assigned zero capacity.

Contact A contact is an opportunity to send data over an edge. More precisely, it is a specific edge and a corresponding time interval during which the edge capacity is strictly positive.

Messages Communication demands are represented by messages. A message is a tuple (u, v, t, m), where u is the source of the message, v is the destination, t is the time at which the message is injected into the system and m is its size (messages can be of arbitrary size). The set of all messages is called the *traffic demand*.

Storage The nodes in a DTN have finite long-term storage (buffers) used for holding in-transit data or data waiting to be consumed by the application at a destination node. In our model, the storage is exclusively used for holding in-transit data. Destination nodes are assumed to have sufficient capacity for holding data to be consumed by an application.

Routing Routing occurs in a store and forward fashion. The routing algorithm is responsible for determining the next edge(s) that a message should be forwarded along. Messages not immediately forwarded wait until they are assigned to contacts by the routing algorithm.

4. DTN ROUTING ISSUES

In this section, we consider a number of important issues in any routing algorithm: the routing objective, the amount of knowledge about the network required by the scheme, when routes are computed, the use of multiple paths, and the use of source routing. We focus on how these issues arise in the context of the DTN routing problem.

4.1 Routing Objective

The routing objective of traditional routing schemes has been to select a path which minimizes some simple metric (e.g. the number of hops). For DTN networks, however, the most desirable objective is not immediately obvious.

One natural objective is to maximize the probability of message delivery. Messages could potentially be lost due to creation of a routing loop or the forced discarding of data when buffers are exhausted. As an approximation, we focus on minimizing the delay of a message (the time between when it is injected and when it is completely received).

[1]The PACSAT satellite systems have limited file storage.

While DTN applications are expected to be tolerant of delay, this does not mean that they would not benefit from decreased delay. Furthermore, we believe this metric is an appropriate measure to use in exploring the differential evaluation of several routing algorithms in an application-independent manner. Minimizing delay lowers the time messages spend in the network, reducing contention for resources (in a qualitative sense). Therefore, lowering delay indirectly improves the probability of message delivery. This is validated by our simulation results.

4.2 Proactive Routing vs. Reactive Routing

In *proactive* routing, routes are computed automatically and independently of traffic arrivals. Most Internet standard routing protocols and some ad-hoc protocols such as DSDV (Destination Sequenced Distance Vector) and OLSR (Optimized Link-State Routing) are examples of this style [4]. In a DTN, these protocols are capable of computing routes for a connected subgraph of the overall DTN topology graph. They fail when asked to provide paths to nodes which are not currently reachable. Despite this drawback, proactive network-layer routing protocols may provide useful input to DTN routing algorithm by providing the set of currently-reachable nodes from which DTN routing may select preferred next hops.

In *reactive* routing, routes are discovered on-demand when traffic must be delivered to an unknown destination. Ad-hoc routing protocols such as AODV (Ad-hoc On-demand Distance Vector) and DSR (Dynamic Source Routing) are examples of this style [4]. In these systems, a route discovery protocol is employed to determine routes to destinations on-demand, incurring additional delay. These protocols work best when communication patterns are relatively sparse. For a DTN, as with the proactive protocols, these protocols work only for finding routes in a connected subgraph of the overall DTN routing graph. However, they fail in a different way than the proactive protocols. In particular, they will simply fail to return a successful route (from a lack of response), whereas the proactive protocols can potentially fail more quickly (by determining that the requested destination is not presently reachable).

In a DTN, routes may vary with time in predictable ways and can be precomputed using knowledge about future topology dynamics. Employing a proactive approach would likely involve computing several sets of routes and indexing them by time. The associated resource requirements would be prohibitive unless the traffic demand is large and a large percentage of the possible network nodes exchange traffic. Otherwise, a reactive approach would be more attractive.

A related issue is *route stability*, a measure of how long the currently-known routes are valid. Route stability depends on the rate of topological change. With relatively stable routes one can employ route caching to avoid unnecessary routing protocol exchanges. With future knowledge about topology changes, caching could be especially effective in a DTN because it may be possible to know ahead of time exactly when to evict existing cached route entries.

4.3 Source Routing vs Per-hop Routing

In *source routing* the complete path of a message is determined at the source node, and encoded in some way in the message. The route is therefore determined once and does not change as the message traverses the network. In contrast, in *per-hop routing* the next-hop of a message is determined at each hop along its forwarding path. Per-hop routing allows a message to utilize local information about available contacts and queues at each hop, which is typically unavailable at the source. Thus, per-hop routing may lead to better performance. Unfortunately, due to its local nature, it may lead to loops when nodes have different topological views (e.g. due to incomplete or delayed routing information).

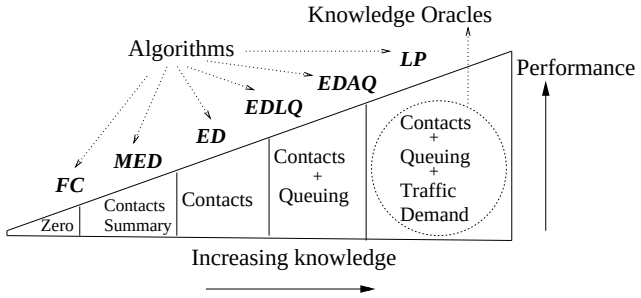


Figure 3: Conceptual performance vs knowledge trade-off. The x-axis depicts the amount of knowledge (increasing in the positive direction). The y-axis depicts the expected performance that can be achieved using a certain amount of knowledge. The figure shows that more knowledge is required to attain better performance. Labels on top show algorithms developed in this paper using the corresponding oracles.

4.4 Message Splitting

A message is *split* when forwarded in such a way that different parts (fragments) are routed along different paths (or across different contacts on the same path). This technique may reduce the delay or improve load balancing among multiple links. It is particularly relevant in DTNs because messages can be arbitrarily large and may not fit in a single contact. However, splitting complicates routing because, in addition to determining the sizes of the fragments, we also have to determine corresponding paths for the fragments.

5. ROUTING EVALUATION FRAMEWORK

5.1 Knowledge Oracles

The DTN routing problem has many input variables such as dynamic topology characteristics and traffic demand. Complete knowledge of these variables facilitates the computation of optimal routes. However, with partial knowledge, the ability to compute optimal routes is hampered, and the performance of the resultant routing is expected to be inferior. To understand this fundamental trade-off between performance and knowledge, we create a set of abstract *knowledge oracles*, each able to answer questions we ask of them. These oracles are notational elements used to encapsulate particular knowledge about the network required by different algorithms.

A key objective of our study is to understand the relationship between algorithm performance and the use of these oracles. Figure 3 illustrates this conceptually by showing the expected performance and oracle requirements for each proposed routing algorithm:

Contacts Summary Oracle This oracle can answer questions about aggregate statistics of the contacts. In particular, the contacts summary oracle provides the average waiting time until the next contact for an edge. Thus, the contacts summary oracle can only respond with *time-invariant* or *summary* characteristics about contacts.

Contacts Oracle This oracle can answer any question regarding contacts between two nodes at any point in time. This is

equivalent to knowing the time-varying DTN multi-graph. The contacts summary oracle can be constructed using the contacts oracle, but not vice versa.

Queuing Oracle This oracle gives information about instantaneous buffer occupancies (queuing) at any node at any time, and can be used to route around congested nodes. Unlike the other oracles, the queuing oracle is affected by both new messages arriving in the system and the choices made by the routing algorithm itself. We expect it to be the most difficult oracle to realize in a distributed system.

Traffic Demand Oracle This oracle can answer any question regarding the present or future traffic demand. It is able to provide the set of messages injected into the system at any time.

5.2 Routing Algorithm Classes

We now present different routing algorithms (Table 1 gives an overview). They fall into the following three classes, based upon the amount of knowledge they need to compute routes.

Zero knowledge These algorithms do not utilize any oracles. Not surprisingly, they perform poorly. They define the minimal extreme of the knowledge-performance relationship in Figure 3.

Complete knowledge This class consists of algorithms that utilize all the oracles (contacts, queuing and traffic demand). The Linear Programming formulation of the DTN routing problem falls into this category and is discussed in Section 8. While these assumptions are far too strong to operate in a widely distributed, dynamic routing environment envisioned by DTNs, we believe this to be the first full formulation of the problem and is, therefore, an important step to establish a deeper understanding and baseline for performance comparisons.

Partial knowledge These algorithms route in the absence of the traffic demand oracle and use one or more of the other oracles (congestion, queuing). Messages are routed independently of the future traffic demand. This is a more practical assumption from an implementation perspective. Therefore, we devote most of our attention to algorithms in this class.

6. ROUTING WITH ZERO KNOWLEDGE

Algorithms in this class route with almost no assistance from any knowledge oracle. We explore one very simple such algorithm that in effect routes randomly using any available contact. Its main purpose is to provide one extreme point in the knowledge-performance relationship space of Figure 3.

Algorithm: **First Contact (FC)**

Oracles: None

A message is forwarded along an edge chosen randomly among all the current contacts. If all edges are currently unavailable, the message waits for an edge to become available and is assigned to the first available contact.

Properties FC performs poorly in nontrivial topologies because the chosen next-hop is essentially random and forwarding along the selected edge may not make any progress toward the destination. A message may also oscillate forever among a set of nodes (especially when frequent contacts are present among a small set of nodes) or be delivered to a dead end. It has no provision to route around congestion. Clearly, FC requires only local knowledge about the network and is trivial to implement.

Improvements The basic approach can be enhanced in many ways. One is to incorporate a sense of trajectory between the source and the destination so that the message is routed in a direction closer to the destination [17]. To prevent loops, a path vector type of approach can be used.

7. ROUTING WITH PARTIAL KNOWLEDGE

The algorithms in this category compute paths using one or more of the following oracles: contacts summary, contacts, and queuing. Further, each message is routed independently of the future demand because the traffic oracle is not used. These algorithms are all based upon assigning *costs* to edges and computing a form of minimum-cost ("shortest") path. Costs are assigned to edges (by consulting the available oracles) to reflect the estimated delay of the message in taking that edge. The challenge and sophistication lies in assigning costs such that the assigned costs are close to the delay that will actually be encountered when a message is forwarded across the DTN.

The reasons for considering only cost-based algorithms in this class are two-fold. First, they provide a convenient and common way to utilize the different knowledge oracles (thereby, identifying to what extent global knowledge is necessary). Second, they correspond naturally to traditional shortest-path based routing problems which are well-understood and for which simple computationally-efficient distributed algorithms are known. This simplicity, however, comes at the price of imposing certain restrictions on the nature of routing paths determined. One key limitation is that only a single path to a destination is derived. As argued earlier, for DTNs it may be important to use multiple paths (with splitting) to achieve near-optimal performance. Interestingly, the basic ideas introduced here can be used to find multiple routes and good split sizes. This is discussed briefly at the end of this section.

7.1 Computing Shortest (minimum cost) Paths

To model the forwarding delay of a message in a DTN, we consider three delay components: 1) queuing time: time until a contact becomes available, 2) transmission delay: time to inject a message completely into an edge, and 3) propagation delay. Queuing time includes both the time waiting for an edge to become available (waiting time) plus the time to drain messages already scheduled for departure on that edge. Queuing time can be large because edges may be unavailable for long periods of time. Given that edge capacities and propagation delays vary with time, we also expect route selection to vary with time.

When edge costs are time-invariant, shortest paths can be computed using Dijkstra's shortest path algorithm. However, if the costs are changing with time the straightforward approach does not work. We must make two modifications to overcome this problem. First, the time a message will arrive at a particular node must be predicted. Second, the predicted arrival time must be used to determine the cost of taking subsequent edges. This would, in turn, affect the time the message arrives at neighboring nodes. Interestingly, Dijkstra's algorithm can be adapted to compute the shortest paths for this case. Pseudocode for the modified algorithm is given in Algorithm 7.1.

The key difference between this algorithm and the traditional Dijkstra's algorithm is the definition and the use of the w (cost) function. It takes into account the time a message arrives at a node. This time is then used to compute the cost of traversing edges emanating from that node (lines 7 and 8 of Algorithm 7.1).

The modified Dijkstra's algorithm requires the cost function for all edges to have the *FIFO property*. This property ensures that a message can not arrive earlier at the destination of an edge by simply waiting longer at the source of the edge (i.e. you will not travel more quickly over an edge if you wait to use it). Formally, it means that for all edges e and all pairs of time t_1, t_2 with $t_1 < t_2$, $w(t_1, e) + t_1 \leq w(t_2, e) + t_2$. When considering

Abbr.	Name	Description	Oracles Used
FC	First Contact	Use any available contact	None
MED	Minimum Expected Delay	Dijkstra with time-invariant edge costs based on average edge waiting time	Contacts Summary
ED	Earliest Delivery	Modified Dijkstra with time-varying cost function based on waiting time	Contacts
EDLQ	Earliest Delivery with Local Queue	ED with cost function incorporating local queuing	Contacts
EDAQ	Earliest Delivery with All Queue	ED with cost function incorporating queuing information at all nodes and using reservations	Contacts and Queuing
LP	Linear Program	-	Contacts, Queuing and Traffic

Table 1: Overview of different routing algorithms. All Dijkstra-based algorithms incorporate a cost function sensitive to edge propagation and transmission delays. Costs are ascertained by consulting the respective oracles.

```
Input: G = (V, E), s, T, w(e, t)
Output: L
 1: Q ← {V}
 2: L[s] ← 0 , L[v] ← ∞ ∀ v ∈ V s.t v ≠ s.
 3: while Q ≠ {} do
 4:    Let u ∈ Q be the node s.t L[u] = min_{x∈Q} L[x]
 5:    Q = Q ← {u}
 6:    for each edge e ∈ E, s.t. e = (u, v) do
 7:       if L[v] > (L[u] + w(e, L[u] + T)) then
 8:          L[v] ← L[u] + w(e, L[u] + T)
 9:       end if
10:    end for
11: end while
```

Algorithm 1: Dijkstra's Algorithm modified to use time-varying edge costs. s is the source node. T is the start time. $L : V \rightarrow R$ is the array returning the cost of the shortest path for all nodes. The cost function $w : E \times R^+ \rightarrow R^+$, gives the cost as a function of edge and time. The interpretation of $w(e,t)$ is the following: Let e be an edge from node u to node v. Given a message at u at time t, $w(e,t)$ is the cost (delay) of sending it to v. Therefore, if e is taken the message will reach v at time $t+w(e,t)$. The algorithm also works if the network topology is a multigraph. The unmodified Dijkstra's algorithm (for time invariant costs) is the same except that the cost function $w(e, L[u] + T)$ is replaced by $w(e)$ in lines 7 and 8.

different cost assignments, we make sure the above condition holds. In practice it does because, in a single physical link, any message would not be able overtake other messages sent earlier.

The above property does *not* prevent a message from waiting at a node in order to reduce the overall delay. For example, consider a case in which two nodes have two edges between them with different propagation delays. The algorithm may prefer to wait for the lower propagation delay link over the other even if it is currently unavailable.

7.2 Algorithms with Time-Invariant Costs

Algorithm: **Minimum Expected Delay** (MED)
Oracles: Contacts Summary

The cost of an edge is the sum of the average waiting time, propagation delay and transmission delay. The route of a message is independent of time so a proactive routing approach can be used. MED uses the same path for all messages with the same source-destination pair. No mechanism is employed to route around congestion or avoid message drops if storage space is unavailable.

Properties The key property of MED is that it minimizes the average waiting time. It fails to exploit superior edges which become available after the route has been computed. For example, a direct contact to the message destination arises when the message is waiting for the pre-computed next-hop to become available. In this case, the new contact would not be used.

Improvements Finding multiple disjoint paths with similar costs and randomly selecting among them could improve load balancing and reduce congestion [16]. The precomputed route could be modified in-transit if a superior contact becomes available. This would, in effect, make it a form of *loose source routing*, with a somewhat reactive behavior.

7.3 Algorithms with Time-Varying Costs

The w function varies with both edge and time. In addition, it depends on the size of the message under consideration (because of transmission delay). It may also depend on the node assigning costs because costs may depend on its local queue occupancy. Therefore, for sake of uniformity, we represent the cost function $w(e,t)$ in the following form:

$$w(e,t) = w'(e,t,m,s)$$

Here, e is the edge, t is the time for which we are computing the cost, m is the size of the message under consideration and s is the node assigning the costs (and invoking Dijkstra's algorithm). The w' function is now defined as:

$$w'(e,t,m,s) \;=\; t'(e,t,m,s) - t + d(e,t')$$

where,

$$t'(e,t,m,s) \;=\; \min\{t'' | \int_{x=t}^{t''} c(e,x)\; dx \;\geq\; (m + Q(e,t,s))\}$$

The functions $c(e,t)$ and $d(e,t)$ are the capacity and the propagation delay functions for the DTN topology (given by the contacts oracle). The function $Q(e,t,s)$ is the queue size at the source of edge e at time t as predicted by the node s. The parameter s in $Q(e,t,s)$ is used to distinguish between local and global queuing. We now explain how w' models the delay that will be seen by the message when sent over the edge e starting at time t.

t' is the earliest time the queued data at edge e and the message under consideration can be unloaded into the network for transmission (assuming FIFO queuing). The integral captures

the volume of data that could be moved through the edge during the time interval $[t, t'']$. The $d(e, t')$ function captures the propagation delay seen by the message. Therefore, the w' function represents the total time to transfer a message of size m over edge e (starting at time t).

The cost assignments used in the algorithms we present next differ primarily in the definition of the Q function. The first algorithm uses only the contacts oracle and lacks information about either local or global queuing (it thus assigns a value of zero to Q). The second algorithm assigns a non zero value to $Q(e, t, s)$ whenever e is a local edge with respect to s and a zero value otherwise. Finally, the third algorithm uses the queuing oracle to determine $Q(e, t, s)$ everywhere, thereby incorporating global knowledge about the status of queues.

Algorithm: **Earliest Delivery** (ED)
Oracles: Contacts

This algorithm does not incorporate any queuing information and assigns a value of zero to the Q function:

$$Q(e, t, s) = 0$$

Computed routes are loop free, because the paths computed by the Dijkstra's shortest path algorithm are loop free. The route is determined once at the source and fixed, making ED a form of source routing. Paths are computed without considering the availability of storage at intermediate nodes on the path and this may lead to drops when buffers overflow. The route computation is not affected by existing messages in the system, limiting ED's ability to route around congestion.

Properties ED is optimal in the following two cases: if the nodes on the selected path have no queued messages, or if contact capacities are sufficiently large. In the first case, ED is optimal because it has assigned the correct value (zero) to the Q function everywhere. In the second case, it is also optimal (even if queuing is not zero) because once an edge is available and its capacity is large, the time to transmit all the queued data (including the message itself) is negligible. Thus, the queue size does not affect the delay in these cases, and ED's selection of a zero value for Q is appropriate.

Paths computed by ED do not take into account queuing delays. If an edge is purported to be available at a certain time, the algorithm assumes that it can, in fact, be used to send the message. However, if many other messages are ahead in the queue, the contact may finish before the message is sent. This is now disastrous because the time the message reaches the next-hop is very different from what was predicted when the route was computed. Therefore, continuing the route computed earlier may be far from optimal. We explore this point in detail in our evaluation in Section 9.4.

Algorithm: **Earliest Delivery with Local Queuing** (EDLQ)
Oracles: Contacts

In this algorithm, local queue occupancy is taken into account in estimating the edge delays. The Q function is:

$$Q(e, t, s) = \begin{cases} \text{data queued for } e \text{ at time } t & if\ e = (s, *) \\ 0 & \text{otherwise} \end{cases}$$

The Q function accounts for queuing at all edges outgoing from the current node (denoted by $*$ above) and helps to route around congestion at the first hop. It does not account for queuing that will be encountered when the message reaches other nodes in the path. However, unlike ED, we recompute the route at every hop (per-hop routing). This allows the path traversed to be sensitive to the queuing present at all edges in the path.

Properties The EDLQ cost assignment function depends on the node which is computing the route. This may lead to loop formation and the possibility that messages may oscillate forever. Such oscillations can be avoided by employing path vectors and performing a re-computation with fixed routes (e.g. calculated using ED) when a loop is detected. Like ED, messages might get dropped because of buffer overrun.

Algorithm: **Earliest Delivery with All Queues** (EDAQ)
Oracles: Contacts, Queuing

EDAQ uses the queuing oracle to determine the instantaneous queue sizes across the entire topology at any point in time. The Q function is:

$$Q(e, t, s) = \text{data queued for } e \text{ at time } t \text{ at node } s$$

Like ED, messages are source routed. Routes are not recomputed at every hop because when routes were selected, the Q function already took into account queuing at all nodes.

After computing the best route for a message, edge capacity must be *reserved* for the message over all edges (at appropriate times) along its path. Such reservations ensure that messages will have been moved in sufficient time to avoid missing scheduled contacts. In addition, reservations allow the queuing oracle to make accurate predictions about queuing in the network. Realizing bandwidth reservations is likely to be a significant challenge for a DTN, where communicate with some nodes may be significantly delayed. For systems where centralization is practical (e.g. a separate low-delay control network exists), bandwidth allocation would be greatly simplified.

Properties EDAQ determines an optimal route for a new message given existing reservations for the previous messages. This follows because it correctly accounts for the queuing delay, provided by the queuing oracle. Bandwidth reservations are required to accurately implement the queuing oracle. Finally, like previous algorithms, EDAQ is also oblivious to available buffer capacity. Incorporating storage constraints within the framework of computing shortest paths by assigning costs is much harder and open for future investigation. One approach is to use a different algorithmic methodology such as a linear programming formulation, as discussed in the next section. Dynamic flow control may also be effective.

7.4 Other Algorithmic Variations

The algorithms discussed here can be extended in various ways. We briefly introduce these ideas and leave a detailed study for future work.

Active route re-computation The ED algorithm may perform poorly when computing paths over a congested network because precomputed routes remain unchanged even if intended contacts are missed during delivery. This can be addressed by re-computing routes as a message transits the network (i.e. when it becomes apparent a message will miss its next anticipated contact).

Global queuing estimation Implementing the queuing oracle (used by EDAQ) in a practical system will be difficult due to its requirement for global knowledge. To approximate its function, the EDLQ algorithm can be augmented by keeping track of the size of messages it forwards along each routing path. This can be used to construct an an estimate of queuing at nodes beyond the local one, so as to mimic the queuing oracle, but can still be implemented with only local computations.

Splitting revisited The algorithms discussed so far determine a single route for a given message. This approach is fine if messages are small enough to be sent completely during one contact. But if messages are large, it may be better to split them and send different parts over different paths. A slight variant of EDAQ can be used to determine both the split sizes and the routes. The algorithm works by repeatedly invoking EDAQ to determine the shortest path in the network (for a very small message size). After each invocation of EDAQ, the capacity of the computed shortest path is determined and that amount of data is sent over it. Reservations have to be made before the next iteration to ensure correctness. This idea is similar to the use of shortest path algorithms in determining a minimum cost flow in a network (known as the sequential shortest path algorithm) [1]. Another heuristic approach is to split a large message into small messages of a predetermined size. In many cases, the underlying edge properties can be used to determine a suitable size. Each of the smaller messages can then be routed independently. This offers simplicity at the cost of sub-optimal routes.

8. ROUTING WITH COMPLETE KNOWLEDGE

In the previous section we considered algorithms that compute routes without regard to the future traffic demand. It can be shown easily by constructing examples that these algorithms are not globally optimal. The sub-optimal performance is fundamental because these algorithms lack knowledge about the traffic demand and do not consider the buffer constraints. We now present a Linear Programming formulation that uses all the oracles to determine the optimal routing for minimizing average delay in the network. The LP formulation is an adaptation of the dynamic version of the classical multi-commodity flow problem [12]. The dynamic version involves balancing flow during a set of disjoint *time intervals*. Thus, the first step in employing an LP approach is to determine the time intervals over which the balance equations must hold. The second step is to construct the other LP constraints for the DTN routing problem in which edges and nodes are capacitated in a time-varying fashion. These constraints may cause messages to split.

8.1 Time Intervals

For the DTN LP formulation, we have a flow balance equation for each time interval. Each equation balances the flow entering or leaving a node against both input/output flows (as in the traditional multi-commodity flow problem) and flows entering or leaving the local buffer. One simple approach to obtain the set of time intervals is to discretize time into very small, fixed-sized intervals. However, this can lead to a very large number of time intervals and may not be practical. Instead, we can show that a set of intervals satisfying a certain subdivision property is sufficient (though not necessary) to achieve the same results. Intuitively, these time intervals represent time shifts of the periods when message transmission or reception can occur. Contact start and end times and the message arrival times are the natural markers for constructing these intervals, but with propagation delays, further refinement is needed. For a detailed discussion of the algorithm to determine these time intervals (and proofs of their sufficiency to solve the problem) please refer to the technical report [2]. The rest of the paper assumes that the time intervals have been determined.

8.2 LP Formulation

Time Intervals The set of time intervals is denoted by $\mathcal{I}_E$. The time intervals are numbered $I_1, \ldots, I_q, \ldots, I_h$. Further, by construction $I_q = [t_{q-1}, t_q)$ (and $t_{q-1} < t_q$). Therefore, the set $\mathcal{I}_E$ partitions the interval $[t_0, t_h)$.

For an interval $I = [a, b)$ and $r \in R^+$, we let $I \oplus r$ denote the (shifted) interval $[a + r, b + r)$. The conditions for constructing time intervals enforce that an interval, when shifted by any propagation delay, is also a valid interval in the set $\mathcal{I}_E$. This ensures that for all edges, the messages transmitted during an interval of the sending node would be received at the destination node also in a valid interval.

We make an important assumption that both the capacity and delay of an edge are constant over a time interval. If not, the time interval can be subdivided so that the variable capacity (or delay) can be satisfactorily approximated by a constant in each interval.

Graph construction The following definitions are based upon our discussion of the DTN model in Section 3.

V is the set of nodes in the network.

E is the set of edges in the network.

$c : E \times R^+ \rightarrow R^+$, where $c_{e,t}$ is the capacity of the edge e at time t.

$d : E \times R^+ \rightarrow R^+$, where $d_{e,t}$ is the propagation delay of the edge e at time t.

b_v is the storage capacity of the node v.

I^{v} is the set of edges whose destination node is v (incoming edges).

O^{v} is the set of edges whose source node is v (outgoing edges).

Traffic demand Traffic demand is the set of all messages and is denoted by K. A message is a tuple (u, v, t, m) where (u, v) is the source-destination node pair, t is the time at which the message is injected and m is the message size. For a message $k \in K$, the functions s(k), d(k), $\omega(k)$, m(k) are used to retrieve the source node, the destination node, the start time and the size of the message respectively.

Variables used in formulating the LP The following definitions capture the state and the transitions in the network.

- $N^k_{v,t}$ is the amount of message k occupying the buffer at node v at time $t \in T_E$
- $X^k_{e,I}$ is the amount of message k transmitted (at the tail of the edge) over edge e during $I \in \mathcal{I}_E$
- $R^k_{e,I}$ is the amount of message k received (at the destination of the edge) over edge e during $I \in \mathcal{I}_E$
- $K^v = \{k | k \in K \text{ and } \mathrm{d(k)} = v\}$ i.e the set of messages whose destination node is v.

The transmission variables (denoted by X) and the reception variables (denoted by R) are used together to model the propagation delay encountered in sending messages.

Objective function The objective function is to minimize the average delay, which can be realized by minimizing the sum of the delays for all messages:

$$\min \sum_{v \in V} \sum_{k \in K^v} \sum_{I_q \in \mathcal{I}_E} (t_{q-1} - \omega(k)) \cdot (\sum_{e \in \mathrm{I}^{\mathrm{v}}} R^k_{e,I_q} - \sum_{e \in \mathrm{O}^{\mathrm{v}}} X^k_{e,I_q}) \quad (1)$$

The summation, $\sum_{e \in \mathrm{I}^{\mathrm{v}}} R^k_{e,I_q}$ represents the amount of data belonging to message k that is coming into the node v in the interval I_q. Because of limited storage, it is possible that the data leaves the destination node temporarily to some other node with more storage and re-enters it at a later point. This is accounted for by subtracting the term $\sum_{e \in \mathrm{O}^{\mathrm{v}}} X^k_{e,I_q}$. The above difference is multiplied by the length of time that has passed since the start of that message (i.e. $t_{q-1} - \omega(k)$) to get the total

delay suffered by that fraction of the message k that arrived in the interval I_q at node v. Finally, we sum over all the possible intervals for all messages and all nodes.

Observe that if a message is received over two time intervals (due to splitting) the net delay of the message is the weighted sum of the delay of the two fractions. A more natural approach would be to consider the time at which the last fraction of the message is received. However, we found that modeling that as a linear constraint requires the use of integer variables. This results in significantly increased computational complexity as compared to solving standard linear programs and therefore was not considered.

LP constraints

$$\sum_{e\in \mathrm{I^v}} R^k_{e,I_q} - \sum_{e\in \mathrm{O^v}} X^k_{e,I_q} = \begin{cases} N^k_{v,t_q} - N^k_{v,t_{q-1}} + \mathrm{m(k)} & \text{if } \mathrm{s(k)} = v,\ \omega(k) = t_q \\ N^k_{v,t_q} - N^k_{v,t_{q-1}} & \text{otherwise} \end{cases} \qquad k, v, I_q \quad (2)$$

$$R^k_{e,I_q \oplus d_{e,t_{q-1}}} = X^k_{e,I_q} \qquad k, e, I_q \quad (3)$$

$$\sum_{k\in K} N^k_{v,t_{q-1}} \le b_v \qquad v, I_q \quad (4)$$

$$\sum_{k\in K} X^k_{e,I_q} \le c_{e,t_{q-1}} \cdot |I_q| \qquad e, I_q \quad (5)$$

$$N^k_{v,t_0} = \begin{cases} \mathrm{m(k)} & \text{if } v = \mathrm{s(k)},\ t_0 = \omega(k) \\ 0 & \text{otherwise} \end{cases} \qquad k, v \quad (6)$$

$$N^k_{v,t_h} = \begin{cases} \mathrm{m(k)} & \text{if } v = \mathrm{d(k)} \\ 0 & \text{otherwise} \end{cases} \qquad k, v \quad (7)$$

The notation k, v, I_q in equation (2) means that there is an equation of this form for all $k \in K$, $v \in V$ and $I_q \in \mathcal{I}_E$. The same is true for other equations.

Equation (2) gives the flow constraints, which balance the change in the storage occupancy of a node against the net incoming flow for every time interval.

Equation (3) relates the variables X and R by stating that the traffic transmitted at the initial point of e during I_q is equal to the traffic received at the end point of e during the time interval $I_q \oplus d_{e,t_{q-1}}$ (i.e. after the edge delay).

Constraints are also needed to ensure that the amount of data sent over a link is limited by its capacity over that time interval and that the storage at any node does not exceed the specified limit. These are captured by equation (4) and equation (5).

Finally, equations (6) and (7) are the initial and the final conditions regarding storage. Equation (6) says that in the beginning, only nodes that have messages to send have an occupied buffer. Equation (7) states that at the end, only nodes that are destinations for messages have an occupied buffer.

9. PERFORMANCE EVALUATION

9.1 Simulator for DTN

We use simulation to compare the performance of the routing algorithms in different environments. We developed a publicly-available custom simulator for DTN environments [7]. We use it to explore two scenarios, the remote village described in Section 2 and a city bus network scenario (Section 9.3).

The DTN simulator is a discrete event simulator written in Java. Its main theme is to simulate DTN-like store and forwarding of messages over long periods of time sustaining link disconnections and failures. The two key components of the simulator are the nodes and the links, which can be created and destroyed dynamically (and also temporarily or permanently). Nodes have finite storage capacity.

Links are attached to nodes and are directional by default. They have finite propagation delay and finite bandwidth. To model link unavailability (or availability), *patterns* (simulator objects) can be associated with a link to specify the exact time intervals during which the link is down (or up). These patterns can be generated randomly from a probability distribution or specified explicitly in a file. Unlike traditional network simulators, the DTN simulator distinguishes between the following two modes of link disconnection:

1) *Complete link failure*: causes all the transiting messages on the link to be dropped. This corresponds to failure of the physical media or complete interference.

2) *Link close at the source*: prevents sending any more data into the link. The data that has been sent, however, continues to reach the destination. This is similar to a wireless/satellite connection fading away. It is especially relevant for networks with very long propagation delays.

The simulator also supports *reactive fragmentation*. If a message is being sent when a link closes, a fraction of the message is transferred successfully (the amount is based upon the transmission start time, the link capacity, and the close time). The transmitting node is then informed of the amount of data transmitted, and is given the opportunity to route the remaining fraction. The final destination of the message is responsible for reassembling its constituent parts.

9.2 Scenario 1: Routing to a Remote Village

We now return to the problem of routing to a remote village mentioned in Section 2 (Figure 1). The village is Kwazulu-Natal and and the city is Capetown, both in South Africa. Their respective (latitude, longitude) locations are as follows: (28.8830S, 31.4670E) and (31.282S, 29.45E). We shall assume the dialup provides 4 Kb/s and is available only during late night (11 pm to 6 am, local time).[2] We assume the availability of three PACSAT satellites for the purpose of delivering data: OSCAR-11, PACSAT and PCSAT [3]. We use the PREDICT [15] satellite tracking software to determine the time of the passes of these satellites. The satellite is assumed to be in range of both the city and the village at the same time as it is flying over them. This is a reasonable assumption because the village and the city are geographically close in this case. We also have three motorbikes that travel from the village to the city (and vice-versa) every day at different times. Each trip takes about two hours (one way), the bandwidth to/from the motorbike is taken to be 1Mbps, its contact time at the city or the village is 5 minutes, and it can store up to 128MB (the size of a USB dongle).

These (partially hypothetical) parameters represent only one of the many ways to connect a remote village, and we cannot currently claim it to be representative. Nevertheless, even this relatively simple scenario exhibits a richness in routing decision opportunities and allows us to examine the type of choices made by our proposed algorithms. In the next section, we consider a more general and complex DTN topology and explore its performance in more detail.

[2]During day time it is too expensive to use this connection.

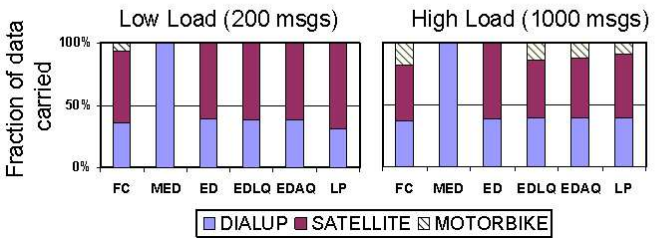


Figure 4: Traffic breakdown by different connectivity options.

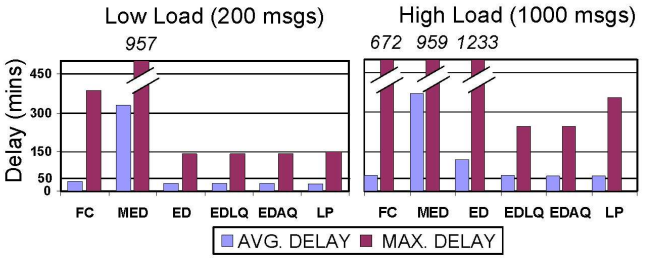


Figure 5: Delay comparison for different algorithms.

Traffic demand generation Messages are injected at the village (destined for the city) and vice-versa. Messages from the village are small (1KB average) and messages from the city to the village are larger (10KB average). These are based upon average web request and web response sizes [21]. We consider two traffic rates, a) a *low load* of 200 messages from village to city (and vice-versa) per day, and b) a *high load* of 1000 messages per day. Messages are injected at randomly-chosen points in time during the first 24 hours of simulation. Simulation is performed over a period of 48 hours, starting at 11–Jan–2004, 11:59 pm.[3] The simulation duration is chosen to ensure that no traffic remains undelivered when the simulation ends.

Routing issues The topology of this scenario is straightforward. In some sense, the "routing" decision is merely to choose the first hop among the three classes of contacts: the satellite, the dialup, or the motorbike. Consider a message originating at the village at 6 pm (local time). The viable options for delivering the message are: 1) wait until 11 pm to use the dialup connection, 2) use the motorbike scheduled to leave the village at 8 pm, or 3) use a satellite which will be visible at 8:10 pm.

Taking the first available contact may be sub-optimal. Here, the first contact (motorbike) would deliver the message to the city at 10:00 pm. Using a satellite would have been the best option in this case because the message would be transferred to the city (using the satellite as an intermediate hop) at 8:10 pm, saving 110 minutes.

The routing decision becomes more complex when a large amount of data has to be delivered. In the same example, if the size of the message is larger than what can be delivered using the satellite, choosing the higher-bandwidth motorbike may be better. Clearly, these decisions depend on the size and time of the requests, the available connectivity options, and the other messages already waiting in the system.

Results One interesting question to investigate is how different routing algorithms utilize the available connectivity resources (i.e. dialup, satellites, and motorbike). To understand this, we plot in Figure 4 the fraction of data routed using each connectivity mode by the different algorithms.

For MED, all the data is routed using the dialup connection during both low and high load because MED statically chooses a single path to route all data based on aggregate link properties. The dialup connection, which is available for a significant fraction of the day (7 hours), has the best average delay. For ED, most data is routed using satellites (60%) and the rest using dialup. No traffic is routed using the motorbike because doing so would require at least two hours to reach the city and vice-versa. Thus, ED will take the motorbike only if both the satellite and the dialup connections are more than two hours in the future. This rarely occurs because there are three satellites each visiting about four times a day. For low loads, EDLQ and EDAQ make choices similar to those of ED.

The FC algorithm routes data based on the first available contact and sometimes selects the motorbike (for about six percent of the messages). This explains why the maximum delay for FC is much higher as compared to EDAQ. The average, however, is similar to the other algorithms because only a small fraction of the traffic is routed this way. The moderately good performance of a simple technique like FC is somewhat surprising at first glance and is due to the simple topology under consideration. FC can never make a terrible choice because all paths lead directly to the city. In Section 9.3, where we consider a more complex topology, the performance of FC is much worse.

At higher loads, the situation is somewhat different. The congestion-aware schemes (EDLQ, EDAQ, and LP), which under low load did not use the motorbike at all, now route about fifteen percent of the data using it. However, the choices made by ED and MED remain the same as in the low load case because they are not traffic-aware. Both suffer because of this. In particular, the performance of ED deteriorates sharply when the next-hop is a satellite. Because of the large number of requests, only some of them can be served during one satellite pass. The rest have to wait for the next visible pass of the same satellite (as constrained by the ED algorithm) which, in the worst case, can be as long as ten hours. FC continues to route using the first available contact. With an increased volume of traffic, a larger proportion of it gets routed through the motorbike because the motorbike offers higher bandwidth and can consume data at a faster rate than either the motorbike or the satellites.

We also solved this scenario using the LP formulation presented in Section 8. We used the CPLEX optimization suite [6] from ILOG on an 8-processor Pentium-III (700MHz) machine with 3 GB of RAM. Even for this simple scenario, the resulting LP had close to 500,000 constraints containing 550,000 variables and took about 8 minutes with 16,000 iterations to solve in CPLEX. For more complex scenarios (as the one presented in Section 9.3), the size of the resulting LPs were very large and we were unable to solve them practically.

A key observation is that EDAQ compares favorably, in terms of average delay, with the optimal solution. We also observe that the LP's maximum delay sometimes exceed that of EDAQ. This is possible because LP optimizes for the *average* delay.

[3]The precise starting time, coupled with the location on Earth, is required by the satellite tracking software in order to determine the satellite contact parameters.

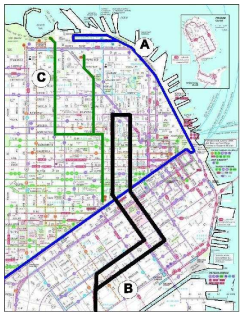


Figure 6: Map of San Francisco used for the bus movements. The physical cross-section of the above area is 4400m X 5600m. The three labeled lines A,B,C denote three of the twenty bus routes. It takes approximately 30 minutes for Bus A to complete one round-trip.

Discussion The village scenario is a simple scenario, but serves to illustrate that many factors have to be taken into consideration when making a routing decision. Using our evaluation framework, we find that simple techniques may fail to deliver the best performance, especially under congestion. We use our framework in the next section to explore a more complex scenario with multiple hops and mobile routers.

9.3 Scenario 2: A Network of City Buses

Our second scenario is a city bus network. The nodes in the scenario are 20 city buses making scheduled bus trips inside San Francisco (Figure 6). The buses are capable of DTN-style store and forward operations and are equipped with radio-based communication capabilities. We compute the complete DTN graph for this scenario in two steps.

In the first step, we plot the motion of each individual bus on a two-dimensional plane representing the city. For each bus, an ordered sequence of stops (representing its route) is placed on the plane. Every time the bus makes a turn onto a new street, a stop is generated at the corresponding intersection. The bus moves along a straight line between two bus stops with a constant base speed. We use the actual bus routes published in the transit bus network schedule for San Francisco. We then add random pause times (chosen uniformly between 0 and 5 minutes) at each stop. In addition, the bus base speed is modified at each stop (chosen uniformly between 10 and 20 meters/second).

In the second step, we compute the time intervals during which the buses can communicate. We assume that the buses are fitted with radio transceivers and the ability to communicate is based on a very simple disc radio model. In the disc model, two buses can communicate if they are within a certain threshold distance, called the radio range. This model is an approximation and its main goal here is to provide a simple way to generate (dis)connections based on a known mobility pattern. To generate the time intervals when the buses can communicate, we wrote a separate program which moves nodes on a two-dimensional surface and computes the times when nodes are separated by the radio range threshold or less. The output of this program provides the dynamic topology input needed by the DTN simulator. The default radio range is 100 m in simulations. We also study the effect of varying radio range.

Traffic workload Traffic is generated over a period of 12 hours, which is divided into 12 intervals of 1 hour each. For each interval, 20 random source/destination bus pairs are chosen. Each source bus sends 200 messages to its destination bus during the one hour interval. The messages are injected simultaneously at a randomly-chosen time inside the one hour interval. This represents a bursty traffic pattern and creates more congestion in the network as compared to a more uniform load. The total simulation time is 24 hours. It gives most of the algorithms enough contacts to completely deliver the traffic demand.

Load As we shall see, the relative performance of the different algorithms is most sensitive to the amount of congestion present in the network. Congestion in a DTN depends on the relative ratio of traffic demand to the product of capacity and frequency of contacts. We define this ratio as the *load* on the network:

$$Load = \frac{Traffic\ Demand\ in\ time\ T}{Contact\ Volume}$$

where,

$$Contact\ Volume = \sum_{e \in E} \int_0^T c(e,t)\ dt$$

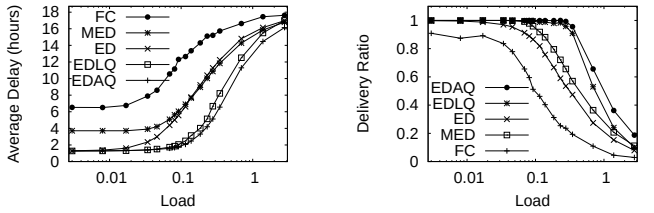
The contact volume gives the maximum amount of data that can be exchanged during the whole simulation time (T). The traffic demand accounts for the entire volume of messages injected into the simulation. Even with a load of less than one, the network may be congested. A message may have to traverse multiple hops to reach its destination, thus reducing the effective usable bandwidth to handle other traffic demand. Furthermore, because both traffic demand and contacts are time sensitive, there may be times when contacts are available but no traffic is present to utilize them, thus underutilizing the available contact volume. In spite of these limitations, the above ratio provides some useful insight regarding the relationship between scenario parameters and algorithm performance.

Parameter sensitivity The load can be increased by either generating additional traffic demand or by reducing the contact volume. We fix the traffic demand and vary the contact volume. The contact volume can be varied by either varying the contact bandwidth (edge capacity) or the contact duration (the fraction of time an edge is available). For our scenario, the contact durations can be increased (decreased) by increasing (decreasing) the radio range.

Although increasing bandwidth and increasing the radio range both decrease the overall load on the network, they have different effects. The differences can be understood by observing the impact of parameter changes when traffic demand is minimal. A small radio range effectively disconnects the network even if bandwidth is plentiful. If bandwidth is limited but radio range is large, however, the impact on delivery is only minimal for light traffic. Thus, the role of bandwidth is prominent only when there is relatively large amounts of data to be moved.

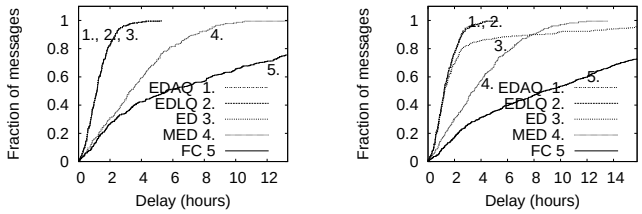
Performance metrics In addition to average delay, we also use *delivery ratio* to compare our algorithms. The *delivery ratio* is defined as the ratio of the total amount of data delivered by the end of simulation to the total amount of data injected into the system. The delivery ratio may be less than one either because of buffer drops or because of insufficient contact volume to move data to its destination before the end of the simulation.

The next three subsections present a comparison of different algorithms as we vary bandwidth, radio range and buffer capacity. In each section, only one parameter is varied and the rest are kept constant. The default value for radio range is 100m. The default storage capacity was 100 MBytes and the default link bandwidth was 100 Kb/s.



(a) Average delay (b) Delivery ratio

Figure 7: Effect of scaling load (by scaling bandwidth). Radio range is fixed at 100m.



(a) Bandwidth=400Kbps (b) Bandwidth=20Kbps

Figure 8: Cumulative Distribution Function of delay for the set of all messages. Radio range is fixed at 100m.

9.4 Results of Varying Bandwidth

Figures 7(a) and (b) show the average delay and the delivery ratio, respectively, as load on the system increases (by decreasing bandwidth).

When the load is very small (less than .01), increasing bandwidth does not lead to any improvement in delay. At low load the bottleneck is due to poor timing: an insufficient volume of contacts is available when needed even though the *aggregate* volume of all contacts is sufficient to move all the generated data. However, in this operating range, the dynamic cost-based algorithms ED, EDLQ, and EDAQ (smarter algorithms) perform much better than FC and MED (simpler algorithms). For example, at a load of .01, MED has three times the delay of ED. As expected, FC has the worst performance among all the algorithms because of its essentially random selection of the next-hop.

As the load increases (or, equivalently, bandwidth decreases), average delay increases because the amount of data generated is comparable to the amount that can be moved in one contact. Therefore, multiple contacts are often required to move the data stored at a node. The effect of increased load is most evident in the line of ED in Figure 7. At low load (about 0.05), ED performs as well as EDAQ and EDLQ, but as load is increased its performance deteriorates and becomes similar to MED. This behavior is explained by recalling that under congestion a contact may finish before all the messages queued for it are sent. In the context of ED, this can now be disastrous for the unsent messages. Those messages delayed only to the next-hop (but reaching in time to catch the contact scheduled at the next-hop) continue on the planned route. However, messages that miss contacts at subsequent nodes as a result of missing the scheduled contact at the current node can get significantly more delayed. The situation is analogous to missing a connecting airline flight. Somewhat surprisingly, EDLQ and EDAQ have similar performance.

For very high load (above one), most of the data cannot be delivered during the simulation run (Figure 7(b)). For undelivered messages, we assign their delivery time as the end of the simulation. This underestimates the delay (perhaps significantly), but also ensures that we never overestimate it. As a consequence, all the algorithms appear to have almost the same (very high) delay when the load is extremely high.

Figures 8(a) and 8(b) show the cumulative distribution function of delay of the set of messages for the two cases: a) when bandwidth is high (400 Kb/s), and b) when bandwidth is low (20 Kb/s). The graph indicates superior performance of the smarter algorithms ED, EDLQ and EDAQ. It also illustrates that ED has exactly the same (optimal) performance when bandwidth is large, illustrated by observing that the lines for ED, EDLQ and EDAQ coincide. However, when bandwidth is low, the performance of ED is significantly worse than both EDLQ and EDAQ. The almost-overlapping CDFs for EDLQ and EDAQ again indicate that their performance is close, even at high load.

Radio range (km)	.05	.1	.2	.4	1	2	4
f_{up}%	.1	.2	.6	2.6	15	45	97

Table 2: Network connectivity as a function of radio range. f_{up} denotes the percentage of time a pair of nodes are in mutual contact. $f_{up} = 100$ denotes that two nodes are in continuous contact. The numbers reported here are obtained by averaging f_{up} over all pairs of nodes.

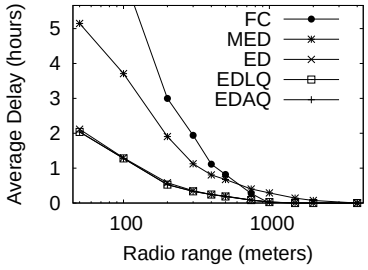


Figure 9: Average delay as radio range is scaled. The plots for ED, EDLQ and EDAQ overlap.

9.5 Results of Varying Radio Range

Figure 9 plots the average message delay as radio range is increased (which effectively increases contact volume). For this plot, the link bandwidth was taken as large, allowing us to factor out congestion issues due to limited bandwidth.

From the graph, one can see that increasing the radio range reduces the average delay. This is expected, because by increasing the radio range, the buses are in mutual contact more frequently (and for longer duration) and hence, the waiting time is reduced. Table 2 shows the increase in network connectivity as radio range is increased (spanning nearly three orders of magnitude).

The difference between the simpler (FC, MED) and the smarter algorithms (ED, EDLQ, EDAQ) is much more pronounced when the radio range is small (i.e when the network is more disconnected), suggesting the increased benefits of smarter routing techniques as networks become more intermittent. Conversely, when radio range is very large (links are mostly available), the benefits of the smarter algorithms vanishes.

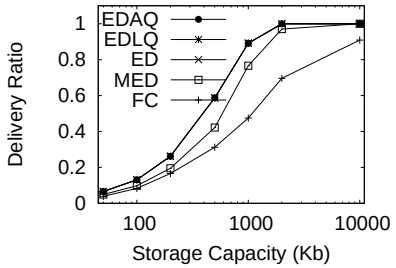


Figure 10: Fraction of data delivered to data generated as a function of storage capacity at nodes. The plots for ED, EDLQ and EDAQ overlap.

9.6 Results of Varying Buffer Capacity

Most of the algorithms considered in this paper (except LP) are oblivious to buffer limitations. Therefore, we expect to see message drops when storage is limited. To explore this behavior, we vary the maximum available storage at each node to see how different algorithms perform with respect to message delivery. Figure 10 plots the data delivery ratio for different algorithms as a function of storage. The bandwidth is 400Kb/s (large) and the radio range is 100m.

When storage is limited, the smarter algorithms (ED, EDLQ, EDAQ) have significantly fewer drops than the simpler algorithms (FC, MED). When storage is sufficiently large, there are no drops and all algorithms have a delivery ratio of one. However, when storage availability is *very* limited, most messages get dropped and there is little benefit in employing a smarter route selection process. Therefore, we conclude that ED, EDLQ and EDAQ perform better than FC and MED when storage is limited, but any such performance benefits may be unrealized for networks with extremely large or extremely small store-and-forward buffering.

9.7 Summary

It is evident that the smarter algorithms (ED, EDLQ, EDAQ) outperform the simpler algorithms (FC, MED), both in terms of delay and delivery ratio. The performance differences become more pronounced as the network becomes more intermittent. Further, as load is increased, ED performs much worse than EDLQ and EDAQ, as it is unable to mitigate the effects of congestion. Finally, EDLQ, which routes around congestion using only local queuing, performs comparably to EDAQ. This point is encouraging from an implementation standpoint, as implementing the queuing oracle on a frequently-disconnected network would likely present a significant distributed systems problem.

10. RELATED WORK

10.1 Store-and-Forward Systems

Electronic mail may appear as a natural approach for handling message delivery in a frequently-disconnected environment. It does provide the required store-and-forward capability, but typically lacks any robust approach to dynamic routing. In the Internet, successful e-mail exchange (i.e. with SMTP) generally depends both on a successful DNS request/response transaction along with a reliable transport layer exchange protocol. The mail exchanger DNS facility ("MX record") provides a limited form of routing in the e-mail overlay, but falls far short of handling the type of routing problem discussed here. Thus, e-mail (at least Internet e-mail) is unable to deal with true dynamic routing and is not very robust to errors during message transfer.

Prior to the wide availability of Internet connectivity, the UUCP network carried news and electronic mail to a small but growing (and active) user community. It used source routing, and responsibility for route selection was left primarily to end-users. It was later enhanced with a form of static routing based on computing shortest paths on a topology graph distributed by e-mail (or entered manually). [4] To choose among local outgoing connections, fixed costs were assigned manually to links based upon the frequency and the quality of the connection. The MED algorithm is similar to above idea.

10.2 Optimization Techniques and Network Flows

The field of operations research is rich with variations on optimization problems involving network flows, shipment of materials, and scheduling. A complete survey could not possibly be accomplished here. However, some of the more relevant work includes the quickest transshipment problem, dynamic multi-commodity flow problems, etc. [14]. Frequently, network flow problems that involve temporal sensitivities are solved as traditional graph problems on *time-expanded graphs*. These graphs, introduced as early as 1962 by Ford and Fulkerson [9], can be used to capture the temporal dynamics as additional nodes and edges. The issue with computing on such graphs is that they can significantly expand the search space, leading to very large problems.

Focusing specifically on shortest path (single path) solutions, algorithms for dynamic networks have been investigated by Orda et.al. in [19]. In this respect, our primary contribution has been to define appropriate cost functions in the context of the DTN routing problem. The issue of splitting (allowing message fragments to take multiple paths) and buffer constraints makes the problem much more challenging. The problem is further complicated because of multiple commodities, time-varying capacities, non-zero and possibly time-varying propagation delays, etc. The specific case of zero propagation delays with finite buffers and time-varying capacities has been addressed by Ogier [18]. In summary, although many of these aspects have been discussed in literature either individually (or as a subset), the complete LP formulation taking into account all these aspects is novel.

10.3 Routing in Disconnected Mobile Adhoc Networks

Our present exploration of the DTN routing problem is focused on cases where the topology dynamics are known (or nearly known) in advance. Clearly, however, many systems will not exhibit such predictability or will exhibit partial predictability. A series of efforts [13, 22, 24, 5] in the context of sensor/mobile-adhoc networks looks into providing connectivity when topology dynamics are unknown. Generally, these techniques employ a form of data duplication within the network and achieve *eventual* delivery. Such duplication requires a way of getting rid of unnecessary copies to reduce the buffer occupancy. We believe that a hybrid approach, possibly employing some of the epidemic techniques in conjunction with the techniques discussed here, may be appropriate for such systems.

11. CONCLUSION

DTN routing appears to be a rich and challenging problem. It requires techniques to select paths, schedule transmissions, estimate delivery performance, and manage buffers. The problem of networking on frequently-disconnected networks is receiving more attention as the desire to have data connectivity in devices

[4] A user program called *pathalias [11]* performed this function.

which may be mobile or into regions that may only be reachable by non-conventional network devices (e.g. motorbikes) increases. We believe that in many frequently-disconnected scenarios, communication opportunities may be predictable. The algorithms devised in this paper focus on these situations, and we believe such systems have received little attention to date.

In this paper, we have developed a framework for evaluating DTN routing algorithms, suggested and evaluated several individual algorithms, and provided a basis for future work in the area. Our findings suggest that in networks with plentiful communication opportunities, the need for smart routing algorithms is minimal. In situations where resources are limited (contact opportunities, bandwidth or storage, in our case) smarter algorithms may provide a significant benefit. Indeed, those which take network congestion into account (EDLQ, EDAQ, LP) do especially well in these environments. The finding that global knowledge may not be required for good performance in many cases suggests that implementing the queuing oracle (the most challenging to realize except for the traffic oracle), in particular, may not be worthwhile. This last point is significant, and merits further investigation. Our construction of the knowledge oracles allowed us to avoid the complexities of how routing meta-data is created and propagated. In real-world situations, however, the realization of these oracles would present a distributed systems challenge made worse by the assumed frequent network disconnections.

12. FUTURE WORK

Our work to date has revealed that many additional questions remain to be answered. We now discuss several of these. The algorithms presented here (except LP) do not account for buffer limits at intermediate nodes when determining routes. If a node has insufficient storage to hold in-transit data, that data is dropped. Flow control mechanisms could be employed to prevent such drops in some circumstances, but many existing methods for handling dynamic flow control do not work well with long propagation delays. Thus, it would appear some form of proactive admission control may be more appropriate, but discovering the best form of this mechanism for DTNs remains open. Removal of messages at a node, either because they have expired or for application specific reasons, is another approach for controlling buffer occupancy.

As mentioned earlier, in some environments contacts become available *opportunistically*. Routing under such environments might employ techniques of epidemic routing (data replication), and the most robust solution may incorporate those techniques with the approaches described here. A related variant is when oracles provide only *probabilistic* knowledge about available contacts (such as a time dependent probability distribution on waiting time). Here, we hope to leverage work from the transportation community on stochastic dynamic shortest paths [10].

The objective function we have selected here minimizes delay, but in some scenarios other metrics (e.g. monetary cost), which may not be directly derivable from delay, may be more important. This would present the problem of how to measure the metric of interest and would again raise the implementation question for the corresponding new knowledge oracle. For the case of village networking presented earlier, this may be especially true, and this line of research is already underway. As can be seen, a rich collection of questions (both theoretical and practical) arise in the context of these types of networks.

Acknowledgements

We are grateful to Gaetano Borriello, Ratul Mahajan, David Wetherall and the SIGCOMM reviewers for providing helpful feedback on the paper. This work was supported in part by the Intel Research Council, Intel Corporation.

13. REFERENCES

[1] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin. *Network Flows: Theory, Algorithms, and Applications*. Prentice Hall, 1993.

[2] J. Alonso and K. Fall. A Linear Programming Formulation of Flows over Time with Piecewise Constant Capacity and Transit Times. Technical Report IRB-TR-03-007, Intel Research Berkeley, July 2003.

[3] AMSAT. `http://www.amsat.org/`.

[4] J. Broch, D. A. Maltz, D. B. Johnson, Y. C. Hu, and J. Jetcheva. A Performance Comparison of Multi-Hop Wireless Ad Hoc Network Routing Protocols. In *ACM Mobicom*, Aug. 1998.

[5] X. Chen and A. L. Murphy. Enabling Disconnected Transitive Communication in Mobile Adhoc Networks. In *Workshop on Principles of Mobile Computing*, August 2001.

[6] CPLEX: Linear Programming Solver. `http://www.ilog.com/`.

[7] DTN Research Group. `http://www.dtnrg.org/`.

[8] K. Fall. A Delay-Tolerant Network Architecture for Challenged Internets. In *ACM SIGCOMM*, Aug. 2003.

[9] L. R. Ford and D. R. Fulkerson. *Flows in Networks*. Princeton University Press, 1962.

[10] S. Gao. Routing Problems in Stochastic Time-Dependent Networks with Applications in Dynamic Traffic Assignment. Master's thesis, MIT, 2002.

[11] P. Honeymoon and S. Bellovin. PATHALIAS: The Care and Feeding of Relative Address. In *USENIX Conference*, 1986.

[12] B. Hoppe and É. Tardos. The Quickest Transshipment Problem. In *SODA*, Jan. 1996.

[13] P. Juang, H. Oki, Y. Wang, M. Margaret, P. Li-Shiuan, and R. Daniel. Energy-Efficient Computing for Wildlife Tracking: Design Tradeoffs and Early Experiences with ZebraNet. In *ASPLOS-X*, October 2002.

[14] B. Kotnyek. An Annotated Overview of Dynamic Network Flows. Technical Report RR-4936, INRIA, Sept. 2003.

[15] J. A. Magliacane. PREDICT: Satellite Tracking Software. `http://www.qsl.net/kd2bd/predict.html/`.

[16] M. K. Marina and S. R. Das. On-demand Multipath Distance Vector Routing in Ad Hoc Networks. In *IEEE ICNP*, Nov. 2001.

[17] D. Niculescu and B. Nath. Trajectory based Forwarding and its Applications. In *ACM Mobicom*, 2003.

[18] R. Ogier. Minimum-delay Routing in Continuous-time Dynamic Networks with Piecewise-constant Capacities. *Networks*, 18:303–318, 1988.

[19] A. Orda and R. Rom. Shortest-Path and Minimum Delay Algorithms in Networks with Time-Dependent Edge-Length. *Journal of the ACM*, 37(3), 1990.

[20] A. Pentland, R. Fletcher, and A. Hasson. DakNet: Rethinking Connectivity in Developing Nations. In *IEEE Computer*, Jan. 2004.

[21] S. Saroiu, K. Gummadi, R. Dunn, S. Gribble, and H. Levy. An Analysis of Internet Content Delivery Systems. In *OSDI*, Dec. 2002.

[22] R. C. Shah, S. Roy, S. Jain, and W. Brunette. Data MULEs: Modeling a Three-tier Architecture for Sparse Sensor Networks. In *IEEE SNPA*, May 2003.

[23] TIER Project. `http://tier.cs.berkeley.edu/`.

[24] A. Vahdat and D. Becker. Epidemic Routing for Partially-connected Ad hoc Networks. Technical Report CS-2000-06, Duke University, July 2000.

[25] Wizzy Project. `http://www.wizzy.org.za/`.