ROUTING IN DELAY TOLERANT NETWORKS WITH PERIODIC CONNECTIONS

A THESIS

SUBMITTED TO THE DEPARTMENT OF COMPUTER ENGINEERING

AND THE INSTITUTE OF ENGINEERING AND SCIENCE

OF BILKENT UNIVERSITY

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

FOR THE DEGREE OF

MASTER OF SCIENCE

By

Cem Mergenci

August, 2010

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

Asst. Prof. Dr. İbrahim Körpeoğlu (Advisor)

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

Prof. Dr. Cevdet Aykanat

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

Assoc. Prof. Dr. Ezhan Karaşan

Approved for the Institute of Engineering and Science:

Prof. Dr. Levent Onural
Director of the Institute

ABSTRACT

ROUTING IN DELAY TOLERANT NETWORKS WITH PERIODIC CONNECTIONS

Cem Mergenci
M.S. in Computer Engineering
Supervisor: Asst. Prof. Dr. İbrahim Körpeoğlu
August, 2010

In delay tolerant networks (DTNs), the network may not be fully connected at any instance of time, but connections occurring between nodes at different times make the network connected through the entire time continuum. In such a case, traditional routing methods fail to operate as there are no contemporaneous end-to-end paths between sources and destinations. This study examines the routing in DTNs where connections arise in a periodic nature. Various levels of periodicity are analyzed to meet requirements of different network models. We propose various routing algorithms for periodic connections. Our proposed methods can find routes that can guarantee earliest delivery and minimum hop count. We evaluate our routing schemes via extensive simulation experiments and also compare them to some other popular routing approaches proposed for delay tolerant networks. Our evaluations show the feasibility and effectivenes of our schemes as alternative routing methods for delay tolerant networks.

Keywords: Delay Tolerant Networks, Routing.

ÖZET

GECİKME DİRENÇLİ AĞLARDA SÜRELİ BAĞLANTILAR İÇİN YÖNLENDİRME

Cem Mergenci
Bilgisayar Mühendisliği, Yüksek Lisans
Tez Yöneticisi: Asst. Prof. Dr. İbrahim Körpeoğlu
Ağustos, 2010

Gecikme dirençli ağlarda, ağ her ne kadar anlık olarak bağlı bulunmasa da düğümler arasında zaman içinde oluşan bağlantılar, ağı tüm zaman aralığında bağlı hale getirir. Böyle bir durumda, geleneksel yönlendirme yöntemleri kaynak ve hedef arasında hali hazırda uçtan uca bir yol bulunmadığı için başarılı olamaz. Bu çalışma, süreli bağlantılardan oluşan gecikme dirençli ağlarda yönlendirmeyi incelemektedir. Farklı ağ yapılarının gereksinimlerini karşılamak için farklı süreli bağlantı türleri çözümlenerek, süreli bağlantılar için çeşitli yönlendirme yöntemleri önerilmekte, önerilen yöntemler en erken ve en kısa yoldan teslimi garantilemektedir. Yönlendirme yöntemleri, ayrıntılı benzetim ve deneyler ile değerlendirilip gözde yönlendirme yöntemleriyle de karşılaştırılmaktadır. Değerlendirmeler, önerilen yöntemlerin gecikme dirençli ağlarda uygulanabilir ve etkin bir seçenek olduğunu göstermektedir.

Anahtar sözcükler: Gecikme Dirençli Ağlar, Yönlendirme.

Acknowledgement

In the first place, I would like to express my gratitude to my supervisor, Asst. Prof. Dr. İbrahim Körpeoğlu, for his guidance and support in supervision of this thesis. I would like to thank especially for his patience, understanding and constant encouragement in the process. Working with him has always been a pleasure.

I am thankful to Prof. Dr. Cevdet Aykanat for being a great teacher. I tried to apply the concepts I learned from him at my best in this thesis. He was very kind to accept being a jury member despite his busy schedule.

I am indebted to Assoc. Prof. Dr. Ezhan Karaşan from whom I learned very much about computer networks. Having him as a jury member for this thesis is very meaningful.

Special thanks to Scientific and Technical Research Council of Turkey (TÜBİTAK) for financially supporting me in my graduate education.

Last, but not least, I am grateful to my family for their sincere and endless support at every moment throughout my life and for providing me every opportunity they can, especially for my education. This thesis would be a dream without them.

Contents

List of Figures

Chapter 1

Introduction

Delay tolerant networks (DTNs) [4] are networks where an end-to-end path between a source and destination is not guaranteed to exist at any time instant. Connections between nodes at different times provide an end-to-end path in future, therefore, making the network connected throughout the entire time continuum. This condition causes DTNs to suffer from large delays; long disconnection times and partitioning in the network are also typical problems, which make communication a challenging task in a DTN.

Not all of the applications are suitable for DTNs. Applications requiring a flow of data, such as multimedia streaming, or a connection to be present, such as secure shell (SSH) or instant messaging (IM), are not good candidates for a high-delay disconnected environment. On the other hand, delay tolerant applications can tolerate large delays and still work as expected. Email is a good example to delay tolerant applications, DNS and Web can also be considered in this category. Although conventional file transfers require a flow of data, different forms like BitTorrent have a delay tolerant nature and can operate in a DTN environment easily.

Apart from the existing applications, very different application types have emerged with the DTN concept. Contextual applications using locally available data are popular in DTNs. Suppose a social networking application running on people's mobile handsets. Researchers from different locations around the world meet in a conference, providing a convenient way for all people with similar interests to come together. Our application collects profile of the attendees and presents a list of people who are working in our research field. Applications for polling, invitation and announcement disseminations can work in a similar manner. Vehicular ad-hoc networks (VANETs), military ad-hoc networks, wireless sensor networks (WSNs), satellite and free-space communication are other fields to which delay tolerant networking concepts can be applied.

The main problem with the DTNs is the fact that existing networking protocols that are in use today, such as ubiquitous TCP/IP stack, assume the availability of an end-to-end path and acceptable round-trip times in communication. These assumptions make them unsuitable to operate in a DTN environment. Even mobile ad-hoc network (MANET) routing protocols like AODV and OLSR are not designed to work for a delay tolerant situation. As a result, different set of networking protocols are devised for DTNs to meet various requirements.

In this study, we examine routing issues in a DTN. We are particularly interested in routing in a DTN with periodic connections. We begin our work with an analysis of a simple connection model, in which contacts occur at a given time in future without any periodicity. We propose an algorithm based on Dijkstra's shortest path algorithm with customizations to meet requirements of the simple connection model. This algorithm guarantees the earliest delivery (ED), therefore, optimal in terms of time. Then we extend the connection model to incorporate periodic connections, which is our main area of focus. In this new model, contacts occur first on a given time and repeats at certain intervals. We revise our routing algorithm to compute future contact times from the new connection model by taking periodicity into account. This connection model assumes contact durations to be insignificant such that a connection becomes available and unavailable instantly though allowing enough time for packet exchanges. This assumption is not very realistic, because connection times are comparable to disconnection times in real scenarios. We introduce a connection model with separate connection and disconnection states taking different times and also occurring periodically in an alternating fashion. We provide the most general version of the ED routing algorithm using this connection model. It computes future contact times accordingly and exploits connection periods.

ED routing is optimal in time, but not in hop count. The greedy approach taken by the algorithm (similar to the greedy approach taken by the Dijkstras shortest path algorithm) produces routes that are suboptimal in hop count. We address this problem with min-hop earliest delivery routing (MHED), which finds paths that guarantee the earliest delivery and are minimal in hop count. There are two flavors of MHED, one based on a Dijkstra-like approach and one based on BFS (breadth first search). Running time analysis reveals that BFS-based MHED runs asymptotically faster than Dijkstra-based MHED.

We perform experiments to compare our algorithm to epidemic routing, which is a popular routing algorithm optimal in time. Results show that ED and MHED routing achieve the same delivery time as epidemic routing by making very few number of transmissions. We compare ED and MHED in terms of average path length and routing tree stability. As expected, simulation results prove MHED finding shorter routes than ED under different scenarios. A side effect of the MHED algorithm is the fact that its routing tree is less susceptible than that of ED, to changes in the connections between nodes. Routing tree of MHED changes less frequently at each time interval.

The rest of the thesis is organized as follows: Chapter 2 gives a general background information about DTNs, explains challenges for such networks and looks at applications. It also presents related work in DTN routing literature, compares and contrasts existing works to our study. Chapter 3 defines the problem and proposes different solutions for different cases. Chapter 4 explains experiments and evaluates the results. Chapter 5 concludes the thesis and gives future research ideas on the topic.

Chapter 2

Background and Related Work

2.1 Delay Tolerant Network Architecture

Roots of delay tolerant networking lies in the work of Interplanetary Internet (IPN). On March 2003, in the first draft of the RFC 4828, the term delay tolerant network was coined to extend properties of IPN to different types of networks [5]. These types of networks suffer from disruptions in connections and partitions in the network.

DTN architecture is evolved to work on top of a very diverse range of communication types. It utilizes network layer services like TCP/IP, MANET routing, WSN infrastructure and uses link layer technologies like Ethernet, serial connection, Wi-Fi and Bluetooth. Furthermore, it benefits from the mobility of hosts, whether they are people, vehicles or data ferries.

A generic naming and addressing scheme is proposed to enable interoperability in such a heterogenous environment. Hosts are identified with URI of different schemes for each type of underlying technology. A node may have multiple identifiers depending on its connection opportunities. Data packets are encapsulated in generic bundles, which carry the necessary information for a packet to reach its destination through different communication types. Error detection is left up to higher layers; however, it may not be necessary when using error-free transmission mediums.

2.2 Challenges for DTNs

One of the most significant challenges in DTNs is routing. Routing algorithms are expected to have high delivery ratio, low latency and require less number of transmissions. The details of some routing strategies are discussed in the next section.

Although each challenge has many other challenges on its own, [4] identifies the following main categories of challenges for DTNs:

- *High Latency and Low Data Rate:* Dealing with different transmission mediums over a path, DTNs experience high latency, therefore, low data rate. Reverse path characteristics may be different from the delivery path, resulting in an asymmetric connection. DTN protocols should avoid unnecessary packet exchanges to avoid drawbacks of high latencies.

- *Disconnection:* DTNs are distinguished from other types of networks by their long disconnection periods, which are more common than connections. One of the most important causes of disconnections is the mobility in wireless networks. Disconnections due to unavailability of a transmission link are rather rare. Disconnections are best handled at network layer by routing protocols, which should try to send packets through the best available medium at the best time.

- *Long Queueing Times:* As a result of disconnections, intermediate nodes on a path need to buffer messages for long durations. This requires a routing decision to be made at each time a connection is established, because that connection might be a good candidate for any of the buffered messages.

- *Interoperability:* As discussed earlier interoperability is an important issue, because a DTN is composed of heterogeneous transmission mediums, not only in the physical layer, but also in the network layer. A good protocol design should make minimal assumptions regarding the types of communication channels throughout the path, in order to utilize as much connection as possible.

- *Security:* An end-to-end security mechanism is not appropriate for a DTN as it requires many message exchanges before actual data exchange can take place. Security concern remains as a major challenge in DTNs.
- *Low Duty Cycle Operation:* Wireless mobile nodes are restricted by their batteries or by the availability of a link to the rest of the network. Even if a packet arrives closer to a destination, it may fail to reach the destination due to these constraints. Protocols should schedule nodes duty cycles ahead of time to avoid these conditions.
- *Limited Resources:* Nodes that are participating in the network generally have limited memory and processing resources. DTN protocols should require minimal resource usage in intermediate nodes.

2.3 Related Work

This section provides an overview of routing studies in DTN literature, compares and contrasts these studies to our work.

In [6], authors formulate a DTN routing problem given the connectivity patterns of nodes, then present a comprehensive framework for evaluating and classifying different routing algorithms. They define four types of knowledge in DTNs: contacts summary, contacts information, queuing and traffic demand. Contacts summary is the aggregate statistics of contacts. Contacts information is the exact state of the contacts at any given time between any pair of nodes. Queueing information is the instantaneous buffer occupancy amount at any node at any time. Traffic demand is the information regarding the traffic in the network at any time. They also classify routing algorithms into three classes: zero knowledge, complete knowledge and partial knowledge. Zero knowledge routing algorithms use no knowledge about the network. Complete knowledge routing uses all of the knowledge types presented above. Partial knowledge routing uses one or more of the information types about the network. The study presents routing algorithms based on Dijkstra's shortest path algorithm for partial knowledge models and formulates a linear programming for complete knowledge routing.

[14] applies epidemic algorithm concept [3] to partially-connected ad hoc networks. Nodes buffer messages they received even if they do not know a route to destination. When two nodes come into contact they exchange summary vectors which is an index of the messages they buffered. By examining its neighbors summary vector a node determines the messages it does not have and requests them. In this way, a message is delivered to every contacting node and finally delivered to its destination.

Similar to epidemic routing, PROPHET [8] introduces probabilistic routing decisions. Every node maintains a delivery predictability for each node it encounters. Nodes that are frequently met have higher delivery predictability and it decreases as each node does not encounter each other for a while. Delivery predictability is also transitive meaning that a node may have a higher delivery predictability to another node through a third node which encounters both of them frequently. Epidemic forwarding of packets occur only when delivery predictability of a neighbor is higher than the node itself for a destination.

As opposed to epidemic flooding, single-copy routing schemes are presented in [12]. These strategies depend on the fact that every local progress will lead a packet to its destination. In the most basic strategy, direct transmission, source node forwards the packet only if it encounters the destination. Randomized routing algorithm forwards the copy to a neighbor with probability $p > 0$. In utility-based routing, nodes compute a utility value for each destination depending on their location and last encounter time. Nodes forward a packet only if a neighbor has a higher utility value to destination.

Another single-copy technique is suggested in [1]. Using only the average inter-contact time estimates between nodes, a two-hop relay strategy is extended to a recursive multi-hop relay strategy. The routing algorithm aims to minimize delivery time in case of independent exponential pairwise inter-contacts. It is also loop-free, making it suitable to be used in DTNs.

Spray and wait routing [13] aims to compromise between epidemic routing and single-copy routing by limiting the number of copies a packet can have. It reduces the number of transmissions with respect to epidemic routing and achieves a better delivery ratio than single-copy schemes. Spraying phase consists of distributing L number of copies of a packet to the network. Every node transfers $\lfloor n/2 \rfloor$ copies of its n copies to a neighbor and reserves remaining $\lceil n/2 \rceil$ for future encounters. Wait phase begins when a node has only a single copy. The node forwards the copy to a neighbor only if it is the destination.

[11] discusses routing in networks with predictable mobility. Nodes follow a certain deterministic trajectory that is defined as a function of time. Given any time instant, network graph can be constructed from the location of nodes. A space-time graph is the combination of different network graphs at different time instants as a single large graph. There are two kinds of links in the space-time graph. Temporal links connect the different versions of the same node at different time instants. Spatial links connect different nodes at different time instants. By defining a distance function over these links, routing is performed to minimize the delivery time.

In the study presented in [9], the authors focus on scalability issues in routing for DTNs. Under the same mobility model defined in [11], they propose DTN Hierarchical Routing (DHR), which applies hierarchical routing to multiple levels of a multi-level clustered network. Number of levels used in the routing algorithm makes the algorithm scalable and is bounded by aggregate connection information of the network. They also use compression methods to summarize contact information. In another study [10], same authors examine optimality issue for probabilistic forwarding protocols. Authors define a 1-hop delivery probability metric and extend it to K-hop. The forwarding rule defined using the metric is formulated inside an optimal stopping rule problem to find optimal routes.

Practical concerns about DTN routing is examined in [7]. Presented protocol depends on an estimate of how long a message waits on a host until transmission to the next hop. The estimate is calculated from contact history, therefore does not use global contact information. Forwarding is performed when a neighbor is estimated to be closer to destination. Network topology is distributed using epidemic flooding of link-state packets. Routing table is recomputed every time a connection is established. Less frequently encountered contacts are utilized in this way.

According to the classification in [6] our study uses only contact information. We do not utilize queueing or traffic demand for routing decisions; therefore, our approach is a partial knowledge routing method. As suggested in the study, we use routing algorithms based on Dijkstra's algorithm. Using contact information eliminates the need for probabilistic routing decisions.

We propose a single copy routing scheme, as opposed to multi-copy schemes like epidemic routing or spray-and-wait. On the other hand, our algorithms are similar to epidemic routing in terms of delivery time; both are optimal strategies guaranteeing the earliest delivery.

Rather than using a mobility model, we use connection models as an abstraction from mobility. We focus on how connections are established without considering the reason behind whether it is mobility, availability or interference. Therefore, our study is different from [11] and [9].

To the best of our knowledge, we are the first to address minimum hop and earliest delivery objectives together in a routing algorithm in the context of DTNs.

Chapter 3

Routing for Periodic Connections

3.1 Problem Statement

In our delay tolerant network model we assume connection opportunities arise periodically between any two nodes. This assumption is realistic due to the fact that interactions between entities are periodic in nature. We meet our colleagues every morning, friends from other departments during lunch time and our family in the evening. Students, similarly, come across each other during breaks. Vehicles in public transportation arrive at stations periodically; sensors periodically transmit data to base stations; and so on.

We identify three cases of connection models of increasing complexity. In the first model, contacts occur only once in the future. The second model introduces periodic connections with negligible connection durations. Finally, third model considers both connection and disconnection durations separately for periodic connections.

We represent the network as an undirected connected graph, where there is an edge between two nodes if there is a possibility of connection in-between, and apply shortest path routing with some modifications to meet our requirements.

3.2 Dijkstra's Shortest Path Algorithm

We first present Dijkstra's shortest path algorithm as presented in [2]. For network models with different properties we only define a custom relax method that relaxes edges according to its own requirements. Hence, in most of our algorithms presented in this chapter, the main Dijkstras shortest path algorithm is left unmodified except trivial argument modifications for different network models.

Algorithm 1 shows the initialization procedure for vertices in the graph. Distance t to every vertex v from source s is defined to be ∞ and parent of each vertex is initially null. By definition, distance to source is 0.

Algorithm 1 Initialization of vertex properties.

```
INITIALIZE-SINGLE-SOURCE($G, s$)
1: for all $v \in V[G]$ do
2:     $t[v] \leftarrow \infty$
3:     $\pi[v] \leftarrow nil$
4: end for
5: $t[s] \leftarrow 0$
```

Algorithm 2 relaxes a given edge (u, v) with weight w, if necessary. When a shorter path to v from u is found, its distance is updated accordingly and its parent is set to be u.

Algorithm 2 Original vertex relaxing mechanism in Dijkstra.

```
RELAX($u, v, w$)
1: if $t[v] > t[u] + w(u, v)$ then
2:     $t[v] \leftarrow t[u] + w(u, v)$
3:     $\pi[v] \leftarrow u$
4: end if
```

Algorithm 3 is the Dijkstra's single source shortest path algorithm. Vertices are initialized and inserted into a min-heap. Since the source has the minimum distance 0, where all other vertices have ∞, it is first extracted from the heap. Then we relax its neighbors using edge weights. At each iteration, the minimum distance vertex is extracted; therefore all vertices are assigned their shortest possible distance to the source.

It should be noted that, in the algorithms presented, any assignment to $t[u]$ value for any node u is actually a Decrease-Key$(Q, u, t[u])$ operation on the min-heap Q.

Algorithm 3 Dijkstra's algorithm.

```
Dijkstra($G, w, s$)
1: Initialize-Single-Source($G, s$)
2: $Q \leftarrow V[G]$
3: while $Q \neq \emptyset$ do
4:     $u \leftarrow$ Extract-Min(Q)
5:     for all $v \in Adj[u]$ do
6:         Relax($u, v, w$)
7:     end for
8: end while
```

3.3 Scheduled Connections

We begin with a simple connection model in which connections occur only once according to a predetermined schedule. We can calculate perfect shortest routes given source, destination, packet generation/arrival time and connection establishment time between nodes. We further simplify the model by neglecting contact durations. An established connection is assumed to be lost as soon as all the necessary packets have been exchanged between two nodes.

We now let edge weights, $w(u, v)$, represent the contact times between nodes in a mobile wireless network environment. Figure 3.1 shows an example network. The link between Nodes 0 and 1 will be available at $t = 2$, the link between Nodes 0 and 2 will be available at $t = 5$, and so on.

In such a scenario, the shortest path is redefined to be the shortest path in time with non-decreasing edge weights; since, a decreasing edge weight from one link to another would mean a missed contact opportunity. This implies that if a non-decreasing weight path does not exist for a vertex, that vertex is not reachable from source.

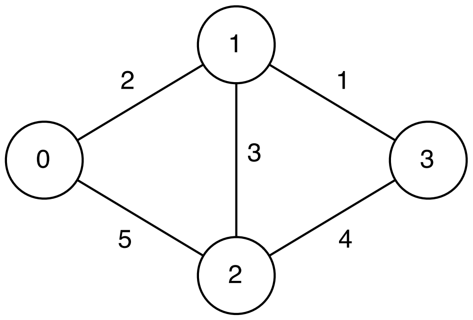


Figure 3.1: Example Network 1.

Figure 3.2 shows the routing tree of Node 0 at $t = 0$. Node 1 is reachable at $t = 2$. Although Node 2 is directly reachable at $t = 5$, there exist a shorter path where packets destined to Node 2 can be delivered at $t = 3$ over Node 1. The link between Nodes 1 and 3, available only at $t = 1$, cannot be utilized since packets arrive at Node 1 at $t = 2$, earliest. Therefore, shortest path to Node 3 is over Node 2.

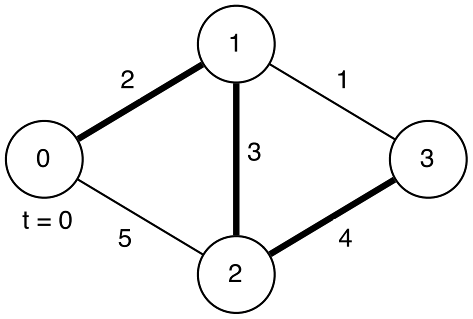


Figure 3.2: Routing tree of Node 0 at $t = 0$ in Example Network 1.

Shortest distance $t[u]$ to a vertex u from the source is also redefined to be the time a packet generated at $t = 0$ is delivered. In the Example Network 1, shortest distances to Nodes 1, 2 and 3 from Node 0 are 2, 3 and 4, respectively.

The refined version of Algorithm 2 to work under these conditions is given in Algorithm 4. The condition in the if statement checks whether the edge is in non-decreasing order in the path. Edge weights are not accumulated as in the original version, since they are absolute contact times.

Algorithm 4 Vertex relaxing for scheduled connections.

```
RELAX($u, v, w$)
1: if $w(u, v) \geq t[u]$ and $t[v] > w(u, v)$ then
2:     $t[v] \leftarrow w(u, v)$
3:     $\pi[v] \leftarrow u$
4: end if
```

3.3.1 Routing at $t > 0$

So far we have discussed how to compute a routing tree at $t = 0$. Algorithm 5 is a simple modification to INITIALIZE-SINGLE-SOURCE(G, s) procedure given in Algorithm 1 to enable shortest route computation at an arbitrary time t.

Algorithm 5 Initialization of Dijkstra's algorithm with time t

```
INITIALIZE-SINGLE-SOURCE($G, s, t$)
1: for all $v \in V[G]$ do
2:     $t[v] \leftarrow \infty$
3:     $\pi[v] \leftarrow nil$
4: end for
5: $t[s] \leftarrow t$
```

The initial distance to source is set to given t. The source node remains the first node to be extracted from the min-heap, since t is smaller than the initial distances of all other nodes, ∞. Relax method in Algorithm 4 considers only edges that have contact times in now or future; therefore, non-decreasing edge weight path property is still maintained.

The signature of the main routing procedure Dijkstra should be updated to accept the time t at which the routing tree will be computed as DIJKSTRA(G, w, s, t).

Figure 3.3 shows routing tree for Node 0 at $t = 3$. Only Node 2 is reachable at $t = 5$, as others' contact opportunities are lost either at $t = 3$ or $t = 5$.

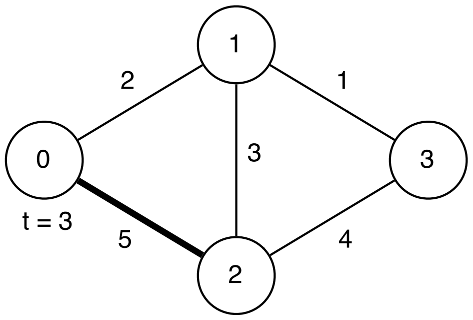


Figure 3.3: Routing tree of Node 0 at $t = 3$ in Example Network 1.

3.4 Scheduled Periodic Connections

In this section, we extend our network model to utilize periodic connections which occur according to an exact schedule defined earlier. We first examine the case where contact durations are insignificant and then consider periodic connections with significant contact times.

3.4.1 Insignificant Contact Durations

We assign each edge (u, v) a period $T(u, v)$ after which connection is reestablished. In this model, no vertex is unreachable, because a connection will be available after at most $T(u, v)$ time. As stated earlier, we assume the graph representing the possible connections among nodes is a connected graph.

Figure 3.4 shows an example network with periodic connections. Each link is tagged with pair (w, T) where w represents initial wait time of a link after which connections begin to occur with period T. For example, the link between Nodes 0 and 1 becomes available at $t = 2$ and then becomes up again at every 3 units of time, $t = \{5, 8, 11, 14, \ldots\}$. The following function gives the k^{th} connection time for a link (u, v):

$$f(k, u, v) = w + k \cdot T(u, v)$$

Routing tree for Node 0 at $t = 0$ is identical to the one depicted on Figure 3.2, because exact schedules on scheduled connections case corresponds to initial

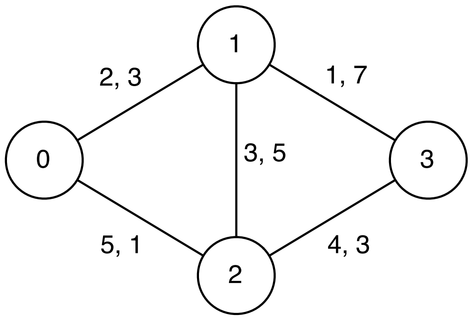


Figure 3.4: Example Network 2 with initial wait times and connection periods.

wait times in periodic connections case. However, routing tree at $t = 3$, shown in Figure 3.5, is different than that on Figure 3.3. Time signatures on edges denote when a link becomes available after one of its nodes receive a packet from Node 0. All of the nodes are reachable in this case, as periodic connections are utilized.

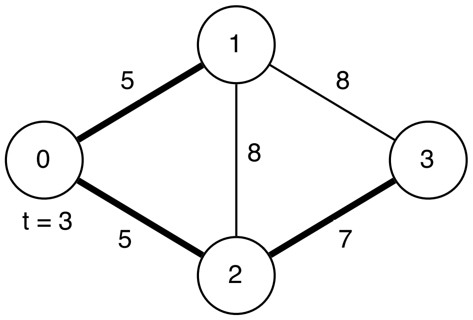


Figure 3.5: Routing tree for Node 0 at $t = 3$ in Example Network 2.

Algorithm 6 extends the $Relax(u, v, w)$ given in Algorithm 4 with another input for connection periods. An initial statement calculates the first connection time after $t = t[u]$ using $T(u, v)$. The first condition in the if statement of Algorithm 4 is no longer needed since we assure the weights to be non-decreasing with the initial calculation.

Throughout the execution of the algorithm, whenever $w(u, v)$ is assigned, it is assigned the next connection time of the corresponding link.

Algorithm 6 Vertex relaxing for insignificant contact durations.

```
RELAX($u, v, w, T$)
1: if $w(u,v) < t[u]$ then
2:     $w(u,v) \leftarrow w(u,v) + T(u,v) \cdot \left\lceil \frac{t[u]-w(u,v)}{T(u,v)} \right\rceil$
3: end if
4: if $t[v] > w(u,v)$ then
5:     $t[v] \leftarrow w(u,v)$
6:     $\pi[v] \leftarrow u$
7: end if
```

3.4.2 Significant Contact Durations

So far we have assumed connections remain available for a very small amount of time, which is negligible. This assumption is not very realistic as connection times are comparable to disconnection times. To alleviate this problem, we define $T_{ON}(u,v)$ and $T_{OFF}(u,v)$ to be connection and disconnection durations between nodes u and v respectively. In this case the original period definition becomes $T(u,v) = T_{ON}(u,v) + T_{OFF}(u,v)$. Connection and disconnection phases alternate after initial wait time.

Figure 3.6 shows a network with significant contact durations. Links are labeled with w, (T_{ON}, T_{OFF}) values. A link first goes up at the end of its initial wait time, w. The connection lasts for T_{ON} time after which a disconnection occurs for T_{OFF} time. *ON* and *OFF* states follow each other after that point. For instance, the link between Nodes 0 and 1 becomes available at $t = 2$, disconnection occurs at $t = 4$ and next connection period begins at $t = 5$. The following two functions give the beginning of the k^{th} connection and disconnection periods, respectively, for a link (u,v):

$$f_{ON}(k,u,v) = w + k \cdot T(u,v)$$

$$f_{OFF}(k,u,v) = w + T_{ON}(u,v) + k \cdot T(u,v)$$

The routing tree for Node 0 at $t = 3$ is illustrated in Figure 3.7. The current time is the t. For each link in the figure, the next time interval just after or including the current time during which the link is on is given as the link label.

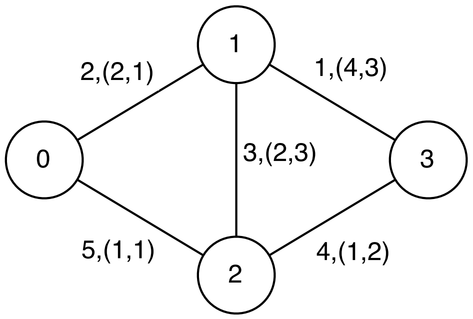


Figure 3.6: Example Network 3 with initial wait times, connection and disconnection periods.

When a node u receives a packet at $t[u]$, it can forward it to node v at $t[v] = t[u]$ if they are connected at that time (i.e., the link is up at that time). Otherwise, the node u should wait until the beginning of the next connection to forward the packet, as in the previous cases. In this example shown in Figure 3.7, every destination gets the packet at $t = 3$ as all of the links are on at that time.

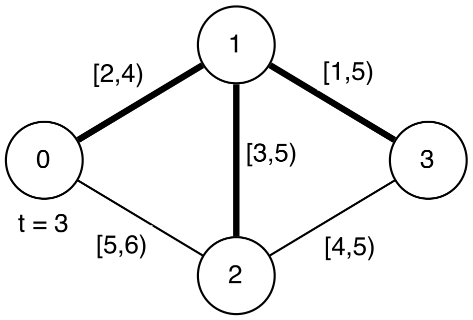


Figure 3.7: Routing tree for Node 0 at $t = 3$ in Example Network 3.

Algorithm 7 defines the relax method for significant contact durations model. One of the drawbacks of it is the fact that it brings a restriction on the initial wait times of links. An initial wait time of a link should satisfy the inequality: $w(u, v) \leq T_{OFF}(u, v)$. Otherwise, the algorithm fails to predict the beginning time of the first ON period after the initial wait time. This problem is addressed in Algorithm 8 by keeping the beginning time of the last ON period in $w(u, v)$ and calculating ON durations accordingly.

Algorithm 7 Vertex relaxing for significant contact durations.

```
RELAX($u, v, w, T_{ON}, T_{OFF}$)
 1: if $w(u,v) \leq t[u]$ then
 2:     $w(u,v) \leftarrow w(u,v) + T(u,v) \cdot \left( \left\lfloor \frac{t[u]-w(u,v)}{T(u,v)} \right\rfloor + 1 \right)$
 3: end if
 4: if $t[v] > t[u]$ and $t[u] < w(u,v) - T_{OFF}(u,v)$ then
 5:     $t[v] \leftarrow t[u]$
 6:     $\pi[v] \leftarrow u$
 7: else if $t[v] > w(u,v)$ then
 8:     $t[v] \leftarrow w(u,v)$
 9:     $\pi[v] \leftarrow u$
10: end if
```

3.5 Min-hop Earliest Delivery (MHED) Routing

So far, we have discussed routing to achieve earliest delivery, routes that are shortest in time. In this section, we are going to define shortcomings of earliest delivery routing and introduce min-hop earliest delivery (MHED) routing.

Suppose we have the following network in Figure 3.8 with scheduled connections.

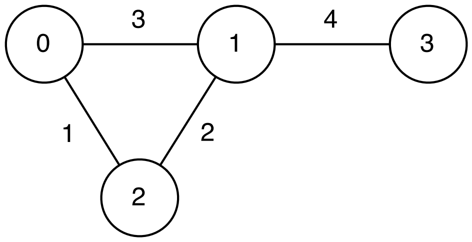


Figure 3.8: Example Network 4.

Earliest delivery algorithm run in Node 0 would find the route to Node 3 as [0, 2, 1, 3] with $t = 4$ being the delivery time. However, there exists a shorter route if we consider hop-count as a secondary metric. The route [0, 1, 3] is also shortest in delivery time ($t = 4$); moreover, it is shorter than the former path in terms of hop count; $n = 3$ vs $n = 2$.

Algorithm 8 Vertex relaxing with arbitrary initial wait times.

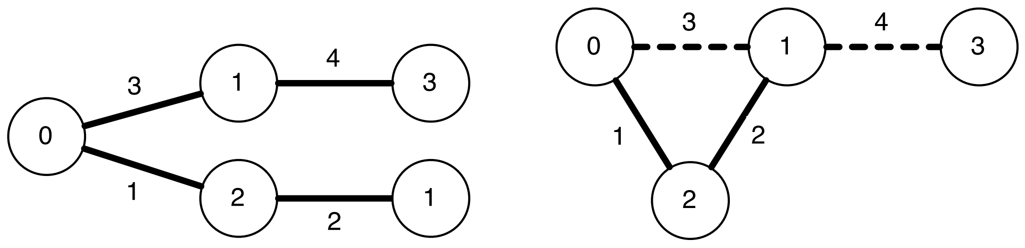
```
RELAX($u, v, w, T_{ON}, T_{OFF}$)
1: if $w(u,v) + T(u,v) \leq t[u]$ then
2:     $w(u,v) \leftarrow w(u,v) + T(u,v) \cdot \left\lfloor \frac{t[u]-w(u,v)}{T(u,v)} \right\rfloor$
3: end if
4: if $w(u,v) \leq t[u]$ then
5:     if $t[v] > t[u]$ and $t[u] < w(u,v) + T_{ON}(u,v)$ then
6:         $t[v] \leftarrow t[u]$
7:         $\pi[v] \leftarrow u$
8:     else if $t[v] > w(u,v) + T(u,v)$ then
9:         $t[v] \leftarrow w(u,v) + T(u,v)$
10:        $\pi[v] \leftarrow u$
11:    end if
12: else if $t[v] > w(u,v)$ then
13:     $t[v] \leftarrow w(u,v)$
14:     $\pi[v] \leftarrow u$
15: end if
```

The previous algorithms fail to detect such a route because they make greedy choices based only on delivery time. The route to Node 1 from Node 0 is over Node 2 as it reaches Node 1 in earliest time. Once we set Node 2 as the parent of Node 1, all routes descending from Node 1 in the routing tree pass over Node 2 independent of their time. However, we can choose any other route to Node 1, satisfying non-decreasing edge weight path property, to route packets destined to nodes that are children of Node 1 in the original routing tree. In this case, path [0, 1] with $t = 3$ satisfies non-decreasing edge weight path property for path [1, 3] with $t = 4$; therefore, we can choose path [0, 1, 3] to route packets from Node 0 to Node 3. Since we are trying to find min-hop routes, paths we choose should be shorter in terms of hop count.

3.5.1 Routing Tree

In routing tree concept, every node has a parent node through which it receives its packets. In min-hop earliest delivery routing, routing tree concept is different: every edge through which packets are routed has a parent edge. This causes a node to appear in different branches of the routing tree; on the other hand, an edge can appear only in a single branch.

Figure 3.9 shows the routing tree of min-hop earliest delivery routing for Example Network 4. There are two branches in the routing tree as shown in Figure 3.9a. Node 1 appears in top branch as an intermediate node in route to Node 3 and as a terminal node in bottom branch. Figure 3.9b presents an alternative view showing routing tree inside the network graph.



(a) Routing tree (b) Alternative view

Figure 3.9: Min-hop Earliest Delivery Routing Tree for Example Network 4.

3.5.2 Solution

3.5.2.1 Data Structures

The solution to MHED routing comes from the observation that we should utilize paths that are shorter than the earliest delivery path in hop count. Among those paths having the same hop count, we should still choose the path providing the earliest delivery; so that, resulting path is the shortest. We are not interested in paths that are longer than the earliest delivery path in hop count, because earliest delivery path is already shorter.

Another important observation is the fact that any-number-of-hop paths to a vertex may become the shortest path for a neighbor vertex. This observation is illustrated in Figure 3.10.

Assuming Node 0 is the source, shortest path to Node 4 is [0, 1, 2, 4] with $h = 3$ and $t = 3$. ED algorithm would route all of the packets to Nodes 5 and

6 through this path to Node 4; however, there are shorter alternatives. Shortest path to Node 5 is [0, 3, 4, 5] ($h = 3$, $t = 5$) and to Node 6 is [0, 4, 6] ($h = 2$, $t = 7$). Shortest paths to both Nodes 5 and 6 is still through Node 4, but uses different paths with different number of hops until Node 4. In this example, all possible number-of-hop paths (1-hop, 2-hop, 3-hop) to Node 4, are utilized as a shortest path to some node.

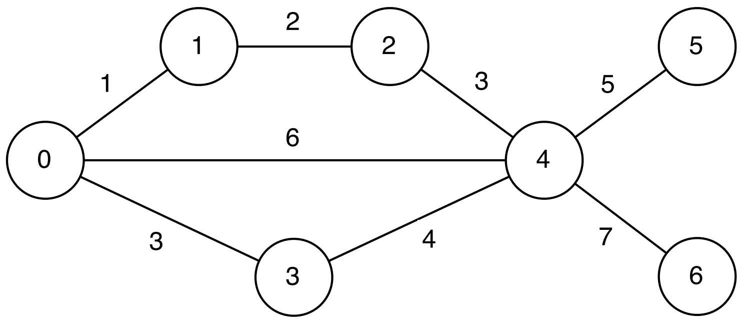


Figure 3.10: Example Network 5.

As a result, our solution should keep track of n-hop earliest delivery routes to all nodes where $n \leq$ hop count of the earliest delivery path. Initialization of data structures of MHED algorithm is given in Algorithm 9.

Algorithm 9 Initialization procedure for MHED algorithm.

```
INITIALIZE-MHED($G, s, t$)
1: for all $v \in V[G]$ do
2:     $hop[v] \leftarrow 0$
3:     for $n = 0$ to $|V| - 1$ do
4:         $t[v][n] \leftarrow \infty$
5:         $\pi[v][n] \leftarrow nil$
6:     end for
7: end for
8: $t[s][0] \leftarrow t$
```

$hop[v]$ is the hop count of the earliest delivery path to a vertex. $t[v][n]$ and $\pi[v][n]$ respectively hold the earliest delivery time of and parent vertex for an n-hop path to vertex v. Therefore, $t[v][hop[v]]$ and $\pi[v][hop[v]]$ correspond to $t[v]$ and $\pi[v]$, respectively in earlier initialization procedure, INITIALIZE-SINGLE-SOURCE(G, s, t) defined in Algorithm 5. Lastly, earliest delivery time of source node at 0 hops is set to t.

Size of $t[v]$ and $\pi[v]$ is $|V|-1$, since shortest paths are simple paths and length of a simple path can be at most $|V|-1$. Although these structures allocate $|V|-1$ space, at most $|Adj[v]|$ of them, one for each neighbor, will be used for a vertex v.

3.5.2.2 Main Algorithm

In the main algorithm, we should traverse the graph and fill the data structures at the nodes with correct values and, therefore, obtain the routing tree. The objective is to find the n-hop earliest delivery paths to each vertex. We already know that previous algorithms based on Dijkstra's algorithm achieve the earliest delivery. Now we also keep track of hop counts together with earliest delivery times. We modify min-heap structure to keep hop-vertices, a two-tuple (n, u), where n is the hop count and u is the vertex. Comparison between two hop-vertices is done in the order of hop and delivery time to vertex at n hops. Following function is the formal definition of *min* function used by min-heap.

$$min((n,u),(m,v)) = \begin{cases} (n,u) & \text{if } n < m \\ (n,u) & \text{if } n = m \text{ and } t[u][n] \leq t[v][m] \\ (m,v) & \text{if } n = m \text{ and } t[u][n] > t[v][m] \\ (m,v) & \text{if } n > m \end{cases}$$

min function implies that all n-hop hop-vertices will be extracted from the heap before an (n+1)-hop hop-vertex is extracted.

Algorithm 10 presents MHED algorithm. The fundamental difference from earliest delivery algorithm in Algorithm 3 is the structure of the heap. Since we do not know which hop-vertices will be used, we initialize an empty min-heap and add them in the relax procedure.

The relax function in Algorithm 11, takes hop count, n, after which edge (u, v) is reached. The first condition in if statement checks whether this edge satisfies non-decreasing edge weight path property after n-hop path to vertex u.

Algorithm 10 MHED algorithm based on Dijkstra's algorithm.

```
MHED($G, w, s, t$)
1: INITIALIZE-MHED($G, s, t$)
2: $Q \leftarrow$ MAKE-HEAP()
3: while $Q \neq \emptyset$ do
4:     $(n, u) \leftarrow$ EXTRACT-MIN(Q)
5:     for all $v \in Adj[u]$ do
6:         RELAX($u, v, w, n$)
7:     end for
8: end while
```

The second condition checks whether this edge constitutes a shorter path than up to n-hop paths to vertex v. Because of the way hop-vertices are extracted from the min-heap, $hop[v]$ is never greater than $n+1$ and converges to path length of earliest delivery path throughout the execution. If both of these conditions hold, it means edge (u, v) forms an (n+1)-hop path to v. If $hop[v] < n+1$, we need to insert a new hop-vertex to the heap, otherwise the hop-vertex is already in the heap and we need to perform a DECREASE-KEY operation. DECREASE-KEY, which was omitted in previous versions, is explicitly written for clarity.

Algorithm 11 Vertex relaxing for MHED.

```
RELAX($u, v, w, n$)
1: if $w(u,v) \geq t[u][n]$ and $t[v][hop[v]] > w(u,v)$ then     ▷ $hop[v] \leq n+1$
2:     $t[v][n+1] \leftarrow w(u,v)$
3:     $\pi[v][n+1] \leftarrow u$
4:     if $hop[v] < n+1$ then
5:         $hop[v] \leftarrow n+1$
6:         INSERT($Q, (n+1, v)$)
7:     else                                                      ▷ $hop[v] = n+1$
8:         DECREASE-KEY($Q, (hop[v], v)$)
9:     end if
10: end if
```

Note that inserted hop-vertices can have a hop count of $n+1$. Given the property that all n-hop hop-vertices are extracted before (n+1)-hop ones, it can be concluded that only n-hop and (n+1)-hop hop-vertices can occur in the heap at the same time. Therefore, the hop difference between minimum and maximum hop hop-vertex is at most 1.

This property is the same as the property of a BFS queue in operation; therefore, we can substitute min-heap with a FIFO queue. Algorithm 12 presents MHED algorithm based on BFS. The only difference from the Dijkstra based version is the use of queue operations instead of min-heap operations.

Algorithm 12 MHED algorithm based on BFS.

```
MHED($G, w, s, t$)
1: INITIALIZE-MHED($G, s, t$)
2: $Q \leftarrow$ MAKE-QUEUE()
3: while $Q \neq \emptyset$ do
4:     $(h, u) \leftarrow$ DEQUEUE(Q)
5:     for all $v \in Adj[u]$ do
6:         RELAX($u, v, w, h$)
7:     end for
8: end while
```

Vertex relaxing requires some modifications. Since we are not using a min-heap, we can omit DECREASE-KEY operation and use ENQUEUE instead of INSERT. Vertex relaxing for BFS based MHED is given in 13.

Algorithm 13 Vertex relaxing for BFS based MHED.

```
RELAX($u, v, w, n$)
1: if $w(u,v) \geq t[u][n]$ and $t[v][hop[v]] > w(u,v)$ then        $\triangleright$ $hop[v] \leq n+1$
2:     $t[v][n+1] \leftarrow w(u,v)$
3:     $\pi[v][n+1] \leftarrow u$
4:     if $hop[v] < n+1$ then
5:         $hop[v] \leftarrow n+1$
6:         ENQUEUE($Q, (n+1, v)$)
7:     end if
8: end if
```

For the sake of simplicity, we omit insignificant contact durations connection model and give relax procedure of MHED for significant contact durations. The main difference from the previous versions is the use of a temporary variable w instead of actual $w(u, v)$. The reason is the fact that a vertex is enqueued multiple times for each hop, therefore, its edges are traversed multiple times. If we were to assign connection times to $w(u, v)$, values might not be correct for another time we go over the edges. By using the temporary w, the next contact time is calculated from the initial $w(u, v)$ every time, regardless of how many times a vertex is dequeued.

A boolean flag *relax* is set when a shorter path to v is found. If *relax* is true, we check whether an enqueue operation is necessary for the vertex.

Algorithm 14 Vertex relaxing of MHED with significant contact durations.

```
RELAX($u, v, w, T_{ON}, T_{OFF}$)
 1: $w \leftarrow w(u, v)$
 2: $relax \leftarrow false$
 3: if $w + T_{ON}(u, v) \leq hopt[u][n]$ then
 4:     $w \leftarrow w + T(u, v) \cdot \left\lfloor \frac{hopt[u][n] - w}{T(u,v)} \right\rfloor$
 5: end if
 6: if $w \leq hopt[u][n]$ then
 7:     if $t[v][hop[v]] > hopt[u][n]$ and $hopt[u][n] < w + T_{ON}(u, v)$ then
 8:         $t[v][n + 1] \leftarrow hopt[u][n]$
 9:         $\pi[v][n + 1] \leftarrow u$
10:         $relax \leftarrow true$
11:     else if $t[v][hop[v]] > w + T(u, v)$ then
12:         $t[v][n + 1] \leftarrow w + T(u, v)$
13:         $\pi[v][n + 1] \leftarrow u$
14:         $relax \leftarrow true$
15:     end if
16: else if $t[v] > w$ then
17:     $t[v][n + 1] \leftarrow w$
18:     $\pi[v][n + 1] \leftarrow u$
19:     $relax \leftarrow true$
20: end if
21: if $relax$ and $hop[v] < n + 1$ then
22:     $hop[v] \leftarrow n + 1$
23:     ENQUEUE($Q, (n + 1, v)$)
24: end if
```

Because of the difference in routing tree structure in MHED routing, we provide PRINT-PATH procedure in Algorithm 15. It prints the n-hop path from source s to vertex v by recursively printing path to parent of v at $n - 1$ hops. In order to get the min-hop earliest delivery route to a vertex v, PRINT-PATH($G, s, v, hop[v]$) should be called.

Algorithm 15 Route printing for MHED.

```
PRINT-PATH($G, s, v, n$)
1: if $s = v$ then
2:     print $s$
3: else if $\pi[v][n] = nil$ then
4:     print “no path from” $s$ “to” $v$ “at” $n$ “hops exists.”
5: else
6:     PRINT-PATH($G, s, \pi[v][n], n-1$)
7:     print $v$
8: end if
```

3.5.3 Running Time Analysis of MHED

We begin running time analysis of MHED by inspecting the number of iterations of the while loop in line 3. The while loop runs until there are no hop-vertices left in the queue; therefore, the iteration count is equal to the number of enqueue and dequeue operations. As discussed earlier, there can be at most $|Adj[v]|$ number of n-hop paths to a vertex v, if every neighbor provides a path with different number of hops from the source. Sum of $|Adj[v]|$ for all vertices v gives the number of edges, $|E|$. As a result, there are $|E|$ number of hop-vertices, enqueue and dequeue operations.

The for loop in line 5 executes $|E|$ times, each execution iterates through $|Adj[u]|$ number of edges. A total of $|V|$ executions of the loop results in $|E|$ iterations; therefore, the iteration count in $|E|$ executions is $\frac{|E|^2}{|V|}$. Relax method runs in constant time, since every statement including the enqueue operation on FIFO queue takes constant time.

The initialization procedure takes $O(V^2)$ time, therefore the running time of the BFS based MHED algorithm is $O(V^2 + \frac{E^2}{V})$, which is always between $O(V^2)$ and $O(V^3)$ for an arbitrary network.

The Dijkstra based MHED algorithm uses insert and extract-min operations on a min heap instead of enqueue and dequeue operations on a FIFO queue. There are also decrease-key operations inside the relax method.

Using a binary heap, insert takes $O(E)$, extract-min takes $O(ElgE)$ and decrease-key takes $O(\frac{E^2lgE}{V})$ time, making the running time of the overall algorithm $O(\frac{E^2lgE}{V})$. By using a Fibonacci heap, extract-min operations take $O(ElgE)$ amortized time and running time of decrease-key operations decrease to $O(\frac{E^2}{V})$ amortized time. Taking the initialization procedure into account, the running time of the MHED algorithm based on Dijkstra using Fibonacci heap is $O(V^2 + \frac{E^2}{V} + ElgE)$, which is equal to $O(V^2 + ElgE)$ for sparse networks.

We conclude that BFS based approach runs asymptotically faster than Dijkstra based approach.

Chapter 4

Experiments and Evaluation

In this chapter, we present experiment results of proposed DTN routing algorithms and evaluate the findings. For this purpose, we first define the simulation environment, properties and parameters of the simulations, then discuss the results and their implications.

4.1 Simulation Properties

We performed simulations to find statistical properties of our routing algorithms for different parameter values. Transmission count, average and maximum path length to destination and routing tree stability statistics are measured. Implementation is done on Java platform using custom graph and routing code. Simulations are run on a Linux machine with a quad-core AMD 64-bit processor and 4 GB memory, though minimum requirements for the implementation are much lower.

A simulation is performed as follows: First, a random graph is generated with a given node count, network density and a connection model. Then, measures are taken over a time interval of 100 units separately for each node as the source. Similarly, every other node becomes the destination when a destination node is needed. Results at each step are averaged. The overall procedure is repeated 30 times to ensure enough randomness and to get statistically enough number of samples.

We produce random graphs given vertex count and graph density. Since DTNs are connected graphs, we must ensure our random graph to be connected. To achieve this, we keep two sets of vertices: connected and disconnected. Until the graph becomes connected, we choose one random node from the set of connected vertices, one from the set of disconnected and connect them. When all of the vertices are connected, we get a connected graph with $|V| - 1$ number of edges. Then, we add edges by randomly choosing two vertices, until the density constraint is satisfied. Note that the produced graph cannot have a lower density than the density of a connected graph with given vertex count and minimum number of edges, $\frac{|V|-1}{|V|^2}$. The properties of a graph produced with this method is random enough for our purposes.

For some of the experiments, we considered two scenarios with different connection models. Scenario 1 simulates a DTN inside a campus. Nodes represent college students, attending 50 minute-long lectures and having 10 minute breaks. We assume that connections occur mainly during break times; though we take into account slightly shorter and longer connections and disconnections. Therefore, T_{ON} ranges between 5 and 20 minutes and T_{OFF} ranges between 15 and 60 minutes. The key property of this scenario is the fact that connections take shorter time than disconnections.

The second scenario is a more homogenous environment, where connection and disconnection periods are the same. Scenario 2 simulates connection properties of a public transportation network. Nodes represent both vehicles (bus, subway etc.) and passengers. Connections occur either in stations or during travel between people and vehicles. We assume waiting time for a vehicle and travel time are similar; therefore, T_{ON} and T_{OFF} values have a range of 10 to 30 minutes.

For both scenarios we use an initial wait time between minimum and maximum of the connection or disconnection times. For Scenario 1, it is 5 to 60 minutes; for Scenario 2, it is 10 to 30 minutes. For practical purposes, we scale down the time and let one time unit of simulation represent 5 and 10 minutes of real time respectively in Scenarios 1 and 2. Actual values for each connection are drawn from a uniform distribution between determined intervals. Unless otherwise stated, Scenario 1 is used in the experiments.

In the experiment results, ED refers to the earliest delivery algorithm based on Dijkstra's algorithm in Algorithm 3 using vertex relaxing procedure in Algorithm 8, MHED refers to min-hop earliest delivery algorithm based on BFS, presented in Algorithm 12 using vertex relaxing procedure in Algorithm 14.

4.2 Results and Evaluation

This section presents evaluation of experiment results.

4.2.1 Transmission Count

Epidemic routing is one of the earliest and most popular routing algorithms for DTNs. Source node forwards its packet to every node it encounters and those nodes forward the packet to their encountered nodes in turn, delivering the packet to destination at some time in future. This scheme is aptly called epidemic flooding. Although ED and MHED algorithms are single copy routing strategies utilizing contact information, they are comparable to epidemic routing in terms of delivery time. All three methods guarantee earliest delivery.

Figure 4.1 shows average transmission count to deliver a single packet to a destination for epidemic, ED and MHED routing in a network with a density of 0.2. For epidemic routing, the network does not have the knowledge of packet reaching its destination; therefore, nodes continue to transmit the packet to other nodes until every node in the network has a copy. To have a better comparison, we count only the number of transmissions until the packet reaches the destination. Transmission count of epidemic routing increases linearly with the number of nodes in the network. It delivers the packet to 90% of the network with a node count of as low as 50. For higher node counts, delivery ratio increases to 98% on the average. ED and MHED routing, on the other hand, achieves earliest delivery with very low transmission counts, MHED performing slightly better than ED.

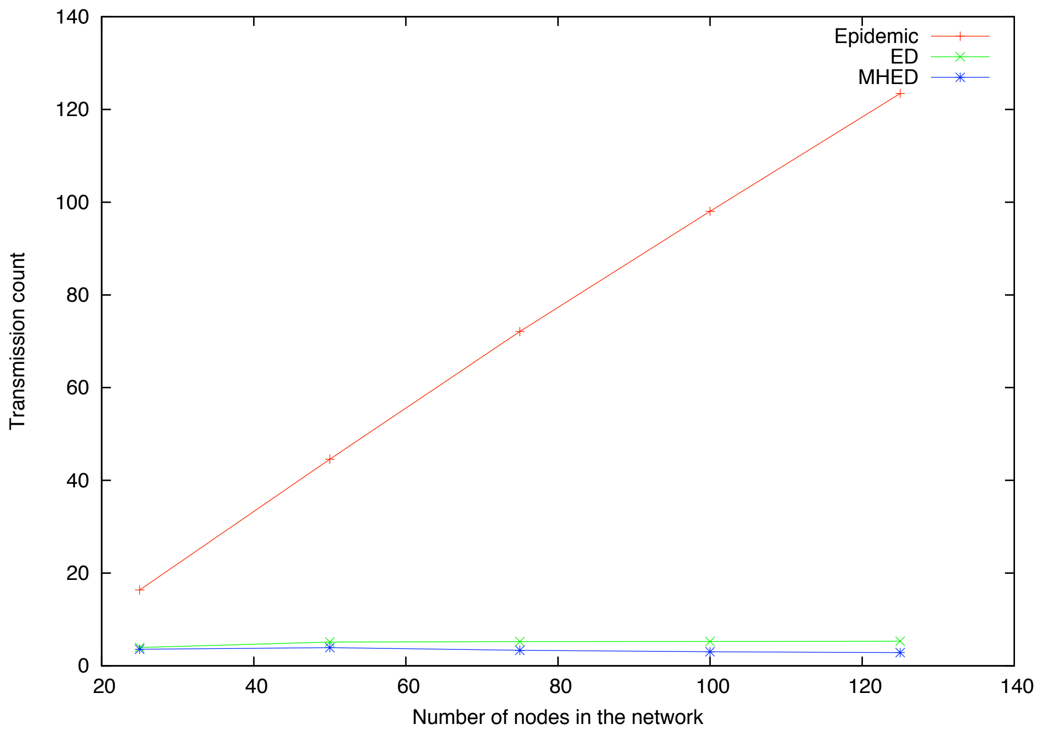


Figure 4.1: Average transmission counts to deliver a single packet to a destination in a network with $density = 0.2$.

We can conclude that ED and MHED successfully uses contact information to deliver packets with much lower number of transmissions compared to epidemic routing.

4.2.2 Path Length

In this experiment, we measure average and maximum path lengths (hop counts) to a destination in a 100 node network for Scenarios 1 and 2. We expect to find MHED using shorter paths than ED, as one its the objectives is to minimize hop count to destination.

Figure 4.2 presents average path length results. For each scenario, MHED finds shorter paths than ED as expected. For Scenario 1, maximum difference between both algorithms appear on 0.2 density with MHED routes being 43% shorter. The difference decreases to 29% in a complete network. This is due to the fact that paths by ED are already shorter on higher density networks; therefore, it is harder for MHED to further shorten the paths. Similarly, MHED paths are shorter than ED paths in Scenario 2. The 44% difference at network density 0.1 decreases to 11% in a complete network.

Comparing different scenarios, the average difference between ED and MHED routes is greater in Scenario 1 than Scenario 2. As every other parameter is the same for two scenarios in this experiment, the explanation is in connection profiles. Disconnection times in Scenario 1 are greater than the ones in Scenario 2, on the average. Higher disconnection times make next connection opportunities later, therefore giving MHED a greater time interval to find routes in shorter hops.

Another difference between the scenarios is the path length regardless of the algorithm. Paths in Scenario 2 are shorter than the ones in Scenario 1. The explanation is very similar to the previous one. Due to longer disconnection times in Scenario 1, routing algorithms search for shorter paths in time by increasing hop count. Since disconnection times are already shorter for Scenario 2, there is more possibility to find earlier delivery paths without increasing the number of hops.

Figure 4.3 gives the comparison of maximum path lengths. The difference between MHED and ED maximum paths are higher than the difference between average paths, 55% versus 37% for Scenario 1 and 54% versus 28% for Scenario 2 on the average. Previously, we discussed it is hard to find a shorter path than an already short path. These results show that it is easier to find shorter paths for longer paths.

An interesting observation is the increase in path lengths for ED routing from a density of 0.0 to 0.1 in Scenario 1. It seems that links that are added after the network becomes connected provide earlier delivery paths through more hops for this particular case.

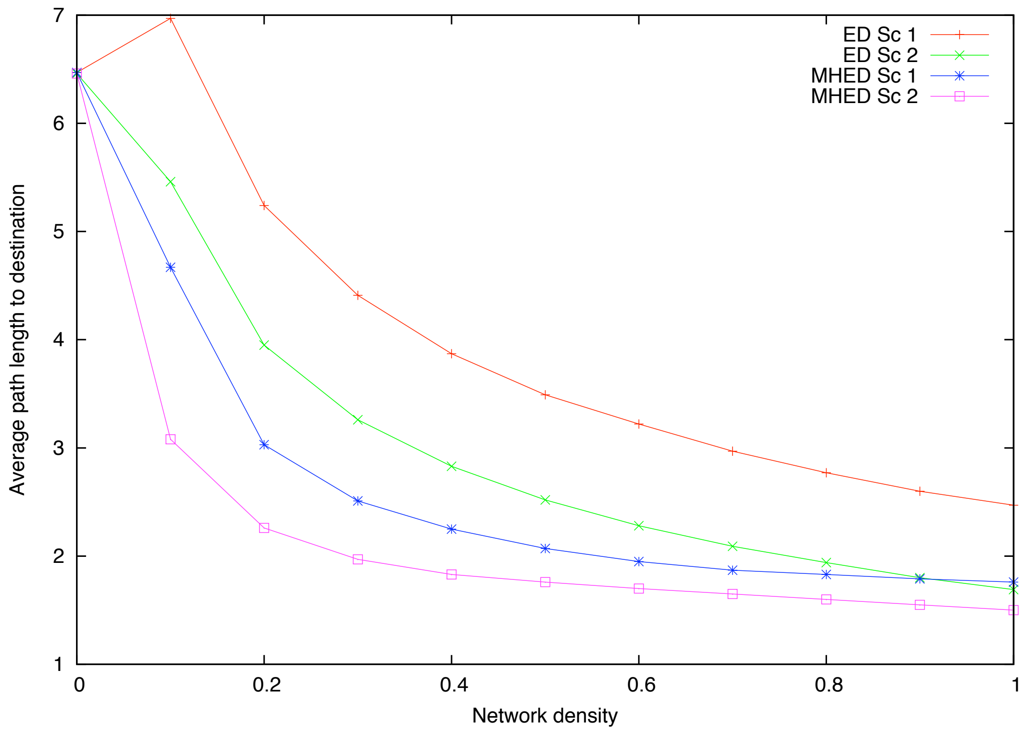


Figure 4.2: Average path lengths to a destination in a 100 node network.

4.2.3 Routing Tree Stability

We define routing tree stability as the tendency of the paths to destinations to remain the same at successive time intervals. We are interested in routing tree stability of the ED and MHED algorithms.

Figure 4.4 shows that MHED routes are more stable than ED routes. We identify two reasons: First, MHED routes are shorter in hop count, therefore, are less likely to change from time to time than a path with more hops. Second, MHED routes prefer paths with later delivery times at the beginning of a path, because it does not make greedy decisions on delivery time. When the first connection of a path is at t_1, it is likely to utilize the connection for $t < t_1$.

Routing tree stability decreases sharply from 20 to 70 nodes, then remains relatively the same. In a 20-node network with density 0.2, each node has about 2 connections on the average. This reduces the number of alternate paths for a destination, therefore increasing the routing tree stability. An average neighbor

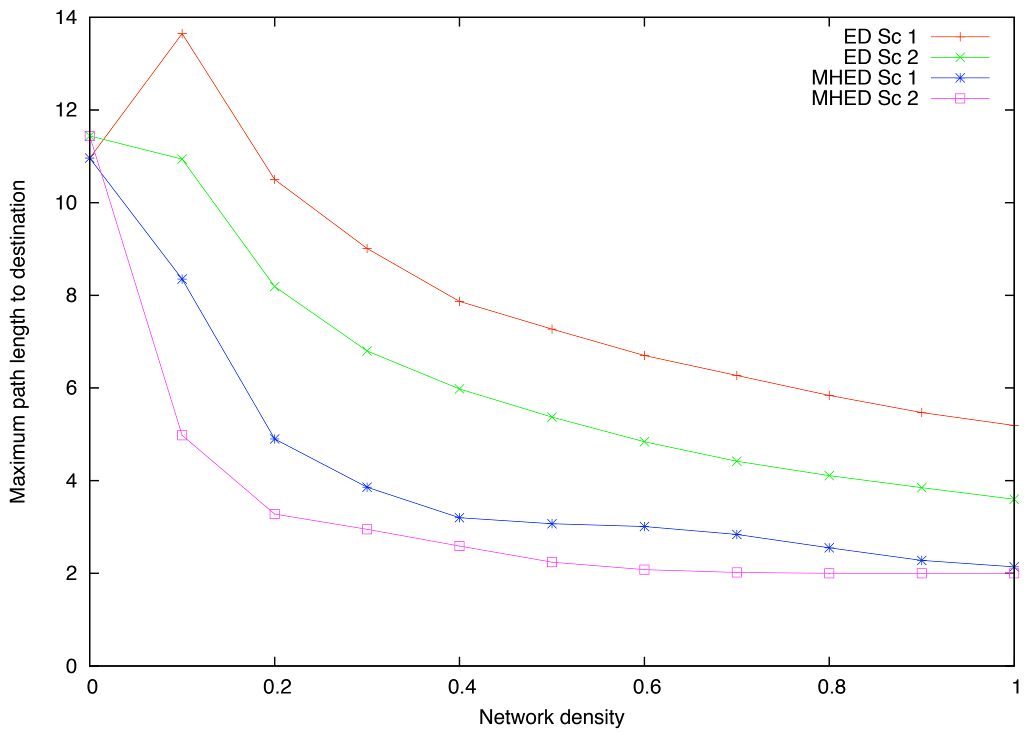


Figure 4.3: Maximum path lengths to a destination in a 100 node network.

count of 7 provides enough alternative paths so that higher number of neighbor counts does not affect routing tree stability. This view is supported by Figure 4.5 showing routing tree stability for a density of 0.5. Stability does not alter much for networks with higher than 30 nodes, which has an average neighbor count of 7.5.

Although routing tree stability is a metric distinguishing ED and MHED, it does not provide any practical benefits; because, route to a destination changes at every time interval on the average. Routing tree stability of a 30-node network with 0.5 density is about 25%, meaning that 25% of all routes remain the same at next time interval. However, since each time a different set of 25% of the routes remain the same, average stability time for a single path is 1 time unit.

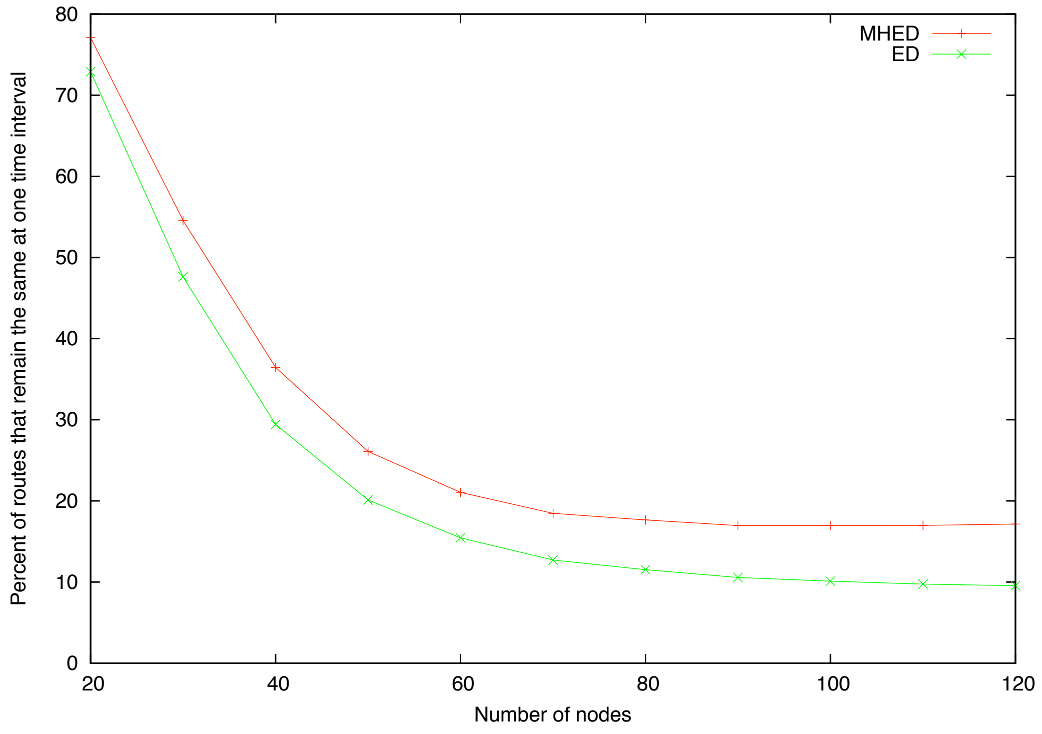


Figure 4.4: Routing tree stability for network density 0.2.

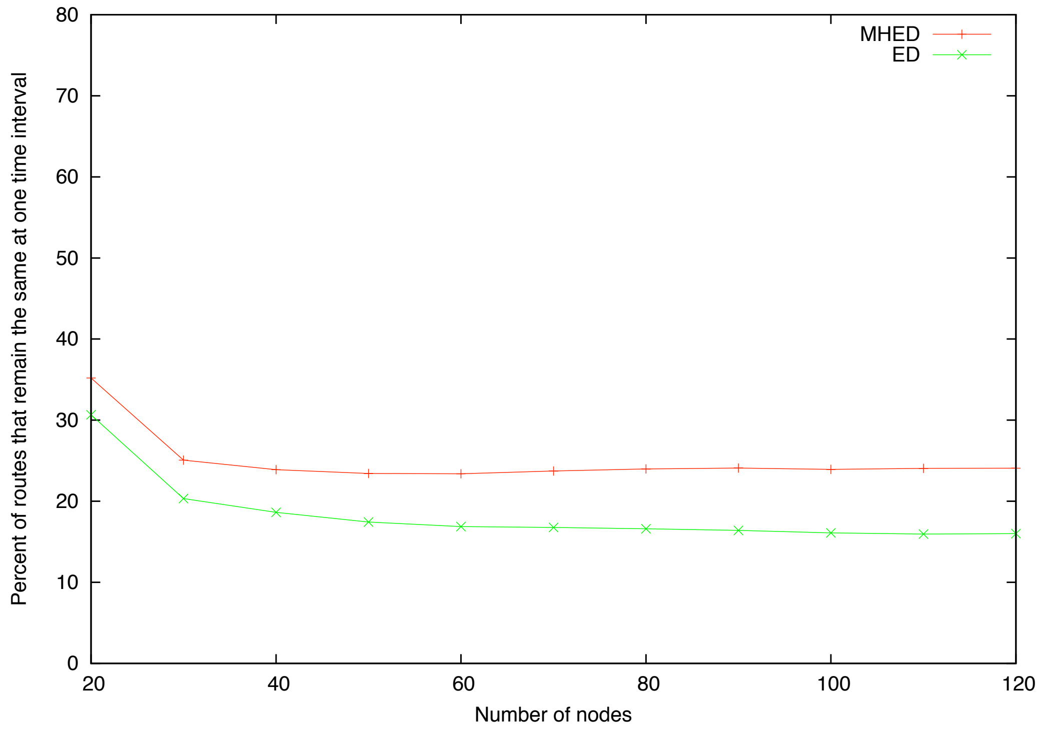


Figure 4.5: Routing tree stability for network density 0.5.

Chapter 5

Conclusion and Future Work

In this thesis, we examined DTN routing problem. We distinguished three DTN connection models of increasing complexity. In the first model, scheduled connections, only the first future contact times between nodes are known. The second model improves upon scheduled connections by introducing periodic connections. In a scheduled periodic connection model, contacts initially take place at a future time, then repeats at certain intervals. Third model, moreover, distinguishes between connection and disconnection times. In this more realistic case, a contact becomes available at a certain time for a duration of T_{ON} after which a disconnection period follows for T_{OFF} time. These connection and disconnection periods alternately occur for a link.

We proposed two routing algorithms, earliest delivery (ED) and min-hop earliest delivery (MHED). These are single-copy routing strategies utilizing contact information defined by the connection model. ED routing is based on Dijkstra's algorithm with some modifications to accommodate the DTN's needs. We provided three versions of ED, one for each of the connection models. Upon observing shortcomings of ED routing, we introduced MHED routing achieving earliest delivery as ED, but also minimizing the hop-count to destinations. To best of our knowledge these two objectives have never been addressed simultaneously in a DTN routing algorithm. MHED has two versions: one based on Dijkstras approach, and one based on BFS. Running time analysis shows that BFS based

MHED has a worst case running time of $O(\frac{E^2}{V})$ and is better than Dijkstra based MHED.

We also presented simulation experiment results comparing both of our algorithms to one of the most popular routing approach in DTNs, epidemic routing. Being a multi-copy routing scheme, epidemic routing makes plenty of transmissions, where ED and MHED achieves the same delivery time with only a few transmissions by using contact information. Other simulation experiment results show statistical properties of ED and MHED. Under two different scenarios with different connection properties, we observed that MHED finds routes that are shorter than ED routes, on the average for both scenarios. We also analyzed maximum path lengths and found that MHED performs better in finding shorter alternatives to longer routes. In another experiment, we examined how a routing tree is susceptible to change in both algorithms. On the average, routing tree of MHED changes less than the routing tree of ED at one time interval, because of the fact that MHED routes are shorter and it prefers routes that have later departure times from the source. On the other hand, we found that routing table stability is not very useful in practice, since individual routes to destinations change almost at every time interval.

We are considering to extend this study by applying the proposed routing algorithms to non-scheduled periodic connections, in which connections and disconnections are not deterministic, but random. We are planning to estimate future contact times by observing past connection patterns. Another future work idea is the application of presented connection models and routing algorithms to different mobility models. In this work, we abstract from the mobility and focus on connections. We are interested in mobility models conforming to our connection models. An alternative strategy is to extend the connection models so that they better abstract the properties of various mobility models.

Bibliography

[1] V. Conan, J. Leguay, and T. Friedman. Fixed point opportunistic routing in delay tolerant networks. *IEEE Journal on Selected Areas in Communications*, 26(5):773 –782, June 2008.

[2] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. The MIT Press, 2nd revised edition, September 2001.

[3] A. Demers, D. Greene, C. Hauser, W. Irish, J. Larson, S. Shenker, H. Sturgis, D. Swinehart, and D. Terry. Epidemic algorithms for replicated database maintenance. In *PODC '87: Proceedings of the Sixth Annual ACM Symposium on Principles of Distributed Computing*, pages 1–12, New York, NY, USA, 1987. ACM.

[4] K. Fall. A delay-tolerant network architecture for challenged internets. In *SIGCOMM '03: Proceedings of the Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, pages 27–34, New York, NY, USA, 2003. ACM.

[5] K. Fall and S. Farrell. Dtn: An architectural retrospective. *IEEE Journal on Selected Areas in Communications*, 26(5):828 –836, June 2008.

[6] S. Jain, K. Fall, and R. Patra. Routing in a delay tolerant network. In *SIGCOMM '04: Proceedings of the Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, pages 145–158, New York, NY, USA, 2004. ACM.

[7] E. P. C. Jones, L. Li, and P. A. S. Ward. Practical routing in delay-tolerant networks. In *WDTN '05: Proceedings of the ACM SIGCOMM Workshop on Delay-tolerant Networking*, pages 237–243, New York, NY, USA, 2005. ACM.

[8] A. Lindgren, A. Doria, and O. Schelén. Probabilistic routing in intermittently connected networks. *SIGMOBILE Mobile Computer Communication Review*, 7(3):19–20, 2003.

[9] C. Liu and J. Wu. Scalable routing in delay tolerant networks. In *MobiHoc '07: Proceedings of the 8th ACM International Symposium on Mobile Ad hoc Networking and Computing*, pages 51–60, New York, NY, USA, 2007. ACM.

[10] C. Liu and J. Wu. An optimal probabilistic forwarding protocol in delay tolerant networks. In *MobiHoc '09: Proceedings of the Tenth ACM International Symposium on Mobile Ad hoc Networking and Computing*, pages 105–114, New York, NY, USA, 2009. ACM.

[11] S. Merugu, M. Ammar, and E. Zegura. Routing in space and time in networks with predictable mobility. Technical report, College of Computing, Georgia Institute of Technology, 2004.

[12] T. Spyropoulos, K. Psounis, and C. S. Raghavendra. Single-copy routing in intermittently connected mobile networks. In *IEEE SECON*, pages 235–244, 2004.

[13] T. Spyropoulos, K. Psounis, and C. S. Raghavendra. Spray and wait: an efficient routing scheme for intermittently connected mobile networks. In *WDTN '05: Proceedings of the ACM SIGCOMM Workshop on Delay-tolerant Networking*, pages 252–259, New York, NY, USA, 2005. ACM.

[14] A. Vahdat and D. Becker. Epidemic routing for partially-connected ad hoc networks. Technical report, Department of Computer Science, Duke University, 2000.