A hybrid replication routing protocol for delay tolerant networks

Richard Joy, Miriam Joy, Serena Mei, Suyog Parajuli
George Mason University
Fairfax, Virginia
[rjoy, mjoy, smei, sparaju3]@gmu.edu

Abstract— Epidemic Routing Protocol (ERP) is an incredibly simple, but extremely resource hungry, method for routing messages within a Delay Tolerant Network (DTN). This research provides a protocol which combines another replication protocol, Probabilistic Routing Protocol using History of Encounters and Transitivity (PRoPHET) with ERP. In this research, we test a specially tailored version of the PRoPHET algorithm within a small virtual environment. In our PRoPHET algorithm, if the probability of delivery falls below a certain threshold, the protocol will switch to standard ERP. Our research is designed to determine whether the use of a hybrid PRoPHET/ERP approach significantly reduces the use of network resources (including associated storage) while delivering the message within acceptable parameters.

Keywords—delay tolerant networks; PRoPHET; epidemic routing protocol; networks;

I. INTRODUCTION

The field of Delay Tolerant Networks (DTN) is not a new one; As early as 1966, computer scientists were engaged in research on delay-tolerant and asynchronous routing exchanges [1]. Obviously, with the significant increase in mobile computing devices and other wireless protocols, the fault tolerance for these kinds of networks must be even higher. In early work, the assumption was made that, while weak, there was always a presence of connected path from source to destination. The reality of the mobile and wireless environment is that in many cases there is not a connected path from source to destination or it is intermittently disrupted. In Epidemic Routing Protocol (ERP), a packet is replicated in every node that comes in contact with the node that carries the packet [2]. This is done rather indiscriminately as is an example of a “greedy” approach; as a result, ERP increases stress on network resources, but is reliable and quick method in low connectivity and low mobility environments where there is no previous knowledge of the network environment. ERP also allows for prioritization of messages. Since a DTN is most often implemented with mobile nodes, high and/or inefficient use of energy resources is not ideal [3].

Probabilistic Routing Protocol using History of Encounters and Transitivity (PRoPHET) is a “utility-based” replication protocol because it is not indiscriminate in its replication [3]. Instead, it uses an adaptive algorithm that attempts to exploit the non-randomness of real-world encounters by maintaining a set of probabilities for successful delivery to known destination in the DTN and replicating messages during opportunistic encounters only if the transmitter that does not have the message appears to have a better chance of delivering it. Lindren, Doria, and Schelén published the PRoPHET algorithm as a response to ERP within low (or intermittent) connectivity environments in 2003 [4].

II. RELATED PROJECTS AND RESEARCH

ERP has a very extensive background of research since its publication in 2000 [2]. A number of papers have addressed variations on ERP solutions and hybrid approaches using those solutions [5, 6]. These papers show differences in performance based on the variations of the ERP protocol and extensively address the metrics and types of analysis which must be performed on DTN routing protocols.

PRoPHET was one of the first protocols to be developed in the early days of DTN research and it is also the only DTN routing protocol that has a well-defined specification of the protocol details in an IRTF Internet Draft [7]. In several of these studies, it has not been found to perform optimally [7]. As a result, studies of these shortfalls have been performed to improve on the overall protocol. For example, PRoPHETv2 has been introduced which modified the transitive property of the algorithm[7]. In a 2011 paper, Zheng et al. added a message arrival notification to the algorithm and improved buffer space efficiency [8]. E-PROPHET introduced contact frequency and contact duration in to the protocol [8].

There has been no work that we have encountered which tests a hybrid approach between these two replicating algorithms.

III. SOLUTIONS AND ANALYSIS

In this research, we present and test a hybrid protocol. By preceding the traditional ERP with PRoPHET, we propose that a hybrid PRoPHET/ERP approach could significantly reduce the use of network resources (including associated storage) while delivering the message within acceptable parameters. Also, since it is an adaptive algorithm, the success rate should increase over time. We propose that we will apply a version of specially tailored PRoPHET algorithm within a small virtual environment. In our PRoPHET algorithm, if the probability of delivery falls below a certain threshold, the protocol will switch to standard ERP. We will test a variety of thresholds.
over a series of environments to determine a potentially ideal combination. We used Java to create our test environment.

A. ProPHET equation

The essential element in this research is the adaptive ProPHET algorithm [4]. The algorithm has three distinct components. First, the algorithm updates whenever a node is encountered. As a result, nodes that are encountered frequently have a high delivery predictability. As described in Appendix A, the ProPHET statistics are shared between each of the communicating nodes. This calculation is shown in Eq. 1, where $P_{\text{init}} \in [0;1]$ is an initialization constant.

$$P_{(a,b)} = P_{(a,b)\text{old}} + (1 - P_{(a,b)\text{old}}) \times P_{\text{init}} \quad (1)$$

If a pair of nodes does not encounter each other frequently, they are less likely to be good forwarders of messages to each other, thus the delivery predictability values must age, being reduced in the process. The aging equation is shown in Eq. 2, where $\gamma \in [0;1)$ is the aging constant, and $k$ is the number of time units that have elapsed since the last time the metric was aged. The time unit used can differ, and should be defined based on the application and the expected delays in the targeted network.

$$P_{(a,b)} = P_{(a,b)\text{old}} \times \gamma^k \quad (2)$$

The final component to the ProPHET algorithm is the transitive property shown in Eq. 3 which allows the algorithm to improve its ability for connection beyond the original set of connected nodes. $\beta \in [0;1)$ is a scaling constant that decides how large impact the transitivity should have on the delivery predictability.

$$P_{(a,c)} = P_{(a,c)\text{old}} + (1 - P_{(a,c)\text{old}}) \times P_{(a,b)} \times P_{(b,c)} \times \beta \quad (3)$$

As shown in Table 1, we utilized the same parameter settings as the original ProPHET research for our network simulation.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>$P_{\text{init}}$</td>
<td>0.75</td>
</tr>
<tr>
<td>$\beta$</td>
<td>0.25</td>
</tr>
<tr>
<td>$\gamma$</td>
<td>0.98</td>
</tr>
</tbody>
</table>

Table 1. ProPHET Parameter Settings

B. Hybrid Protocol Structure

The protocol we designed is a standard handshake protocol. Into this protocol, we introduced an exchange of probabilities. Based on that exchange, the protocol will either proceed with the dissemination of the message using ProPHET or “switch” to an ERP approach. Appendix A contains the entire protocol; the essential elements are presented in Fig. 1.

C. Virtual Network

For the purposes of this research, a network was created as described in Table 2. Our network contained twenty distinct
nodes. As documented in Appendix B, each node was provided with at least one message to communicate to a separate node. The node messages were assigned randomly.

Each node was assigned four potential partners (chosen haphazardly) for connection with a 25% probability of connecting to any given partner assigned as a seed value. Each network node was intentionally connected to its predecessor and successor nodes to ensure that the network was connected as two of its connections. Appendix E includes the source code for the virtual network.

Table 2: Java Classes used to create network

<table>
<thead>
<tr>
<th>Java Class Name</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>DTNConnector.java</td>
<td>Node serving in client role</td>
</tr>
<tr>
<td>DTNListener.java</td>
<td>Node serving in server role</td>
</tr>
<tr>
<td>DTNNode.java</td>
<td>Main program class; launches threads and provides any interface elements</td>
</tr>
<tr>
<td>ConnectionList.java</td>
<td>Reads configuration information and produce a node's next connection partner based on the provided connection probabilities</td>
</tr>
<tr>
<td>MessageStatistics.java</td>
<td>Store messages and PROPHET statistics</td>
</tr>
<tr>
<td>DTNMessageManager.java</td>
<td>Provide reporting interface for recording results</td>
</tr>
<tr>
<td>DTNIDBCConnector.java</td>
<td>Provide method for making connections to the initialization and results database</td>
</tr>
</tbody>
</table>

D. Results

We ran a series of tests incrementing the threshold for switching the protocol to ERP. These test allowed us to not only establish a baseline at both a full ERP and full PROPHET level, but to also determine if there was an ideal mixture. An environment with a completely PROPHET propagation protocol was assigned a threshold of “0”. An entirely ERP-propagated environment was provided with a threshold of “1”.

Each increment was run for 40 minutes with a time-to-live (TTL) for each message of 128 connections. There were 30 messages at the beginning of each run.

Over the course of our test, we delivered one message successfully (Appendix B). This message was delivered using ERP as a protocol.

We speculate that there are two major reasons why the PROPHET protocol was not successful in delivering a message despite the fact that the probability of delivery success was in PROPHET’s favor until the threshold was switched to 100% ERP (Fig. 2).

1. The time increment of 40 minutes was not long enough for the network to adapt into a more effective routing mechanism;
2. We did not let the PROPHET algorithm prime itself with network connections before starting the message simulation. In the original PROPHET protocol, the long history component is considered critical [4]. In our view, a network which must be functional from initiation is more realistic.

Despite the fact that the hybrid approach was not successful in delivering a message, it was abundantly clear that the pure-PROPHET algorithm and the hybrid combinations were much more efficient from a resource usage perspective. As shown in Fig. 3 and 4, as the threshold was increased in favor of ERP propagation, the number of messages being replicated (and buffered) increased significantly, even though the actual selection of the ERP propagation method was relatively low by the nodes themselves. Accordingly, the number of expired messages residing in the network (time-to-live passed) also increased.

Interestingly, as shown in Appendices C and D, each node had a particular communication profile. The one node that successfully delivered its message (Node 12358) and its recipient node (Node 12346) and more active and more consistent connection patterns, respectively.

![Figure 2](image_url)

**Figure 2** Breakdown of each propagation type by threshold.

![Figure 3](image_url)

**Figure 3** Avg and Max message buffer size across all nodes
IV. FUTURE WORK

A. Future Work

In this paper, we specifically addressed two replication protocols and their potential hybridization to produce a more successful result. These are only two of many DTN-related protocols and only represent one strategy (replication). As it relates to our specific research the following topics would be interesting for research:

- Routing Under Uncertainty – How should the probabilities of the partner and target nodes be calculated? How do environmental and technical considerations affect this calculation? Is information on location of target node reliable or being distorted by environmental conditions? How do you adjust for this?
- Resource Allocation – What is the most appropriate method to elevate messages for priority handling? How can buffers be managed more efficiently and how can system storage be managed?
- Minimizing the amount of energy consumed in transmission (particularly applicable to mobile devices serving as carriers for others)
- Security and tracking of messages throughout the transmission chain

REFERENCES

Appendix A
Hybrid Routing Protocol

Terms:
Client - The partner that initiates the connection (i.e. the ‘DTNConnector’ thread)
Server - The partner that accepts the connection (i.e. the ‘DTNListener’ thread)

Keywords:
PROPHET
ENDPROPHET
IDS
ENDIDS
REQUEST
ENDREQUEST
MESSAGE
PROVIDE
ENDPROVIDE
NOMESSAGE
CLOSE

Arguments:
NODE NodeID

Example Session:
Client
NODE <NodeID> -->
PROPHET
<Destination> <Probability> -->
<Destination> <Probability> -->
<Destination> <Probability> -->
ENDPROPHET

Server
<- NODE <NodeID>
<- PROPHET

Send only as the first message from a node. NodeID is the unique ID of the sending node.

PROPHET
Signals the start of Prophet information. Each destination and its associated prophet probability (from the sender’s current perspective) is sent on a single line, with the destination id and the the associated probability seperated by a space.

ENDPROPHET
Signals the end of Prophet information for the sender.

IDS
Signals the beginning of the list of message IDs that the sender currently has. The message IDs should be sent one per line.

ENDIDS
Signals the end of the sender's message ID list.

REQUEST
Signals the beginning of the list of messages that the node is requesting from its partner. The server waits for this information from the client before attempting to send its own version.

ENDREQUEST
Signals the end of the list of messages that the node is requesting from its partner.

MESSAGE messageID
Indicates that the sender would like a copy of the message identified by <messageID>.

PROVIDE messageID
Indicates the beginning of the message identified by <messageID>

ENDPROVIDE messageID
Indicates the end of the message identified by <messageID>

NOMESSAGE
Indicates that the node does not have a copy of the requested message.

CLOSE
Indicates that the connection should be terminated.
<-- <Destination> <Probability>
<-- <Destination> <Probability>
<-- <Destination> <Probability>
<-- â€¦
<-- ENDPROPHET

IDS
<-- -->
<messageID>
<-- -->
<messageID>
<-- -->
<messageID>
<-- -->
<messageID>
<-- -->
<messageID>
<-- -->
<messageID>
<-- -->
<messageID>
<-- -->
<messageID>
<-- -->
<messageID>
<-- -->
<messageID>
<-- -->
<messageID>
<-- -->
<messageID>
<-- -->
<messageID>
<-- -->
<messageID>
<-- -->
<messageID>
<-- -->
<messageID>
<-- -->
<messageID>
<-- -->
<messageID>
<-- -->
<messageID>
<-- -->
<messageID>
<-- -->
<messageID>
<-- -->
<messageID>
<-- -->
<messageID>
<-- -->
<messageID>
<-- -->
<messageID>
<-- -->
<messageID>
<-- -->
<messageID>
<-- -->
<messageID>
<-- -->
<messageID>
<-- -->
<messageID>
<-- -->
<messageID>
<-- -->
<messageID>
<-- -->
<messageID>
<-- -->
<messageID>
<-- -->
<messageID>
<-- -->
<messageID>
<-- -->
<messageID>
<-- -->
<messageID>
<-- -->
<messageID>
<-- -->
<messageID>
<-- -->
<messageID>
<-- -->
<messageID>
<-- -->
<messageID>
<-- -->
<messageID>
<-- -->
<messageID>
<-- -->
<messageID>
<-- -->
<messageID>
<-- -->
<messageID>
<-- -->
<messageID>
<-- -->
<messageID>
<-- -->
<messageID>
<-- -->
<messageID>
<-- -->
<messageID>
<-- -->
<messageID>
<-- -->
<messageID>
<-- -->
<messageID>
<-- -->
<messageID>
<-- -->
<messageID>
<-- -->
<messageID>
<-- -->
<messageID>
<-- -->
<messageID>
<-- -->
<messageID>
<-- -->
<messageID>
<-- -->
<messageID>
<-- -->
<messageID>
<-- -->
<messageID>
<-- -->
<messageID>
<-- -->
<messageID>
<-- -->
<messageID>
<-- -->
<messageID>
<-- -->
<messageID>
<-- -->
<messageID>
<-- -->
<messageID>
<-- -->
<messageID>
<-- -->
<messageID>
<-- -->
<messageID>
<-- -->
<messageID>
<-- -->
<messageID>
<-- -->
<messageID>
<-- -->
<messageID>
<-- -->
<messageID>
<-- -->
<messageID>
<-- -->
<messageID>
<-- -->
<messageID>
<-- -->
<messageID>
<-- -->
<messageID>
<-- -->
<messageID>
<-- -->
<messageID>
<-- -->
<messageID>
<-- -->
<messageID>
<-- -->
<messageID>
<-- -->
<messageID>
<-- -->
<messageID>
<-- -->
<messageID>
<-- -->
<messageID>
<-- -->
<messageID>
<-- -->
<messageID>
<-- -->
<messageID>
<-- -->
<messageID>
<-- -->
<messageID>
<-- -->
<messageID>
<-- -->
<messageID>
<-- -->
<messageID>
<-- -->
<messageID>
<-- -->
<messageID>
<-- -->
<messageID>
<-- -->
<messageID>
<-- -->
<messageID>
<-- -->
<messageID>
<-- -->
<messageID>
<-- -->
<messageID>
<-- -->
<messageID>
<-- -->
<messageID>
<-- -->
<messageID>
<-- -->
<messageID>
<-- -->
<messageID>
<-- -->
<messageID>
<-- -->
<messageID>
<-- -->
<messageID>
<-- -->
<messageID>
<-- -->
<messageID>
<-- -->
<messageID>
<-- -->
<messageID>
<-- -->
<messageID>
<-- -->
<messageID>
<-- -->
<messageID>
<-- -->
<messageID>
<-- -->
<messageID>
<-- -->
<messageID>
<-- -->
<messageID>
<-- -->
<messageID>
<-- -->
<messageID>
<-- -->
<messageID>
<-- -->
<messageID>
<-- -->
<messageID>
<-- -->
<messageID>
<-- -->
<messageID>
<-- -->
<messageID>
<-- -->
<messageID>
<-- -->
<messageID>
<-- -->
<messageID>
<-- -->
<messageID>
<-- -->
<messageID>
<-- -->
<messageID>
<-- -->
<messageID>
<-- -->
<messageID>
<-- -->
<messageID>
<-- -->
<messageID>
<-- -->
<messageID>
<-- -->
<messageID>
<-- -->
<messageID>
<-- -->
<messageID>
<-- -->
<messageID>
<-- -->
<messageID>
<-- -->
<messageID>
<-- -->
<messageID>
<-- -->
<messageID>
<-- -->
<messageID>
<-- -->
<messageID>
<-- -->
<messageID>
<-- -->
<messageID>
<-- -->
<messageID>
<-- -->
<messageID>
<-- -->
<messageID>
<-- -->
<messageID>
<-- -->
<messageID>
<-- -->
<messageID>
<-- -->
<messageID>
<-- -->
<messageID>
<-- -->
<messageID>
<-- -->
<messageID>
<-- -->
<messageID>
<-- -->
<messageID>
<-- -->
<messageID>
<-- -->
<messageID>
<-- -->
<messageID>
<-- -->
<messageID>
<-- -->
<messageID>
<-- -->
<messageID>
<-- -->
<messageID>
<-- -->
<messageID>
<-- -->
<messageID>
<-- -->
<messageID>
<-- -->
<messageID>
<-- -->
<messageID>
<-- -->
<messageID>
<-- -->
<messageID>
<-- -->
<messageID>
<-- -->
<messageID>
<-- -->
<messageID>
<-- -->
<messageID>
<-- -->
<messageID>
<-- -->
<messageID>
<-- -->
<messageID>
<-- -->
<messageID>
<-- -->
<messageID>
<-- -->
<messageID>
<-- -->
<messageID>
<-- -->
<messageID>
<-- -->
<messageID>
<-- -->
<messageID>
<-- -->
<messageID>
<-- -->
<messageID>
<-- -->
<messageID>
<-- -->
<messageID>
<-- -->
<messageID>
<-- -->
<messageID>
<-- -->
<messageID>
<-- -->
<messageID>
<-- -->
<messageID>
<-- -->
<messageID>
<-- -->
<messageID>
<-- -->
<messageID>
<-- -->
<messageID>
<-- -->
<messageID>
<-- -->
<messageID>
<-- -->
<messageID>
<-- -->
<messageID>
<-- -->
<messageID>
<-- -->
<messageID>
<-- -->
<messageID>
<-- -->
<messageID>
<-- -->
<messageID>
<-- -->
<messageID>
<-- -->
<messageID>
<-- -->
<messageID>
<-- -->
<messageID>
<-- -->
<messageID>
<-- -->
<messageID>
<-- -->
<messageID>
<-- -->
<messageID>
<-- -->
<messageID>
<-- -->
<messageID>
<-- -->
<messageID>
<-- -->
<messageID>
<-- -->
<messageID>
<-- -->
<messageID>
<-- -->
<messageID>
<-- -->
<messageID>
<-- -->
<messageID>
<-- -->
<messageID>
<-- -->
<messageID>
<-- -->
<messageID>
<-- -->
<messageID>
<-- -->
<messageID>
<-- -->
<messageID>
<-- -->
<messageID>
<-- -->
<messageID>
<-- -->
<messageID>
<-- -->
<messageID>
<-- -->
<messageID>
<-- -->
<messageID>
<-- -->
<messageID>
<-- -->
<messageID>
<-- -->
<messageID>
<-- -->
<messageID>
<-- -->
<messageID>
<-- -->
<messageID>
<-- -->
<messageID>
<-- -->
<messageID>
<-- -->
<messageID>
<-- -->
<messageID>
<-- -->
<messageID>
<-- -->
<messageID>
<-- -->
<messageID>
<-- -->
<messageID>
<-- -->
<messageID>
<-- -->
<messageID>
<-- -->
<messageID>
<-- -->
<messageID>
<-- -->
<messageID>
<-- -->
<messageID>
<-- -->
<messageID>
<-- -->
<messageID>
<-- -->
<messageID>
<-- -->
<messageID>
<-- -->
<messageID>
<-- -->
<messageID>
<-- -->
<messageID>
<-- -->
<messageID>
<-- -->
<messageID>
<-- -->
<messageID>
<-- -->
<messageID>
<-- -->
<messageID>
<-- -->
<messageID>
<-- -->
<messageID>
<-- -->
<messageID>
<-- -->
<messageID>
<-- -->
<messageID>
<-- -->
<messageID>
<-- -->
<messageID>
## Appendix B

### Schedule Messages by Sender Node and Delivery Success

<table>
<thead>
<tr>
<th>Sender ID</th>
<th>Receiver ID</th>
<th>No</th>
<th>Yes</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>127.0.0.1:12340</td>
<td>127.0.0.1:12343</td>
<td>1</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>127.0.0.1:12347</td>
<td>1</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>127.0.0.1:12340 Total</td>
<td></td>
<td>2</td>
<td></td>
<td>2</td>
</tr>
<tr>
<td>127.0.0.1:12341</td>
<td>127.0.0.1:12346</td>
<td>1</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>127.0.0.1:12349</td>
<td>1</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>127.0.0.1:12341 Total</td>
<td></td>
<td>2</td>
<td></td>
<td>2</td>
</tr>
<tr>
<td>127.0.0.1:12342</td>
<td>127.0.0.1:12348</td>
<td>2</td>
<td></td>
<td>2</td>
</tr>
<tr>
<td>127.0.0.1:12342 Total</td>
<td></td>
<td>2</td>
<td></td>
<td>2</td>
</tr>
<tr>
<td>127.0.0.1:12343</td>
<td>127.0.0.1:12342</td>
<td>1</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>127.0.0.1:12355</td>
<td>1</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>127.0.0.1:12343 Total</td>
<td></td>
<td>2</td>
<td></td>
<td>2</td>
</tr>
<tr>
<td>127.0.0.1:12344</td>
<td>127.0.0.1:12340</td>
<td>1</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>127.0.0.1:12342</td>
<td>1</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>127.0.0.1:12344 Total</td>
<td></td>
<td>2</td>
<td></td>
<td>2</td>
</tr>
<tr>
<td>127.0.0.1:12345</td>
<td>127.0.0.1:12340</td>
<td>1</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>127.0.0.1:12344</td>
<td>1</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>127.0.0.1:12345 Total</td>
<td></td>
<td>2</td>
<td></td>
<td>2</td>
</tr>
<tr>
<td>127.0.0.1:12346</td>
<td>127.0.0.1:12341</td>
<td>1</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>127.0.0.1:12356</td>
<td>1</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>127.0.0.1:12346 Total</td>
<td></td>
<td>2</td>
<td></td>
<td>2</td>
</tr>
<tr>
<td>127.0.0.1:12347</td>
<td>127.0.0.1:12349</td>
<td>1</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>127.0.0.1:12351</td>
<td>1</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>127.0.0.1:12347 Total</td>
<td></td>
<td>2</td>
<td></td>
<td>2</td>
</tr>
<tr>
<td>127.0.0.1:12348</td>
<td>127.0.0.1:12343</td>
<td>1</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>127.0.0.1:12350</td>
<td>1</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>127.0.0.1:12348 Total</td>
<td></td>
<td>2</td>
<td></td>
<td>2</td>
</tr>
<tr>
<td>127.0.0.1:12349</td>
<td>127.0.0.1:12345</td>
<td>1</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>127.0.0.1:12353</td>
<td>1</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>127.0.0.1:12349 Total</td>
<td></td>
<td>2</td>
<td></td>
<td>2</td>
</tr>
<tr>
<td>127.0.0.1:12350</td>
<td>127.0.0.1:12357</td>
<td>1</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>127.0.0.1:12350 Total</td>
<td></td>
<td>1</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>127.0.0.1:12351</td>
<td>127.0.0.1:12344</td>
<td>1</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>127.0.0.1:12351 Total</td>
<td></td>
<td>1</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>127.0.0.1:12352</td>
<td>127.0.0.1:12359</td>
<td>1</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>127.0.0.1:12352 Total</td>
<td></td>
<td>1</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>127.0.0.1:12353</td>
<td>127.0.0.1:12358</td>
<td>1</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>127.0.0.1:12353 Total</td>
<td></td>
<td>1</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>127.0.0.1:12354</td>
<td>127.0.0.1:12347</td>
<td>1</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>127.0.0.1:12354 Total</td>
<td></td>
<td>1</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>127.0.0.1:12355</td>
<td>127.0.0.1:12353</td>
<td>1</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>127.0.0.1:12355 Total</td>
<td></td>
<td>1</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>127.0.0.1:12356</td>
<td>127.0.0.1:12341</td>
<td>1</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>127.0.0.1:12356 Total</td>
<td></td>
<td>1</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>127.0.0.1:12357</td>
<td>127.0.0.1:12354</td>
<td>1</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>127.0.0.1:12357 Total</td>
<td></td>
<td>1</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>127.0.0.1:12358</td>
<td>127.0.0.1:12346</td>
<td>1</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>127.0.0.1:12358 Total</td>
<td></td>
<td>1</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>127.0.0.1:12359</td>
<td>127.0.0.1:12345</td>
<td>1</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>127.0.0.1:12359 Total</td>
<td></td>
<td>1</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>Grand Total</td>
<td></td>
<td>29</td>
<td>1</td>
<td>30</td>
</tr>
</tbody>
</table>
### APPENDIX C

*Average and Maximum Message Buffer Size per Node*

<table>
<thead>
<tr>
<th>Node</th>
<th>MAX (MESSAGE_COUNT)</th>
<th>AVG (MESSAGE_COUNT)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0  0.1  0.2  0.3  0.4  0.5  0.6  0.7  0.8  0.9  1</td>
<td>0  0.1  0.2  0.3  0.4  0.5  0.6  0.7  0.8  0.9  1</td>
</tr>
<tr>
<td>127.0.0.1:12340</td>
<td>2  2  2  2  2  2  2  2  2  2  2</td>
<td>2.00  2.00  2.00  2.00  2.00  2.00  2.00  2.00  2.00  2.00  2.00  2.00  2.00</td>
</tr>
<tr>
<td>127.0.0.1:12341</td>
<td>2  2  6  7  6  6  6  6  6  6</td>
<td>0.79  0.81  0.66  2.04  1.86  1.70  1.61  1.81  1.85  2.06  2.35</td>
</tr>
<tr>
<td>127.0.0.1:12342</td>
<td>2  2  2  2  3  6  7  6  6  6</td>
<td>0.86  0.75  0.71  0.70  0.80  1.51  1.91  1.76  1.72  1.89  2.41</td>
</tr>
<tr>
<td>127.0.0.1:12343</td>
<td>2  2  2  2  2  2  2  2  2  2</td>
<td>2.00  2.00  2.00  2.00  2.99  2.00  2.00  2.00  2.00  2.00  2.00</td>
</tr>
<tr>
<td>127.0.0.1:12344</td>
<td>2  2  2  2  2  2  2  2  2  2</td>
<td>2.00  2.00  2.00  2.00  2.00  2.99  2.00  2.99  2.99  2.99  2.00</td>
</tr>
<tr>
<td>127.0.0.1:12345</td>
<td>2  2  2  4  7  6  6  6  6  6</td>
<td>0.78  0.70  0.69  1.67  1.94  1.70  1.59  1.62  1.82  1.99  2.28</td>
</tr>
<tr>
<td>127.0.0.1:12346</td>
<td>2  2  2  3  3  3  3  3  3  3</td>
<td>0.18  0.17  0.17  0.24  0.21  0.21  0.21  0.21  0.21  0.22  0.27</td>
</tr>
<tr>
<td>127.0.0.1:12347</td>
<td>2  2  2  3  3  3  3  3  3  3</td>
<td>2.00  2.00  2.00  2.99  2.00  3.00  3.00  3.00  3.00  2.99  2.99</td>
</tr>
<tr>
<td>127.0.0.1:12348</td>
<td>2  2  2  4  7  6  6  6  6  6</td>
<td>0.45  0.39  0.31  0.70  0.98  1.66  1.63  1.36  1.78  2.02  2.23</td>
</tr>
<tr>
<td>127.0.0.1:12349</td>
<td>2  2  2  3  5  6  7  6  6  6</td>
<td>0.87  0.70  0.75  1.05  1.41  1.46  1.96  1.68  1.82  1.73  2.27</td>
</tr>
<tr>
<td>127.0.0.1:12350</td>
<td>1  1  1  1  3  3  3  3  3  3</td>
<td>0.09  0.09  0.08  0.09  0.21  0.22  0.21  0.24  0.21  0.33</td>
</tr>
<tr>
<td>127.0.0.1:12351</td>
<td>1  1  1  1  3  6  7  6  6  6</td>
<td>0.07  0.06  0.06  0.05  0.14  0.53  1.53  0.72  1.56  1.64  2.15</td>
</tr>
<tr>
<td>127.0.0.1:12352</td>
<td>1  1  1  1  2  2  2  2  2  2</td>
<td>0.09  0.08  0.09  0.08  0.08  0.14  0.14  0.14  0.14  0.15  0.18</td>
</tr>
<tr>
<td>127.0.0.1:12353</td>
<td>1  1  1  1  2  2  2  3  3  3</td>
<td>0.09  0.09  0.09  0.08  0.14  0.14  0.14  0.21  0.21  0.22  0.18</td>
</tr>
<tr>
<td>127.0.0.1:12354</td>
<td>1  1  1  2  4  6  7  6  6  6</td>
<td>0.09  0.09  0.10  0.16  0.31  0.43  1.61  0.86  1.64  1.68  2.09</td>
</tr>
<tr>
<td>127.0.0.1:12355</td>
<td>1  1  1  2  2  2  2  2  2  2</td>
<td>0.09  0.09  0.10  0.08  0.15  0.15  0.14  0.15  0.15  0.16  0.19</td>
</tr>
<tr>
<td>127.0.0.1:12356</td>
<td>1  1  1  1  3  3  3  3  3  3</td>
<td>1.00  1.00  1.00  1.00  3.00  3.00  3.00  3.00  3.00  3.00  3.00</td>
</tr>
<tr>
<td>127.0.0.1:12357</td>
<td>2  1  2  1  2  2  2  3  3  3</td>
<td>0.18  0.09  0.19  0.07  0.14  0.13  0.14  0.26  0.22  0.20  0.19</td>
</tr>
<tr>
<td>127.0.0.1:12358</td>
<td>1  1  1  4  6  2  6  2  2  2</td>
<td>1.00  1.00  1.00  4.00  6.00  2.00  6.00  2.00  2.00  2.00  5.00</td>
</tr>
<tr>
<td>127.0.0.1:12359</td>
<td>1  1  1  2  2  3  6  3  3  3</td>
<td>1.00  1.00  1.00  2.00  2.00  3.00  5.99  3.00  3.00  3.00  5.00</td>
</tr>
</tbody>
</table>

**Grand Total** | **31** | **30** | **31** | **46** | **70** | **74** | **84** | **76** | **76** | **79** | **15.64** | **15.11** | **15.02** | **23.00** | **28.43** | **27.94** | **36.80** | **28.98** | **31.34** | **32.14** | **39.10**
Appendix D

Average and Maximum Message Buffer Size per Node (Graphical Representation)

12340

12341

12342

12343

12344

12345
Appendix D (continued)
Average and Maximum Message Buffer Size per Node (Graphical Representation)
Appendix D (continued)

Average and Maximum Message Buffer Size per Node (Graphical Representation)

12352

12353

12354

12355

12356

12357
Appendix D (continued)

Average and Maximum Message Buffer Size per Node (Graphical Representation)
Appendix E

Delay Tolerant Network Source Code

//BEGIN CLASS ConnectionList
import java.sql.ResultSet;
import java.util.Scanner;

public class ConnectionList {
    private int connectionfactor;
    private String[] hosts;
    private int[] ports;
    private long current;

    public ConnectionList(){
        hosts = new String[100];
        ports = new int[100];
        connectionfactor = 0;
        current = 0;
    }

    public ConnectionList(ResultSet rs, int connectionfactor){
        this.connectionfactor = connectionfactor;
        this.hosts = new String[100];
        this.ports = new int[100];
        int count = 0;
        String host;
        int port;
        int probability;
        String NodeID;
        Scanner parser;
        current = 0;
        //System.out.println("rows");
        try{
            while (rs.next()) {
                //System.out.println("Processing row");
                NodeID = rs.getString("NODE2_NAME");
                parser = new Scanner(NodeID);
                parser.useDelimiter(":");
                host = parser.next();
                port = parser.nextInt();
                probability = rs.getInt("CONN_PROBABILITY");
                for (int i = 0; i<100; i++) {
                    if (i<(count+probability)){
                        hosts[i] = host;
                        ports[i] = port;
                    }
                }
                //System.out.println(host + " - " + port);
                count = count + probability;
                //System.out.println(rs.getString("NO DE2_NAME") + " " +
                rs.getInt("CONN_PROBABILITY");
                }
            } catch (Exception e) {
                System.out.println("ResultSet Error");
                System.out.println(e.getMessage());
            }
        }
    }
public ConnectionList(ResultSet rs) {
    connectionfactor = 100;
    this.hosts = new String[100];
    this.ports = new int[100];
    int count = 0;
    String host;
    int port;
    int probability;
    String NodeID;
    Scanner parser;
    current = 0;
    //System.out.println("rows");
    try{
        while (rs.next()){
            //System.out.println("Processing row");
            NodeID = rs.getString("NODE2_NAME");
            parser = new Scanner(NodeID);
            parser.useDelimiter(".:");
            host = parser.next();
            port = parser.nextInt();
            probability = rs.getInt("CONN_PROBABILITY");
            for (int i = 0; i<100; i++){
                if (i<(count+probability)){
                    hosts[i] = host;
                    ports[i] = port;
                }
            }
            //System.out.println(host + " - " + port);
            count = count + probability;
            //System.out.println(rs.getString("NODE2_NAME") + " " + rs.getInt("CONN_PROBABILITY"));
        }
    } catch (Exception e){
        System.out.println("ResultSet Error");
        System.out.println(e.getMessage());
    }
}

public ConnectionList(String host, int port, int connectionfactor){
    this.connectionfactor = connectionfactor;
    this.hosts = new String[100];
    this.ports = new int[100];
    current = 0;
    for (int i = 0; i <100; i++){
        hosts[i] = host;
        ports[i] = port;
    }
}

public ConnectionList(String host, int port){
    connectionfactor = 100;
    this.hosts = new String[100];
    this.ports = new int[100];
    current = 0;
    for (int i = 0; i <100; i++){
        hosts[i] = host;
        ports[i] = port;
    }
}
public boolean chooseNextConnection() {
    current = Math.round(Math.random() * 99);
    return ((Math.round(Math.random() * 99)) < connectionfactor);
}

public String getHostName() {
    return hosts[(int)current];
}

public int getPort() {
    return ports[(int)current];
}

// END CLASS ConnectionList

// BEGIN CLASS DTNConnector

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.PrintWriter;
import java.net.Socket;
import java.util.ArrayList;
import java.util.Scanner;

// This class should serve as the "client" portion of a connection
public class DTNConnector implements Runnable {

    private MessageStatistics messages;
    private ConnectionList connections;
    private Socket clientSocket;
    boolean connecting;
    double threshold;
    ArrayList<String> partnerIDs;
    ArrayList<String> prophetIDs;
    DTNMessageManager messageManager;
    String partnerNodeID;
    String NodeID;

    private void Report(String message) {
        System.out.println(NodeID + "[DTNConnector] - " + message);
    }

    public DTNConnector(String NodeID, ConnectionList connections, MessageStatistics messages, double threshold) {
        messageManager = new DTNMessageManager(messages, threshold);
        this.threshold = threshold;
        this.messages = messages;
        this.connections = connections;
        clientSocket = null;
        connecting = true;
        this.NodeID = NodeID;
        partnerIDs = null;
        prophetIDs = null;
    }

}
public void close(){
    connecting = false;
}

public void run(){
    Report("Started.");
    BufferedReader in = null;
    PrintWriter out = null;

    while (connecting){
        try{
            Thread.sleep(Math.round(Math.random()*50));
        } catch (Exception e){

        }

    messages.ageAllProphet();
    messageManager.reportDeliveredMessages(NodeID);
    messageManager.reportUndeliveredMessages(NodeID);
    messageManager.reportMessageBuffer(NodeID);

    if (connections.chooseNextConnection()){  
        try{
            clientSocket = new Socket(connections.getHostName(),connections.getPort());
        }catch (IOException e){
            clientSocket = null;
        }
    } else {
        clientSocket = null;
    }

    if (clientSocket != null){
        try {
            out = new PrintWriter(clientSocket.getOutputStream(), true);
        } catch (IOException e){
            out = null;
        }

        try {
            in = new BufferedReader(new InputStreamReader(clientSocket.getInputStream()));
        } catch (IOException e){
            in = null;
        }

        if ((out != null) && (in != null)){
            ClientServe(in, out);
        }
    } else {
        if (out != null){
            try{
                out.close();
            } catch (Exception e){

            }
        }
    }
}
if (in != null){
    try{
        in.close();
    } catch (Exception e){
    }

    out = null;
    in = null;
}
try{
    clientSocket.close();
} catch (Exception e){
}

clientSocket = null;

Report("Stopped.");

private void ClientServe(BufferedReader in, PrintWriter out){
    final int UNINITIALIZED = 0;
    final int PROPHET = 1;
    final int IDS = 2;
    final int MESSAGES = 3;
    final int FINISHED = 4;
    final int CLOSED = 5;
    final int ERROR = 6;
    int status = UNINITIALIZED;
    partnerIDs = null;
    prophetIDs = null;
    String line;

    Scanner commandScanner;
    String command;
    try{
        int failures = 0;
        while (status == UNINITIALIZED){
            out.println("NODE " + NodeID);
            if (in.ready()){  
                line = in.readLine();
                commandScanner = new Scanner(line);
                if (commandScanner.hasNext()){
                    command = commandScanner.next();
                    if (!command.equalsIgnoreCase("NODE")){
 ...

line = in.readLine();
}

if (line.length() > 5)
    partnerNodeID = line.substring(5);
messages.updateProphet(partnerNodeID);
} else {
    line = in.readLine();
}

if (!partnerNodeID.equalsIgnoreCase(""))
    status = PROPHET;
    Report("Connected to Node " + partnerNodeID);
} else {
    if (failures >= 10)
        return;
    failures++;
    Thread.sleep(Math.round(Math.random() * 50) + 50);
}

while ((status != UNINITIALIZED) && (status != CLOSED) && (status != ERROR))
    switch (status) {
        case PROPHET:
            if (!exchangeProphet(in, out))
                out.println("CLOSE");
                status = ERROR;
            } else {
                status = IDS;
            }
        break;

        case IDS:
            if (!exchangeMessageIDs(in, out))
                out.println("CLOSE");
                status = ERROR;
            } else {
                status = MESSAGES;
            }
        break;

        case MESSAGES:
            if (!exchangeMessages(in, out))
                out.println("CLOSE");
                status = ERROR;
            } else {
                status = FINISHED;
            }
        break;

        case FINISHED:
            out.println("CLOSE");
status = CLOSED;
}
}
messages.deliverMessages(partnerNodeID);
messages.DecrementAllTTL();

} catch (Exception e){
}
}

private boolean exchangeProphet(BufferedReader in, PrintWriter out) throws IOException{
    out.println("PROPHET");
    out.println(messages.AllProphets());
    out.println("ENDPROPHET");
    Report("Sent Prophet Statistics to " + partnerNodeID + "\n\n____________________________\n\n" + messages.AllProphets() + "____________________________\n");
    Scanner lineScanner;
    String line = "";
    prophetIDs = new ArrayList<String>();
    ArrayList<String> keys = new ArrayList<String>();
    ArrayList<Double> values = new ArrayList<Double>();
    String key;
    double value;
    while (!line.equalsIgnoreCase("ENDPROPHET")){
        while (((line.equalsIgnoreCase("PROPHET")) || (line.equalsIgnoreCase("")))){
            line = in.readLine();
        }
        lineScanner = new Scanner(line);
        if (lineScanner.hasNext()){ 
            key = lineScanner.next();
            if (lineScanner.hasNextDouble()){ 
                value = lineScanner.nextDouble();
                keys.add(key);
                values.add(value);
            }
        }
        line = in.readLine();
        lineScanner.close();
    }
    if (keys.size() != values.size()){
        return false;
    }
    String received = "";
    ArrayList<String> prophetKeys = messages.getProphetKeys();
    for (int i = 0; i < prophetKeys.size(); i++){
        if (!keys.contains(prophetKeys.get(i))){
            prophetIDs.add(prophetKeys.get(i));
        }
    }
    for (int i = 0; i < keys.size(); i++){
        if (messages.getProphet(keys.get(i)) > values.get(i)){
            prophetIDs.add(keys.get(i));
        }
    }
    updateProphet(keys.get(i), values.get(i));
    received = received + keys.get(i) + " " + values.get(i) + "\n";
private void updateProphet(String key, Double value) {
    if (!key.equalsIgnoreCase(NodeID)) {
        messages.transitiveProphetUpdate(key, partnerNodeID, value);
    }
}

private boolean exchangeMessageIDs(BufferedReader in, PrintWriter out) throws IOException {
    out.println("IDS");
    out.println(messages.ActiveMessageIDs());
    out.println("ENDIDS");
    Report("Sent Message IDs to " + partnerNodeID + 
    "____________________________
    + messages.ActiveMessageIDs() + "____________________________");
    String line = ""
    ArrayList<String> IDs = new ArrayList<String>;
    while (!line.equalsIgnoreCase("ENDIDS") && (line != null)) {
        line = in.readLine();
        while ((line.equalsIgnoreCase("IDS")) || (line.equalsIgnoreCase(""))) {
            line = in.readLine();
        }
        if (!line.equalsIgnoreCase("ENDIDS")) {
            IDs.add(line);
        }
    }
    Report("Received Message IDs from " + partnerNodeID + 
    "____________________________
    + IDs + "____________________________");
    partnerIDs = new ArrayList<String>(IDs);
    //System.out.println("CLIENT " + NodeID + " RECEIVED IDs\n" + partnerIDs.toString());
    return true;
}

private boolean exchangeMessages(BufferedReader in, PrintWriter out) throws IOException {
    requestMessages(in, out);
    String line = ""
    Scanner commandScanner;
    String command;
    while (!((line.equalsIgnoreCase("ENDREQUEST")) && (line != null))) {
        if (line.equalsIgnoreCase("REQUEST") || (line.equalsIgnoreCase(""))) {
            line = in.readLine();
        }
        commandScanner = new Scanner(line);
        if (commandScanner.hasNext()) {
            command = commandScanner.next();
            if (command.equalsIgnoreCase("MESSAGE")){
                }
    }
}
if (line.length() > 8)
{
    if (messages.getMessage(line.substring(8)) == null)
    {
        out.println("NOMESSAGE");
    } else {
        out.println("PROVIDE "+ line.substring(8));
        out.print(messages.getMessage(line.substring(8)));
        out.println("ENDPROVIDE "+ line.substring(8));
    }
}
}
}  
line = in.readLine();
commandScanner.close();

return true;

private boolean ERPRequest(BufferedReader in, PrintWriter out) throws IOException{
    String receivedMessage;
    Scanner commandScanner;
    String command;
    //System.out.println(NodeID + " has messages\n" + messages.AllMessageIDs() + "________");
    out.println("REQUEST");

    for(int i = 0; i < partnerIDs.size(); i++)
    {
        receivedMessage = "";
        //System.out.print(NodeID + " examining " + partnerIDs.get(i) + "-");
        if (messages.getMessage(partnerIDs.get(i)) == null)
        {
            //System.out.print(" retrieve");
            out.println("MESSAGE "+ partnerIDs.get(i));
            String line = in.readLine();

            commandScanner = new Scanner(line);
            if (commandScanner.hasNext())
            {
                command = commandScanner.next();

                if (command.equalsIgnoreCase("PROVIDE"))
                {
                    while (!command.equalsIgnoreCase("ENDPROVIDE"))
                    {
                        line = in.readLine();
                        commandScanner = new Scanner(line);
                        if (commandScanner.hasNext())
                        {
                            command = commandScanner.next();
                        }

                        if (!command.equalsIgnoreCase("ENDPROVIDE"))
                        {
                            if (!receivedMessage.equals(""))
                            {
                                receivedMessage = receivedMessage + "\n";
                            }
                            receivedMessage = receivedMessage + line;
                        }
                    }
                }
            }
        }  
    }

    //System.out.println();
    //System.out.println(NodeID + " Got Message --" + receivedMessage + "--");
    addMessage(receivedMessage);
    commandScanner.close();
}

out.println("ENDREQUEST");
return true;
private String parseDestination(String messageID)
{
    String destination = "";
    Scanner IDScanner = new Scanner(messageID);
    if(IDScanner.hasNext()){
        IDScanner.next();
        if (IDScanner.hasNext()){
            IDScanner.next();
            if (IDScanner.hasNext()){
                destination = IDScanner.next();
            }
        }
    }
    IDScanner.close();
    return destination;
}

private boolean ProphetRequest(BufferedReader in, PrintWriter out) throws IOException{
    out.println("REQUEST");
    String receivedMessage;
    Scanner commandScanner;
    String command;
    for(int i = 0; i < partnerIDs.size(); i++){
        if (prophetIDs.contains(parseDestination(partnerIDs.get(i))){
            receivedMessage = "";
        if (messages.getMessage(partnerIDs.get(i)) == null){
            out.println("MESSAGE " + partnerIDs.get(i));
            String line = in.readLine();
            commandScanner = new Scanner(line);
            if (commandScanner.hasNext()){
                command = commandScanner.next();
                if (command.equalsIgnoreCase("PROVIDE")){
                    line = in.readLine();
                    commandScanner = new Scanner(line);
                    if (commandScanner.hasNext()){
                        command = commandScanner.next();
                        if (!receivedMessage.equals("")){
                            receivedMessage = receivedMessage + "\n";
                        }
                        if (!command.equalsIgnoreCase("ENDPROVIDE")){
                            receivedMessage = receivedMessage + line;
                        }
                    }
                    commandScanner.close();
                }
            }
        //System.out.println(NodeID + " Got Message --> " + receivedMessage + " -->");
        addMessage(receivedMessage);
        }
        commandScanner.close();
    }
    out.println("ENDREQUEST");
    return true;
}

private void addMessage(String receivedMessage){
}
String source;
String destination;
int sequence;
int TTL;
Scanner messageParser = new Scanner(receivedMessage);
messageParser.useDelimiter(";");
if (messageParser.hasNext()){
    source = messageParser.next();
    if (messageParser.hasNext()){
        destination = messageParser.next();
        if (messageParser.hasNextInt()){
            sequence = messageParser.nextInt();
            if (messageParser.hasNextInt()){
                TTL = messageParser.nextInt();
                messages.AddMessage(source, destination, sequence, TTL);
            }
        }
    }
}
messageParser.close();
//System.out.println(NodeID + " SERVER MESSAGE\n" + receivedMessage);

private boolean requestMessages(BufferedReader in, PrintWriter out) throws IOException{
    if (messages.averageProphet() < threshold){
        Report("Average Prophet value = " + messages.averageProphet() + " < threshold (" + threshold + "). Using ERP.");
        messageManager.reportPropagationType(NodeID, partnerNodeID, 2);
        return ERPRequest(in, out);
    } else {
        Report("Average Prophet value = " + messages.averageProphet() + " >= threshold (" + threshold + "). Using Prophet.");
        messageManager.reportPropagationType(NodeID, partnerNodeID, 1);
        return ProphetRequest(in, out);
    }
}

//END CLASS DTNConnector

//BEGIN CLASS DTNJDBCConnector
//STEP 1. Import required packages
import java.sql.*;
public class DTNJDBCConnector
{
    private void Report(String message){
        //System.out.println(message);
    }

    // JDBC driver name and database URL
    static final String JDBC_DRIVER = "oracle.jdbc.driver.OracleDriver";

    static final String DB_URL = "jdbc:oracle:thin:@apollo.ite.gmu.edu:1521:ite10g";

    // Database credentials
    static final String USER = "infs612";
static final String PASS = "spring2013";

Connection conn = null;

public PreparedStatement createConnection(String sql)
{
    PreparedStatement stmt = null;
    try {
        // STEP 2: Register JDBC driver
        Class.forName("oracle.jdbc.driver.OracleDriver");

        // STEP 3: Open a connection
        Report("Connecting to database...");
        try {
            conn = DriverManager.getConnection(DB_URL, USER, PASS);
        }
        catch (SQLException e) {
            e.printStackTrace();
        }
        // STEP 4: Execute a query
        Report("Creating statement...");
        stmt = conn.prepareStatement(sql);
    }
    catch (SQLException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    }
    // STEP 4: Execute a query
    Report("Creating statement...");
    stmt = conn.prepareStatement(sql);
    }
    catch (SQLException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    }
    catch (ClassNotFoundException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    }
    finally {
    }
    return stmt;
}

public void release() throws SQLException
{
    if(conn != null && !conn.isClosed())
    conn.commit();
    conn.close();
}

/*
public static void main(String[] args) {
    Connection conn = null;
    Statement stmt = null;
    try{
        //STEP 2: Register JDBC driver
        Class.forName("oracle.jdbc.driver.OracleDriver");

        //STEP 3: Open a connection
        System.out.println("Connecting to database...");
        try {
            conn = DriverManager.getConnection(DB_URL, USER, PASS);
        } catch (SQLException e) {
            e.printStackTrace();
            return;
        }

        //STEP 4: Execute a query
        System.out.println("Creating statement...");
        stmt = conn.createStatement();
        String sql;
        sql = "SELECT ROLE_CD, ROLE_DESC FROM PEC_ROLE_RFRNC";
        ResultSet rs = stmt.executeQuery(sql);

        //STEP 5: Extract data from result set
        while(rs.next()){
            //Retrieve by column name
            String roleCd = rs.getString("ROLE_CD");
            String roleDesc = rs.getString("ROLE_DESC");

            //Display values
            System.out.print("Role Code: " + roleCd);
            System.out.print(", Role Desc: " + roleDesc);
        }

        //STEP 6: Clean-up environment
        rs.close();
        stmt.close();
        conn.close();
    } catch(SQLException se){
        //Handle errors for JDBC
        se.printStackTrace();
    } catch(Exception e){
        //Handle errors for Class.forName
        e.printStackTrace();
    } finally{
        //finally block used to close resources
        try{
            if(stmt!=null)
                stmt.close();
        } catch(SQLException se2){
            // nothing we can do
            try{
                if(conn!=null)
                    conn.close();
            } catch(SQLException se){

            }
    }
}
se.printStackTrace();
} // end finally try
} // end try
System.out.println("Goodbye!");
} // end main
} // end FirstExample
*/

// END CLASS DTNJDBCConnector

// BEGIN CLASS DTNLListener
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.PrintWriter;
import java.net.ServerSocket;
import java.net.Socket;
import java.util.ArrayList;
import java.util.Scanner;

// This class should serve as the "server" portion of a connection
public class DTNLListener implements Runnable {

    private MessageStatistics messages;
    DTNMessageManager messageManager;
    private int port;
    private ServerSocket listener;
    private Socket clientSocket;
    private boolean listening;
    private String NodeID;
    private ArrayList<String> partnerIDs;
    private ArrayList<String> prophetIDs;
    private double threshold;

    String partnerNodeID;

    private void Report(String message){
        System.out.println(NodeID + \"[DTNListener] - \" + message);
    }

    public DTNLListener(String NodeID, int port, MessageStatistics messages, double threshold){
        messageManager = new DTNMessageManager(messages, threshold);
        this.threshold = threshold;
        this.messages = messages;
        this.port = port;
        listener = null;
        clientSocket = null;
        listening = true;
        this.NodeID = NodeID;
        partnerIDs = null;
        prophetIDs = null;
    }

    public void close(){
        listening = false;
    }

    public void run(){
        Report("Started.");
        try{
            listener = new ServerSocket(port);
        }
    }
}
catch (IOException e) {
    System.out.println("Server - Could not listen on port "+ port);
    System.exit(-1);
}

PrintWriter out = null;
BufferedReader in = null;

while (listening){
    partnerIDs = null;
    prophetIDs = null;

    try{
        listener.setSoTimeout(2000);
        clientSocket = listener.accept();
    }
    catch (IOException e){
        clientSocket = null;
    }
    if (clientSocket != null){
        try{
            out = new PrintWriter(clientSocket.getOutputStream(), true);
        }
        catch (IOException e){
            out = null;
        }
        try{
            in = new BufferedReader(new InputStreamReader(clientSocket.getInputStream()));
        }
        catch (IOException e){
            in = null;
        }
        if ((out != null) && (in != null)){
            ServeClient(in, out);

            messages.ageAllProphet();

            messageManager.reportDeliveredMessages(NodeID);
            messageManager.reportUndeliveredMessages(NodeID);
            messageManager.reportMessageBuffer(NodeID);
        } else {
            if (out != null){
                try{
                    out.close();
                } catch (Exception e){
                    
                }
            }
            if (in != null){
                try{
                    in.close();
                } catch (Exception e){
                    
                }
            }
        }
    }
}
if (listener != null){
  try{
    listener.close();
    listener = null;
  } catch (Exception e){
    //System.exit(-1);
  }
}
Report("Stopped.");

private void ServeClient(BufferedReader in, PrintWriter out){
  final int UNINITIALIZED = 0;
  final int INITIALIZED = 1;
  final int CLOSED = 2;
  final int ERROR = 3;
  int status = UNINITIALIZED;
  String line;
  //String partnerNodeID = "";
  Scanner commandScanner;
  String command;
  try{
    int failures = 0;
    while (status == UNINITIALIZED){
      out.println("NODE " + NodeID);
      if (in.ready()){
        line = in.readLine();
        //System.out.println("Server received: " + line);
        commandScanner = new Scanner(line);
        if (commandScanner.hasNext()){
          command = commandScanner.next();
          if (!command.equalsIgnoreCase("NODE")){
            line = in.readLine();
          } else {
            // Error handling
          }
        } else {
          // Error handling
        }
      }
      // Handle other status values
    }
  } catch (Exception e){
    //Error handling
  }
}
if (line.length() > 5){
    partnerNodeID = line.substring(5);
    messages.updateProphet(partnerNodeID);
}

} else {
    line = in.readLine();
}

}

if (!partnerNodeID.equalsIgnoreCase("")) {
    status = INITIALIZED;
    Report("Connected to Node " + partnerNodeID);
} else {
    if (failures >= 10){
        return;
    }
    failures++;
    Thread.sleep(Math.round(Math.random()*50)+50);
}

}

while (status == INITIALIZED){
    line = in.readLine();
    commandScanner = new Scanner(line);
    if (commandScanner.hasNext()){
        command = commandScanner.next();
        if (command.equalsIgnoreCase("PROPHET")){
            if (!exchangeProphet(in,out)){
                out.println("CLOSE");
                status = ERROR;
            } else {
                status = INITIALIZED;
            }
        }
        if (command.equalsIgnoreCase("IDS")){
            if (!exchangeMessageIDs(in,out)){
                out.println("CLOSE");
                status = ERROR;
            } else {
                status = INITIALIZED;
            }
        }
        if (command.equalsIgnoreCase("REQUEST")){
            if (!exchangeMessages(in,out)){
                out.println("CLOSE");
                status = ERROR;
            } else {
                status = INITIALIZED;
            }
        }
    }
}

Page 29 of 49
if (command.equalsIgnoreCase("CLOSE")){
    out.println("CLOSE");
    status = CLOSED;
}

//System.out.println(status);

messages.deliverMessages(partnerNodeID);
messages.DecrementAllTTTL();
}

private boolean exchangeProphet(BufferedReader in, PrintWriter out) throws IOException{
    out.println("PROPHET");
    out.print(messages.AllProphets());
    out.println("ENDPROPHET");
    Report("Sent Prophet Statistics to " + partnerNodeID + "\n" +
    messages.AllProphets() + "\n" +
    Scanner lineScanner;
    String line = "";
    prophetIDs = new ArrayList<String>();
    ArrayList<String> keys = new ArrayList<String>();
    ArrayList<Double> values = new ArrayList<Double>();
    String key;
    double value;
    while (!line.equalsIgnoreCase("ENDPROPHET")){
        while ((line.equalsIgnoreCase("PROPHET")) || (line.equalsIgnoreCase("")) || (line.equalsIgnoreCase(""))){
            line = in.readLine();
        }
        //System.out.println(line);
        lineScanner = new Scanner(line);
        if (lineScanner.hasNext()){  
            key = lineScanner.next();
            if (lineScanner.hasNextDouble()){  
                value = lineScanner.nextDouble();
                keys.add(key);
                values.add(value);
            }
        }
        line = in.readLine();
        lineScanner.close();
    }
    if (keys.size() != values.size()){  
        //System.out.println("MISMATCH");
        return false;
    }
    String received = "";
    if (!received.endsWith(prophetIDs)){
        return false;
    }
    Arraylist<String> prophetKeys = messages.getProphetKeys();
    for (int i = 0; i < prophetKeys.size(); i++){
        if (!keys.contains(prophetKeys.get(i))){
if (!prophetIDs.contains(NodeID)){
    prophetIDs.add(NodeID);
}
for (int i = 0; i<keys.size(); i++){
    if (messages.getProphet(keys.get(i)) > values.get(i)){
        prophetIDs.add(keys.get(i));
    }
    updateProphet(keys.get(i), values.get(i));
    received = received + keys.get(i) + " " + values.get(i) + "\n";
}
Report("Received Prophet Statistics from " + partnerNodeID + "\n____________________________\n" + received + "\n_________________________");
return true;
}
private void updateProphet(String key, Double value){
    if (!key.equalsIgnoreCase(NodeID)){
        messages.transitiveProphetUpdate(key, partnerNodeID, value );
    }
}
private boolean exchangeMessageIDs(BufferedReader in, PrintWriter out) throws IOException{
  //System.out.println("Server Sends Messages");
  out.println("IDS");
  out.println(messages.ActiveMessageIDs());
  out.println("ENDIDS");
  Report("Sent Message IDs to " + partnerNodeID + "\n____________________________\n" + messages.ActiveMessageIDs() + "\n_________________________");
  //Scanner lineScanner;
  String line = "";
  ArrayList<String> IDs = new ArrayList<String>();
  //String ID;
  while (!line.equalsIgnoreCase("ENDIDS") && (line != null)){
      line = in.readLine();
      //System.out.println("SERVER reads a line");
      //System.out.println("SERVER read " + line);
      while ((line.equalsIgnoreCase("IDS")) || (line.equalsIgnoreCase(""))){
          //System.out.println("SERVER reads a line");
          line = in.readLine();
          //System.out.println("SERVER read " + line);
      }
      if (!line.equalsIgnoreCase("ENDIDS")){
          IDs.add(line);
      }
  }
  Report("Received Message IDs from " + partnerNodeID + "\n____________________________\n" + IDs + "\n_________________________");
  //System.out.println("SERVER Made it out");
  partnerIDs = new ArrayList<String>(IDs);
  }
private boolean exchangeMessages(BufferedReader in, PrintWriter out) throws IOException {
    String line = in.readLine();
    Scanner commandScanner;
    String command;
    while (!line.equalsIgnoreCase("ENDREQUEST") && (line != null)) {
        commandScanner = new Scanner(line);
        if (commandScanner.hasNext()) {
            command = commandScanner.next();
            if (command.equalsIgnoreCase("MESSAGE") && line.length() > 8) {
                if (messages.getMessage(line.substring(8)) == null) {
                    out.println("NOMESSAGE");
                } else {
                    out.println("PROVIDE " + line.substring(8));
                    out.println(messages.getMessage(line.substring(8)));
                    out.println("ENDPROVIDE " + line.substring(8));
                }
            } else {
                //out.println("Server received: " + line);
                line = in.readLine();
                commandScanner.close();
            }
        }
    }
    return requestMessages(in, out);
}

private String parseDestination(String messageID) {
    String destination = "";
    Scanner IDScanner = new Scanner(messageID);
    if (IDScanner.hasNext()) {
        IDScanner.next();
        if (IDScanner.hasNext()) {
            IDScanner.next();
            if (IDScanner.hasNext()) {
                destination = IDScanner.next();
            }
        }
    }
    IDScanner.close();
    return destination;
}

private boolean ProphetRequest(BufferedReader in, PrintWriter out) throws IOException {
    out.println("REQUEST");
    String receivedMessage;
    Scanner commandScanner;
    String command;
    for (int i = 0; i < partnerIDs.size(); i++) {
        if (prophetIDs.contains(parseDestination(partnerIDs.get(i)))) {
            receivedMessage = "";
            if (messages.getMessage(partnerIDs.get(i)) == null) {
                out.println("MESSAGE " + partnerIDs.get(i));
                String line = in.readLine();
            }
        }
    }
    return true;
}
commandScanner = new Scanner(line);
if (commandScanner.hasNext()){
    command = commandScanner.next();
    if (command.equalsIgnoreCase("PROVIDE")){
        while (!command.equalsIgnoreCase("ENDPROVIDE")){
            line = in.readLine();
            commandScanner = new Scanner(line);
            if (commandScanner.hasNext()){
                command = commandScanner.next();
            }  
            if (!command.equalsIgnoreCase("ENDPROVIDE")){
                if (!receivedMessage.equals("")){
                    receivedMessage = receivedMessage + "\n";
                }
                receivedMessage = receivedMessage + line;
            }
        }
        //System.out.println(NodeID + " Got Message --" + receivedMessage + "--");
        addMessage(receivedMessage);
        commandScanner.close();
    }
}
out.println("ENDREQUEST");
return true;
}

private void addMessage(String receivedMessage){
    String source;
    String destination;
    int sequence;
    int TTL;
    Scanner messageParser = new Scanner(receivedMessage);
    messageParser.useDelimiter(";");
    if (messageParser.hasNext()){
        source = messageParser.next();
        if (messageParser.hasNext()){
            destination = messageParser.next();
            if (messageParser.hasNextInt()){
                sequence = messageParser.nextInt();
                if (messageParser.hasNextInt()){
                    TTL = messageParser.nextInt();
                    messages.AddMessage(source, destination, sequence, TTL);
                }
            }
        }
    }
    messageParser.close();  
    //System.out.println(NodeID + " SERVER MESSAGE\n" + receivedMessage);
}

private boolean ERPRequest(BufferedReader in, PrintWriter out) throws IOException{
    String receivedMessage;
    Scanner commandScanner;
    String command;
    //System.out.println(NodeID + " has messages\n" + messages.AllMessageIDs() + "________");
out.println("REQUEST");
for(int i = 0; i < partnerIDs.size(); i++)
    receivedMessage = "";
    //System.out.print(NodeID + " examining " + partnerIDs.get(i) + "-");
    if (messages.getMessage(partnerIDs.get(i)) == null)
        //System.out.print(" retrieve");
        out.println("MESSAGE " + partnerIDs.get(i));
        String line = in.readLine();
        commandScanner = new Scanner(line);
        if (commandScanner.hasNext()){
            command = commandScanner.next();
            if (command.equalsIgnoreCase("PROVIDE")){
                while (!command.equalsIgnoreCase("ENDPROVIDE")){
                    line = in.readLine();
                    commandScanner = new Scanner(line);
                    if (commandScanner.hasNext()){
                        command = commandScanner.next();
                    }
                    if (!command.equalsIgnoreCase("ENDPROVIDE")){
                        if (!receivedMessage.equals("")){
                            receivedMessage = receivedMessage + "\n";
                        }
                        receivedMessage = receivedMessage + line;
                    }
                }
            }
        }
        //System.out.println();
        //System.out.println(NodeID + " Got Message -- " + receivedMessage + "--");
        addMessage(receivedMessage);
        commandScanner.close();
    }
out.println("ENDREQUEST");
return true;
}

private boolean requestMessages(BufferedReader in, PrintWriter out) throws IOException{
if (messages.averageProphet() < threshold)
    Report("Average Prophet value = " + messages.averageProphet() + " < threshold (" + threshold + ").
Using ERP.");
    messageManager.reportPropagationType(NodeID, partnerNodeID, 2);
    return ERPRequest(in, out);
} else {
    Report("Average Prophet value = " + messages.averageProphet() + " >= threshold (" + threshold + ").
Using Prophet.");
    messageManager.reportPropagationType(NodeID, partnerNodeID, 1);
    return ProphetRequest(in,out);
}

//END CLASS DTNListener

//BEGIN CLASS DTNMessageManager
import java.io.*;
import java.util.*;
import java.sql.*;

public class DTNMessageManager{
    private MessageStatistics messages;
    private double threshold;
    //private DTNJDBCConnector conn;

    private void Report(String message){
        System.out.println(message);
    }

    public DTNMessageManager(MessageStatistics messages, double threshold){
        this.messages = messages;
        this.threshold = threshold;
        //this.conn = new DTNJDBCConnector();
    }

    //For all of the report functions, NodeID is the NodeID of the node that called the function.
    public boolean reportDeliveredMessages(String NodeID){
        //System.out.println("Reporting any delivered messages");
        DTNJDBCConnector conn = new DTNJDBCConnector();
        ArrayList<String> messages2 = messages.deliverMessages(NodeID);
        String source;
        String destination;
        int sequence;
        int TTL;
        String sql;
        String sql = "INSERT INTO DELIVERED_MESSAGE_RPT (RPT_SK, SOURCE, DESTINATION, MESSAGE_ID, TIME_TO_LIVE, THRESHOLD)"
            + " VALUES (RPT_SEQ.NEXTVAL, ?, ?, ?, ?, ?)"
            + ";";
        PreparedStatement stmt = conn.createConnection(sql);
        Scanner messageParser;
        for(int x = 0; x < messages2.size(); x++) {
            messageParser = new Scanner(messages2.get(x));
            messageParser.useDelimiter(";");
            if (messageParser.hasNext()){
                source = messageParser.next();
                if (messageParser.hasNext()){
                    destination = messageParser.next();
                    if (messageParser.hasNextInt()){
                        sequence = messageParser.nextInt();
                        if (messageParser.hasNextInt()){
                            TTL = messageParser.nextInt();
                            try {
                                stmt.setString(1, source);
                                stmt.setString(2, destination);
                                stmt.setInt(3, sequence);
                                stmt.setInt(4, TTL);
                                stmt.setDouble(5, threshold);
                                stmt.executeUpdate();
                                //System.out.println("Delivered a message");
                            } catch (SQLException e) {
                                e.printStackTrace();
                            }
                            System.out.println ("Source: " + source + " Destination: " +
                                destination + "seq: " + sequence + "TTL: " + TTL);  
                        }
                    }
                }
            }
        }
    }
}

Page 35 of 49
messageParser.close();

//if (mailline.getMessageDestination(NodeID) = destination) {
//}
//one per line string
//call deliverMessages(nodeid);
//called at the end of a communication session.
//This should make a record of all messages where getMessageDestination(messageID) = NodeID
try{
    if(stmt != null)
        stmt.close();
    conn.release();
} catch (Exception e){
}

if (messages2.size() > 0)
    Report("Delivered " + messages2.size() + " messages to " + NodeID);
return true;
}

public boolean reportUndeliveredMessages(String NodeID){
    DTNJDBCConnector conn = new DTNJDBCConnector();
    ArrayList<String> messages2 = messages.removeDeadMessages();
    String source;
    String destination;
    int sequence;
    int TTL;
    String sql;
    sql = "INSERT INTO UNDELIVERED_MESSAGE_RPT (RPT_SK, SOURCE, DESTINATION,
MESSAGE_ID, THRESHOLD)"
        + " VALUES (RPT_SEQ.NEXTVAL, ?, ?, ?, ?)";
    PreparedStatement stmt = conn.createConnection(sql);
    Scanner messageParser;
    for(int x = 0 ; x < messages2.size(); x++) {
        messageParser = new Scanner(messages2.get(x));
        messageParser.useDelimiter(";");
        if (messageParser.hasNext()){
            source = messageParser.next();
            if (messageParser.hasNext()){
                destination = messageParser.next();
                if (messageParser.hasNextInt()){
                    sequence = messageParser.nextInt();
                    if (messageParser.hasNextInt()){
                        TTL = messageParser.nextInt();
                        try {
                            stmt.setString(1, source);
                            stmt.setString(2, destination);
                            stmt.setInt(3, sequence);
                            stmt.setDouble(4, threshold);
                            stmt.executeUpdate();
                            //System.out.println("Couldn't deliver a message");
                        }
                    }
                }
            }
        }
    }
}
} //System.out.println ("Source: " + source + " " + "Destination: " +
destination + "seq: " + sequence + "TTL: " + TTL);
}
messageParser.close();
} //removeDeadmessages
//called at the end of a communication session.
//This should make a record of all messages where getMessageTTL(messageID) <= 0
try{
    if(stmt != null)
        stmt.close();
    conn.release();
} catch (Exception e){

}
if (messages2.size() > 0)
    Report(messages2.size() + " undeliverable messages at " + NodeID);
return true;
}

public boolean reportPropagationType(String NodeID, String partnerNodeID, int type) {
    //called at the end of a communication session.
    //This should report which type of communication was used, given by the type
    //parameter. 1=Prophet, 2=ERP
    DTNJDBCConnector conn = new DTNJDBCConnector();
    String sql;
    sql = "INSERT INTO PROPAGATION_TYPE_RPT (RPT_SK, NODE1, NODE2, STAT_TYPE,
    THRESHOLD)"
    + " VALUES (RPT_SEQ.NEXTVAL, ?, ?, ?, ?)";
    PreparedStatement stmt = conn.createConnection(sql);
    try {
        stmt.setString(1, NodeID);
        stmt.setString(2, partnerNodeID);
        stmt.setInt(3, type);
        stmt.setDouble(4, threshold);
        stmt.executeUpdate();
    } catch (Exception e) {
        //e.printStackTrace();
    }

    //called at the end of a communication session.
    //This should report which type of communication was used, given by the type
    //parameter. 1=Prophet, 2=ERP

try {
    if(stmt != null) {
        stmt.close();
    }
    conn.release();
} catch (Exception e) {
    //e.printStackTrace();
}
return true;
}

public boolean reportMessageBuffer(String NodeID) {
    int messagecount = messages.messageCount();

    DTNjdbcConnector conn = new DTNjdbcConnector();
    String sql;
    sql = "INSERT INTO MESSAGE_BUFFER_RPT (RPT_SK, NODEID, MESSAGE_COUNT, THRESHOLD)"
        + " VALUES (RPT_SEQ.NEXTVAL, ?, ?, ?)";
    PreparedStatement stmt = conn.createConnection(sql);
    try {
        stmt.setString(1, NodeID);
        stmt.setInt(2, messagecount);
        stmt.setDouble(3, threshold);
        stmt.executeUpdate();
    } catch (Exception e) {
        //e.printStackTrace();
    }
    return true;
}

//messageCount()
//called at the end of a communication session.
//This should report the total number of messages currently in the buffer.
try {
    if(stmt != null) {
        stmt.close();
    }
    conn.release();
} catch (Exception e) {
    //e.printStackTrace();
}
return true;

//END CLASS DTNMessageManager
/BEGIN CLASS DTNNode
import java.io.BufferedWriter;
import java.io.File;
import java.io.FileWriter;
import java.io.Writer;
import java.sql.*;

public class DTNNode {

    static final int connectionfactor = 100;

    private DTNListener server;
    private DTNConnector client;
    private String NodeID;

    private void Report(String message){
        System.out.println(NodeID + " - " +message);
    }

    public DTNNode(String NodeID, int listenPort, double threshold){
        this.NodeID = NodeID;
        MessageStatistics messages = new MessageStatistics();
        ConnectionList connections;
        DTNJDBCConnector conn = new DTNJDBCConnector();
        PreparedStatement query = conn.createConnection("SELECT NODE2_NAME, CONN_PROBABILITY FROM NODE_STATS WHERE NODE1_NAME = "+ NodeID + "");
            try{
                ResultSet rs = query.executeQuery();
                connections = new ConnectionList(rs, connectionfactor);
                query.close();
            } catch (Exception e) {
                System.out.println(e.getMessage());
                connections = new ConnectionList();
            }

        query = conn.createConnection("SELECT SENDER_ID, RECEIVER_ID, MESSAGE_ID, TIME_TO_LIVE FROM MESSAGES WHERE SENDER_ID = " + NodeID + "");
            try{
                String source;
                String dest;
                int seq;
                int ttl;
                ResultSet rs = query.executeQuery();
                while (rs.next()){
                    source = rs.getString("SENDER_ID");
                    dest = rs.getString("RECEIVER_ID");
                    seq = rs.getInt("MESSAGE_ID");
                    ttl = rs.getInt("TIME_TO_LIVE");
                    messages.AddMessage(source, dest, seq, ttl);
                }
                if (rs!=null)
                    rs.close();
                if (query!=null);
                query.close();
                conn.release();
            }
}
} catch (Exception e) {
    //Report("Error adding message");
}
//messages.AddMessage(NodeID,"127.0.0.1:12340",10000,128);
server = new DTNListener(NodeID, listenPort, messages, threshold);
client = new DTNConnector(NodeID, connections, messages, threshold);
Thread listener = new Thread(server);
Thread connector = new Thread(client);
listener.start();
Report("DTNListener starting.");
connector.start();
Report("DTNConnector starting.");
}

public DTNNode(String NodeID, int listenPort, String partnerHost, int partnerPort, double threshold){
    this.NodeID = NodeID;
    MessageStatistics messages = new MessageStatistics();
    ConnectionList connections = new ConnectionList(partnerHost, partnerPort, connectionfactor);
    messages.AddMessage(NodeID,"127.0.0.1:" + partnerPort,10000,128);
    messages.AddMessage("A","B",partnerPort,128);
    server = new DTNListener(NodeID, listenPort, messages, threshold);
    client = new DTNConnector(NodeID, connections, messages, threshold);
    Thread listener = new Thread(server);
    Thread connector = new Thread(client);
    listener.start();
    Report("DTNListener starting.");
    connector.start();
    Report("DTNConnector starting.");
}

public void end(){
    server.close();
    Report("DTNListener shutting down.");
    client.close();
    Report("DTNConnector shutting down.");
}

public static void dumpTables(){
    DTNJDBCConnector conn;
    PreparedStatement query;
    ResultSet rs;
    String source;
    String dest;
    int seq;
    int ttl;
    double threshold;
    conn = new DTNJDBCConnector();
    query = conn.createConnection("SELECT * FROM DELIVERED_MESSAGE_RPT");
    String deliveredMessages = "SOURCE,DESTINATION,MESSAGE_ID,TIME_TO_LIVE,THRESHOLD\n";
    try{
        rs = query.executeQuery();
        while (rs.next()){
            source = rs.getString("SOURCE");
            dest = rs.getString("DESTINATION");
        }
    }
seq = rs.getInt("MESSAGE_ID");
ttl = rs.getInt("TIME_TO_LIVE");
threshold = rs.getDouble("THRESHOLD");
deliveredMessages = deliveredMessages + source + "," + dest + "," + seq + "," + ttl + "," + threshold + "\n";
}
if (rs!=null)
    rs.close();
if (query!=null)
    query.close();
conn.release();
writeTextFile("./DeliveredMessages.csv", deliveredMessages);
} catch (Exception e) {
    //Report("Error adding message");
}

conn = new DTNJDBCConnector();
query = conn.createConnection("SELECT * FROM UNDELIVERED_MESSAGE_RPT");
String undeliveredMessages = "SOURCE,DESTINATION,MESSAGE_ID,THRESHOLD\n";
try{
    rs = query.executeQuery();
    while (rs.next()){
        source = rs.getString("SOURCE");
        dest = rs.getString("DESTINATION");
        seq = rs.getInt("MESSAGE_ID");
        threshold = rs.getDouble("THRESHOLD");
        undeliveredMessages = undeliveredMessages + source + "," + dest + "," + seq + "," + threshold + "," + 
threshold + "\n";
    }
    if (rs!=null)
        rs.close();
    if (query!=null)
        query.close();
    conn.release();
    writeTextFile("./UndeliveredMessages.csv", undeliveredMessages);
} catch (Exception e) {
    //Report("Error adding message");
}

conn = new DTNJDBCConnector();
query = conn.createConnection("SELECT * FROM PROPAGATION_TYPE_RPT");
String propagationType = "SOURCE,DESTINATION,TYPE,THRESHOLD\n";
try{
    rs = query.executeQuery();
    while (rs.next()){
        source = rs.getString("NODE1");
        dest = rs.getString("NODE2");
        seq = rs.getInt("STAT_TYPE");
        threshold = rs.getDouble("THRESHOLD");
        propagationType = propagationType + source + "," + dest + "," + seq + "," + threshold + "," + 
threshold + "\n";
    }
}
if (rs!=null)
    rs.close();
if (query!=null);
query.close();
conn.release();
writeTextFile("./PropagationType.csv", propagationType);
)} catch (Exception e) {
    //Report("Error adding message");
}

conn = new DTNJDBCConnector();
query = conn.createConnection("SELECT * FROM MESSAGE_BUFFER_RPT");
String queueLength = "NODEID, MESSAGE_COUNT,THRESHOLD\n";
try{
    rs = query.executeQuery();
    while (rs.next()){
        source = rs.getString("NODEID");
        seq = rs.getInt("MESSAGE_COUNT");
        threshold = rs.getDouble("THRESHOLD");
        //System.out.println(source + "," + seq + "," + threshold);
        queueLength = queueLength + source + "," + seq + "," + threshold + "\n";
    }
    if (rs!=null)
        rs.close();
    if (query!=null);
    query.close();
    conn.release();
    writeTextFile("./QueueLength.csv", queueLength);
} catch (Exception e) {
    //Report("Error adding message");
}

public static boolean writeTextFile(String filename, String message){
    try{
        File myFile = new File(filename);
        Writer output = new BufferedWriter(new FileWriter(myFile));
        output.write(message);
        output.close();
        return true;
    } catch (Exception e){
        System.out.println("Could not write to file "+ filename);
        return false;
    }
}

//The main program class. This class should be used to launch threads and provide any interface elements
public static void main(String[] args) {
    if (args.length != 2){
        System.out.println("Usage: DTNNode <Threshold 0.0 - 1.0> <runtime(milliseconds)>");
        System.exit(0);
    }
    double threshold = 0.0;
try{
    threshold = Double.parseDouble(args[0]);
} catch (Exception e){
    System.out.println("Usage: DTNNode <Threshold 0.0 - 1.0> <runtime(milliseconds)>");
    System.exit(0);
}

if (threshold < 0.0){
    threshold = 0.0;
}

if (threshold > 1.0){
    threshold = 1.0;
}

int runlimit = 1200000;

try{
    runlimit = Integer.parseInt(args[1]);
} catch (Exception e){
    System.out.println("Usage: DTNNode <Threshold 0.0 - 1.0> <runtime(milliseconds)>");
    System.exit(0);
}

DTNNode node0 = new DTNNode("127.0.0.1:12340",12340, threshold);
DTNNode node1 = new DTNNode("127.0.0.1:12341",12341, threshold);
DTNNode node2 = new DTNNode("127.0.0.1:12342",12342, threshold);
DTNNode node3 = new DTNNode("127.0.0.1:12343",12343, threshold);
DTNNode node4 = new DTNNode("127.0.0.1:12344",12344, threshold);
DTNNode node5 = new DTNNode("127.0.0.1:12345",12345, threshold);
DTNNode node6 = new DTNNode("127.0.0.1:12346",12346, threshold);
DTNNode node7 = new DTNNode("127.0.0.1:12347",12347, threshold);
DTNNode node8 = new DTNNode("127.0.0.1:12348",12348, threshold);
DTNNode node9 = new DTNNode("127.0.0.1:12349",12349, threshold);
DTNNode node10 = new DTNNode("127.0.0.1:12350",12350, threshold);
DTNNode node11 = new DTNNode("127.0.0.1:12351",12351, threshold);
DTNNode node12 = new DTNNode("127.0.0.1:12352",12352, threshold);
DTNNode node13 = new DTNNode("127.0.0.1:12353",12353, threshold);
DTNNode node14 = new DTNNode("127.0.0.1:12354",12354, threshold);
DTNNode node15 = new DTNNode("127.0.0.1:12355",12355, threshold);
DTNNode node16 = new DTNNode("127.0.0.1:12356",12356, threshold);
DTNNode node17 = new DTNNode("127.0.0.1:12357",12357, threshold);
DTNNode node18 = new DTNNode("127.0.0.1:12358",12358, threshold);
DTNNode node19 = new DTNNode("127.0.0.1:12359",12359, threshold);

try{
    Thread.sleep(runlimit);
} catch (Exception e){

}

node0.end();
node1.end();
node2.end();
node3.end();
node4.end();
node5.end();
node6.end();
node7.end();
node8.end();
import java.util.ArrayList;
import java.util.TreeSet;
import java.util.TreeMap;

public class MessageStatistics {
    final Double gamma = .98;
    final Double Pinit = .75;
    final Double beta = .25;
    public TreeMap<String,Double> prophetProbabilities;
    private TreeMap<String,MessageNode> messages;

    public MessageStatistics(){
        prophetProbabilities = new TreeMap<String,Double>();
        messages = new TreeMap<String,MessageNode>();
    }

    public ArrayList<String> deliverMessages(String destination){
        synchronized (messages){
            ArrayList<String> deliveredMessages = new ArrayList<String>();
            ArrayList<String> allMessages = new ArrayList<String>(messages.keySet());
            for (int i = 0; i < allMessages.size(); i++){
                //System.out.println(messages.get(allMessages.get(i)).getMessageDestination() + "," + destination);
                if (messages.get(allMessages.get(i)).getMessageDestination().equalsIgnoreCase(destination)){
                    deliveredMessages.add(messages.get(allMessages.get(i)).toString());
                    RemoveMessage(allMessages.get(i));
                }
            }
            return deliveredMessages;
        }
    }
}

//END CLASS MessageStatistics

//BEGIN CLASS MessageStatistics

//This class is designed to hold messages and their associated Prophet statistics
//The class provides basic add/remove functionality for messages and Prophet statistics
//and the ability to return values for both in a String format that is designed to be useful
//for transmission to other nodes.
public class MessageStatistics {  
    final Double gamma = .98;
    final Double Pinit = .75;
    final Double beta = .25;
    public TreeMap<String,Double> prophetProbabilities;
    private TreeMap<String,MessageNode> messages;

    public MessageStatistics(){
        prophetProbabilities = new TreeMap<String,Double>();
        messages = new TreeMap<String,MessageNode>();
    }

    public ArrayList<String> deliverMessages(String destination){
        synchronized (messages){
            ArrayList<String> deliveredMessages = new ArrayList<String>();
            ArrayList<String> allMessages = new ArrayList<String>(messages.keySet());
            for (int i = 0; i < allMessages.size(); i++){
                //System.out.println(messages.get(allMessages.get(i)).getMessageDestination() + "," + destination);
                if (messages.get(allMessages.get(i)).getMessageDestination().equalsIgnoreCase(destination)){
                    deliveredMessages.add(messages.get(allMessages.get(i)).toString());
                    RemoveMessage(allMessages.get(i));
                }
            }
            return deliveredMessages;
        }
    }
}

//END CLASS MessageStatistics
public int messageCount()
    synchronized (messages)
    {
        return messages.size();
    }
}

public ArrayList<String> removeDeadMessages()
    synchronized (messages)
    {
        ArrayList<String> deadMessages = new ArrayList<String>();
        ArrayList<String> allMessages = new ArrayList<String>(messages.keySet());
        for (int i = 0; i < allMessages.size(); i++)
        {
            if (messages.get(allMessages.get(i)).getMessageTTL() <= 0)
                deadMessages.add(messages.get(allMessages.get(i)).toString());
            RemoveMessage(allMessages.get(i));
        }
        return deadMessages;
    }
}

public Double updateProphet(String key)
    synchronized (prophetProbabilities)
    {
        Double PabOld = getProphet(key);
        Double newvalue = PabOld + (1 - PabOld) * Pinit;
        return setProphet(key, newvalue);
    }
}

public Double transitiveProphetUpdate(String c, String b, Double Pbc)
    synchronized (prophetProbabilities)
    {
        Double Pab = getProphet(b);
        Double PacOld = getProphet(c);
        Double PacNew = PacOld + (1 - PacOld) * Pab * Pbc * beta;
        return setProphet(c, PacNew);
    }
}

public void ageAllProphet()
    synchronized (prophetProbabilities)
    {
        String[] keys = prophetProbabilities.keySet().toArray(new String[0]);
        for (int i = 0; i < keys.length; i++)
        {
            ageProphet(keys[i]);
        }
    }
}

public Double ageProphet(String key)
    synchronized (prophetProbabilities)
    {
        Double PabOld = getProphet(key);
        Double agedvalue = PabOld * gamma;
        return setProphet(key, agedvalue);
    }
}

public Double setProphet(String key, Double value)
    synchronized (prophetProbabilities)
    {
        if (value < 0.0) {
            return prophetProbabilities.put(key.toUpperCase(), 0.0);
        }
} else {
    prophetProbabilities.put(key.toUpperCase(), value);
}
}

public Double getProphet(String key){
    synchronized (prophetProbabilities){
        if (prophetProbabilities.containsKey(key.toUpperCase())){
            return prophetProbabilities.get(key.toUpperCase());
        } else {
            return 0.0;
        }
    }
}

public ArrayList<String> getProphetKeys(){
    synchronized (prophetProbabilities){
        ArrayList<String> keys = new ArrayList<String>(prophetProbabilities.keySet());
        return keys;
    }
}

public String AllProphets(){
    synchronized (prophetProbabilities){
        String[] keys = prophetProbabilities.keySet().toArray(new String[0]);
        Double[] values = prophetProbabilities.values().toArray(new Double[0]);
        String pairs = "";
        if (keys.length == values.length){
            for (int i = 0; i<keys.length; i++){
                pairs = pairs + keys[i] + " " + values[i] +"n";
            }
        }
        return pairs;
    }
}

public Double averageProphet(){
    synchronized (prophetProbabilities){
        ArrayList<Double> values = new ArrayList<Double>(prophetProbabilities.values());
        ArrayList<String> pKeys = new ArrayList<String>(prophetProbabilities.keySet());
        ArrayList<MessageNode> mKeys = new ArrayList<MessageNode>(messages.values());
        int unknownHosts = 0;
        ArrayList<String> countedDestinations = new ArrayList<String>();
        for (int i = 0; i < mKeys.size(); i++){
            if ((!pKeys.contains(mKeys.get(i).getMessageDestination())) &&
                (!countedDestinations.contains(mKeys.get(i).getMessageDestination()))){
                unknownHosts++;
                countedDestinations.add(mKeys.get(i).getMessageDestination());
            }
        }
        if (values.size() == 0){
            return 0.0;
        }
        Double average = 0.0;
for (int i = 0; i < values.size(); i++)
    average = average + values.get(i);
}
average = average/(double)(values.size() + unknownHosts);
return average;
}

public void DecrementAllTTL()
{
    synchronized (messages)
    {
        ArrayList<MessageNode> allMessages = new ArrayList<MessageNode>(messages.values());
        for (int i = 0; i < allMessages.size(); i++)
            allMessages.get(i).decrementTTL();
    }
}

public boolean AddMessage(String originator, String destination, int ID, int TTL)
{
    synchronized (messages)
    {
        MessageNode temp = new MessageNode(originator, destination, ID, TTL);
        return (messages.put(temp.getDistinctID(), temp) == null);
    }
}

public boolean RemoveMessage(String ID)
{
    synchronized (messages)
    {
        return (messages.remove(ID) == null);
    }
}

public String AllMessages()
{
    synchronized (messages)
    {
        ArrayList<MessageNode> allMessages = new ArrayList<MessageNode>(messages.values());
        String allMessageString = "";
        for (int i = 0; i < allMessages.size(); i++)
            allMessageString = allMessageString + allMessages.get(i).toString() + "\n";
        return allMessageString;
    }
}

public String AllMessageIDs()
{
    synchronized (messages)
    {
        ArrayList<MessageNode> allMessages = new ArrayList<MessageNode>(messages.values());
        String messageIDs = "";
        for (int i = 0; i < allMessages.size(); i++)
            //System.out.println("ID:" + allMessages.get(i).getDistinctID());
            messageIDs = messageIDs + allMessages.get(i).getDistinctID() + "\n";
        return messageIDs;
    }
}

public String ActiveMessageIDs()
{
    synchronized (messages)
    {
        ArrayList<MessageNode> allMessages = new ArrayList<MessageNode>(messages.values());
        String messageIDs = "";
        for (int i = 0; i < allMessages.size(); i++)
            //System.out.println("ID:" + allMessages.get(i).getDistinctID());
            messageIDs = messageIDs + allMessages.get(i).getDistinctID() + "\n";
        return messageIDs;
    }
}
if (allMessages.get(i).getMessageTTL()>0) {
    messageIDs = messageIDs + allMessages.get(i).getDistinctID() + "n";
}

} return messageIDs;
}

public String getMessage(String messageID) {
    synchronized (messages) {
        if (messages.get(messageID) == null) {
            return null;
        }
        return messages.get(messageID).toString() + "n";
    }
}

public int getMessageTTL(String messageID) {
    synchronized (messages) {
        if (messages.get(messageID) == null) {
            return 0;
        }
        return messages.get(messageID).getMessageTTL();
    }
}

public String getMessageDestination(String messageID) {
    synchronized (messages) {
        if (messages.get(messageID) == null) {
            return null;
        }
        return messages.get(messageID).getMessageDestination();
    }
}

// A class to hold a message in a way that can be used in a TreeSet
// along with some standard "getters and setters," a compareTo function,
// and a toString function. Also includes a function that decrements the TTL
// for the message.
private class MessageNode implements Comparable<MessageNode> {
    private String messageOriginator;
    private String messageDestination;
    private int messageID;
    private int messageTTL;

    public MessageNode(String o, String d, int i, int t) {
        messageOriginator = o;
        messageDestination = d;
        messageID = i;
        messageTTL = t;
    }

    public void decrementTTL() {
        messageTTL--;
        if (messageTTL < 0) {
            messageTTL = 0;
        }
    }
}
@Override
public int compareTo(MessageNode m) {
    String myID = this.getDistinctID();
    String targetID = m.getDistinctID();
    return myID.compareToIgnoreCase(targetID);
}

public String getMessageOriginator() {
    return messageOriginator;
}

public void setMessageOriginator(String messageOriginator) {
    this.messageOriginator = messageOriginator;
}

public String getMessageDestination() {
    return messageDestination;
}

public void setMessageDestination(String messageDestination) {
    this.messageDestination = messageDestination;
}

public int getMessageID() {
    return messageID;
}

public void setMessageID(int messageID) {
    this.messageID = messageID;
}

public int getMessageTTL() {
    return messageTTL;
}

public void setMessageTTL(int messageTTL) {
    this.messageTTL = messageTTL;
}

public String getDistinctID() {
    return messageOriginator + " " + messageID + " " + messageDestination;
}

public String toString() {
    return messageOriginator + ";" + messageDestination + ";" + messageID + ";" + messageTTL;
}

} //END CLASS MessageStatistics