|  |
| --- |
| The University of Alabama |
| Delay Tolerant Network (DTN) Protocol Implementation |
| Fall 2011 CS613 Course Project |

|  |
| --- |
| Hüseyin Ergin  12/11/2011 |

Contents

[1 – Introduction 2](#_Toc311474266)

[Server Oracle 2](#_Toc311474267)

[Routing Algorithms 2](#_Toc311474268)

[First Contact 2](#_Toc311474269)

[Epidemic 3](#_Toc311474270)

[2 – Implementation Details 3](#_Toc311474271)

[Data Structures 3](#_Toc311474272)

[Server 3](#_Toc311474273)

[API 3](#_Toc311474274)

[Admin Interface of Server 5](#_Toc311474275)

[Shape Legend in Views 9](#_Toc311474276)

[Client (Nodes) 9](#_Toc311474277)

[Node Communication 9](#_Toc311474278)

[Routing 10](#_Toc311474279)

[Unique Message IDs 10](#_Toc311474280)

[3 – Conclusion 10](#_Toc311474281)

[4 – References 11](#_Toc311474282)

[Appendix A – Scenarios 12](#_Toc311474283)

[Scenario 1 12](#_Toc311474284)

[Scenario 2 12](#_Toc311474285)

[Scenario 3 13](#_Toc311474286)

[Scenario 4 13](#_Toc311474287)

[Scenario 5 14](#_Toc311474288)

[Appendix B – Operation Manual 15](#_Toc311474289)

# 1 – Introduction

In this project, we wanted to see some of the Delay Tolerant Network (DTN) routing protocols at work. DTN is different than traditional networks. Routing in a DTN is currently based on store and forward routing algorithms. Each node in the DTN is both responsible for forwarding the received packet to the next hop and if no other hops are active, it has to store that packet for future forwarding. For implementation purposes we have used a server oracle and implemented three different routing algorithms for DTN.

## Server Oracle

We have used server oracle to simulate the mobility requirement of the DTN. DTN nodes are generally mobile nodes that is rarely connecting to the whole network structure. The bus example[1] is a perfect example with mobility pattern.

In our oracle, each node in DTN is designed as a client in a client-server architecture. Each node connects to the server at beginning and stay connected during the lifetime of the DTN. Server is responsible for managing link information, adding new links to DTN and removing links from it.

We also added some more functionalities to the server to track the DTN more effectively. Other functionalities are:

* Viewing the current link structure of the DTN visually.
* Activate/deactivate/delete link information.
* Logging messages sent and received by nodes
* Showing the statuses of each message and the history path visually.
* Modifying the configurations like TTL and the routing algorithm.
* Generating a random network by adding some links automatically.

## Routing Algorithms

We implemented two routing algorithms. One is First Contact[2] (FC). We designed two variations of FC. First one is with history and the other one is without history. The second algorithm we implemented is Epidemic[3].

### First Contact

A contact is a chance to send data to another node along the time they are connected. First contact (FC) algorithm is based on sending the received/buffered message to the first node that is connected. For the implementation purposes, we have created two variants of FC.

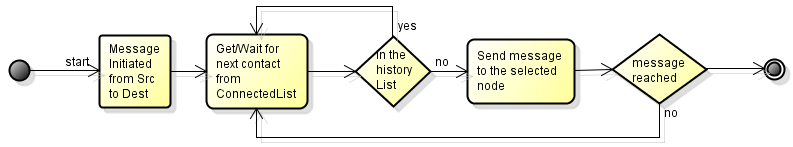
1. First Contact with History: When a message is sent from source to destination, it will follow a path according to the first contact. Each message consists of a text and a history of the nodes it already passed. This algorithm routes the message to the next hop if the next hop is not already in the history list of the message. The flowchart of the algorithm is in Figure 1.  
     
   

Figure 1: Flowchart of First Contact with History

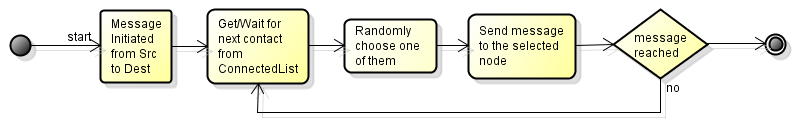
1. First Contact without History: The only difference of this algorithm with the first one is not taking the history of the message into account. It randomly chooses the next hop among the connected nodes and forwards the message to it. The flowchart is in Figure 2.  
     
   

Figure 2: Flowchart of First Contact without History

### Epidemic

Epidemic algorithm is like a brute-force mechanism. It forwards the message to all of the nodes that it is connected. Since all nodes will have the copy of the message, the success of message delivery is guaranteed if there will eventually be a connection to the destination. The flowchart is in Figure 3.

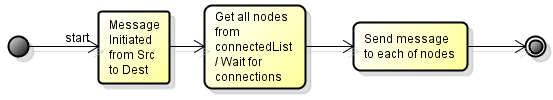
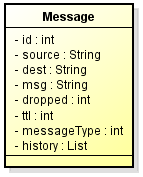
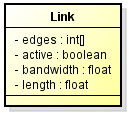
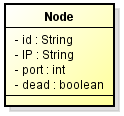


Figure 3: Flowchart of Epidemic

# 2 – Implementation Details

## Data Structures

The following data structures are used in our implementation.  


## Server

Server is a web server on Tomcat application[7] written in Java programming language. It has the capacity to store the list of all nodes, list of all links and list of all messages. Server also has the configurations of TTL and current algorithm type of the system.

### API

The clients are communicating with the server via a servlet. The servlet has the following interfaces available to the clients.

|  |  |  |
| --- | --- | --- |
| **ATTACH** | | |
| **URL** | <server IP>/dtn.serv | |
| **Parameters** | *type* | attach |
|  | *nodeId* |  |
| *IP* |
| *port* |
| **Purpose** | Introducing the node to the server | |
| **When** | Called when the client is connected to the server at the beginning of the client | |
| **Execution** | The node is added to the node list in the server. | |
| **Output** | None | |
| **Consequences** | All other connected nodes are notified about this new node. | |
|  |  |  |
| **DETACH** | | |
| **URL** | <server IP>/dtn.serv | |
| **Parameters** | *type* | detach |
|  | *nodeId* |  |
| **Purpose** | Removing the node to the server | |
| **When** | Called when the client is closed | |
| **Execution** | The node is flagged as dead. | |
| **Output** | None | |
| **Consequences** | \*All other connected nodes are notified about this new detaching. \*The links of this node is set as inactive from that time (and can't be activated after that point). \*Any messages in messageList that has that node as destination will be flagged as DEST\_DETACHED. | |
|  |  |  |
| **GETNODELIST** | | |
| **URL** | <server IP>/dtn.serv | |
| **Parameters** | *type* | getNodeList |
| **Purpose** | Getting the list of all nodes in DTN | |
| **When** | Called by each node when they started. | |
| **Execution** | All nodes in the nodeList is traversed and the id's are concantenated by a pipeline (|) sign if they aren't dead. | |
| **Output** | *id1|id2|id3|…..|idn* | |
| **Consequences** | None | |
|  |  |  |
| **GETCONNECTEDNODELIST** | | |
| **URL** | <server IP>/dtn.serv | |
| **Parameters** | *type* | getConnectedNodeList |
|  | *nodeId* |  |
| **Purpose** | Getting the list of all nodes that is connected to the node in the parameter | |
| **When** | Called by each node while they are sending/forwarding a message | |
| **Execution** | All nodes in the nodeList is traversed and the id's that are connected to that node are concantenated by a pipeline (|) sign if they aren't dead. | |
| **Output** | *id1|id2|id3|…..|idx* *ERROR* if no connected node is found | |
| **Consequences** | None | |
|  |  |  |
| **GETNODEINFO** | | |
| **URL** | <server IP>/dtn.serv | |
| **Parameters** | *type* | getNodeInfo |
|  | *nodeId* |  |
| **Purpose** | Get the IP and port information of a single node | |
| **When** | Called by each node to send a message to the node in the parameter | |
| **Execution** | The node is checked from nodeList and IP and port information is concantenated by a colon (:) sign. | |
| **Output** | *IPx:portx* | |
| **Consequences** | None | |
|  |  |  |
| **LOG** | | |
| **URL** | <server IP>/dtn.serv | |
| **Parameters** | *type* | log |
|  | *msg* |  |
| **Purpose** | Send a message structure to server to log | |
| **When** | Called by each node while sending/forwarding a message or saving the message to the buffer | |
| **Execution** | The message is added to messageList if it doesn't already exist. If it exists, the historyList of the incoming message is appended to the existing one | |
| **Output** | None | |
| **Consequences** | None | |
|  |  |  |
| **GETPROTOCOL** | | |
| **URL** | <server IP>/dtn.serv | |
| **Parameters** | *type* | getProtocol |
| **Purpose** | Get the protocol information from server | |
| **When** | Called by each node when they started | |
| **Execution** | Returns the existing algorithm information | |
| **Output** | *1* if algorithm is First Contact (with History) *2* if algorithm is First Contact (without History) *3* if algorithm is Epidemic | |
| **Consequences** | None | |
|  |  |  |
| **GETTTL** | | |
| **URL** | <server IP>/dtn.serv | |
| **Parameters** | *type* | getTTL |
| **Purpose** | Get the TTL information from server | |
| **When** | Called by each node when they started | |
| **Execution** | Returns the existing TTL information | |
| **Output** | *ttl* | |
| **Consequences** | None | |

### Admin Interface of Server

Server has an admin panel which is accessible via web browser. Struts technology[6] is used in the admin panel. Struts lets developers easily maintain active web pages. Behind web pages, there is Java technology and in front Jsp is used to view web pages.

User can add/remove links from DTN. A screenshot of this page is in Figure 4. A self link can’t be added.

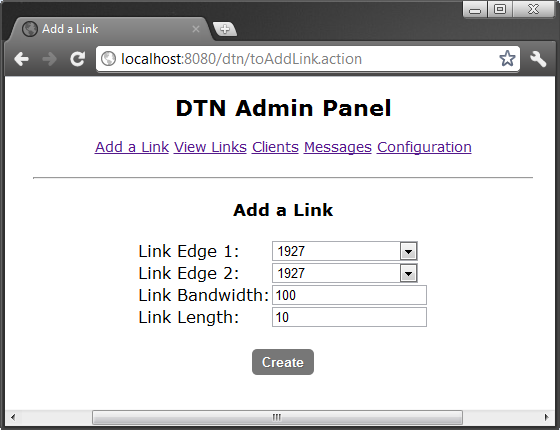


Figure 4: Admin Panel > Add Link Page

The current link information of DTN can be viewed from the panel. While drawing the current link structure of the network, we used DOT[4] graph language. DOT is a really simple graph language that is used to represent graphs as text. Figure 5 shows a simple graph written in DOT language with 4 nodes (A,B,C,D) and 5 links between these nodes (A-B, A-C, A-D, B-C, D-C).

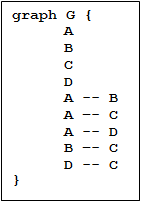


Figure 5: Sample DOT Graph

After the link structure is represented as DOT language, it is compiled by Graphviz[5] program and the picture is generated.

The link viewing page in admin panel consists of an image of links and also a list of all links and operation information. The current operations are activate/deactivate and deleting links. A screenshot of this page is in Figure 6.

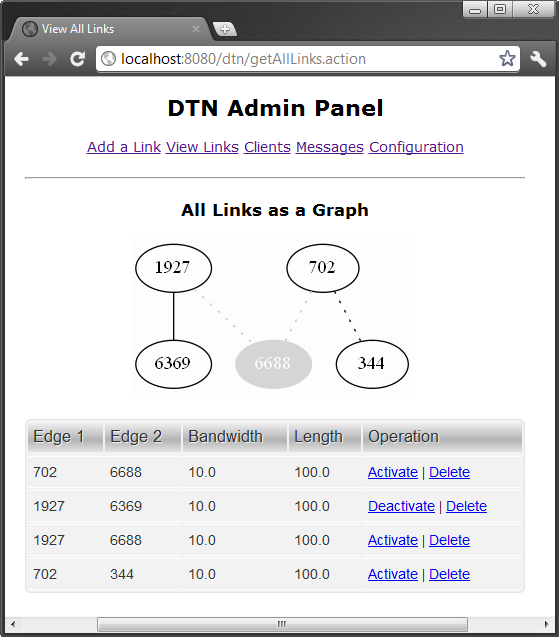


Figure 6: Admin Panel > View Links Page

User can also view the current links in the DTN. This page is shown in Figure 7 and has delete operation for the clients. When client is deleted from the system, it is not removed totally. It is lazy-deleted and flagged as deleted. We are using lazy-deletion method, because we use this feature while showing the message details (path history).

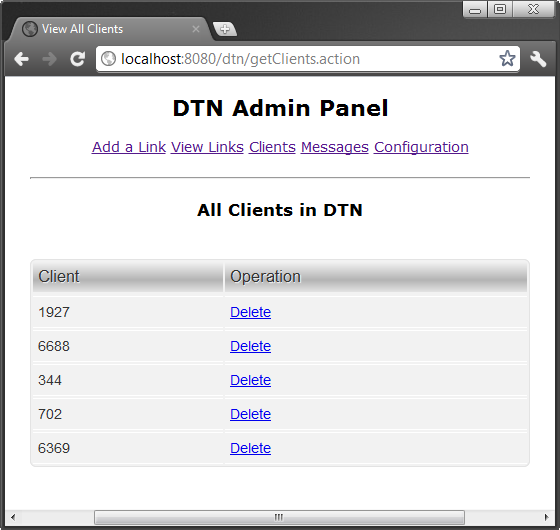


Figure 7: Admin Panel > View Clients Page

The configuration page is used for modifying the current variables in DTN. We have two variables for now. First one is the algorithm type which is mentioned earlier. Second one is TimeToLive (TTL) value for messages in DTN. TTL is used for preventing the messages to be forwarded forever in the network. After TTL amount of forwarding, the message will not be forwarded any more.

Configuration page has also a link for random generation of networks. When this link is clicked, the server randomly adds a specific number of links to the system. A screnshot of the configuration page is in Figure 8.

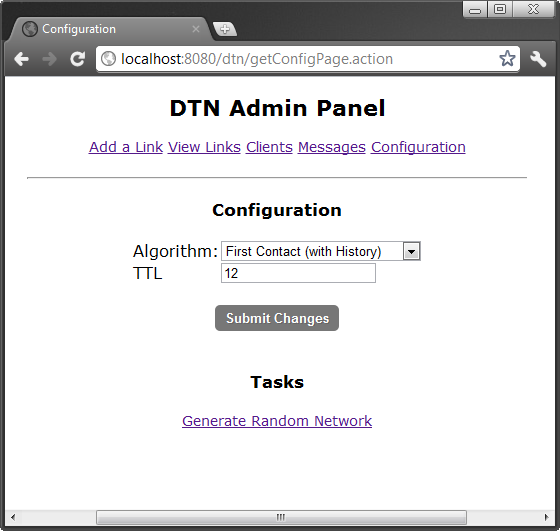


Figure 8: Admin Panel > Configuration Page

When a variable is modified in the configuration page, this change will automatically be broadcasted to all connected nodes in DTN.

Server has also a messages page, where we can see all the messages and their statuses in the network. A message may be in one of the following status:

* Died of TTL: The message is died because TTL is not enough for the message to reach to destination.
* On the way: The message is buffered in one (or more) of the nodes and will be forwarded again if any other links is available.
* Reached: The message succesfully reached to destination.

A screenshot from the message history page is in Figure 9.

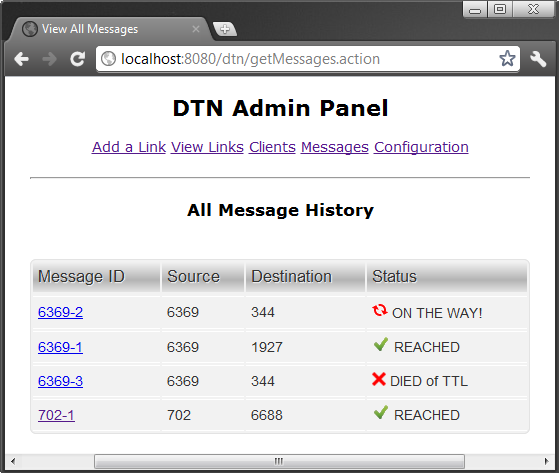


Figure 9: Admin Panel > View Messages Page

Each message can be tracked from this page. We can also show the details of the message like which nodes it passed while going to destination, what is the source and destination and the current node. In epidemic algorithm, the message is buffered in more than one nodes but DTN will again show one node has it as current. The message details page is in Figure 10.

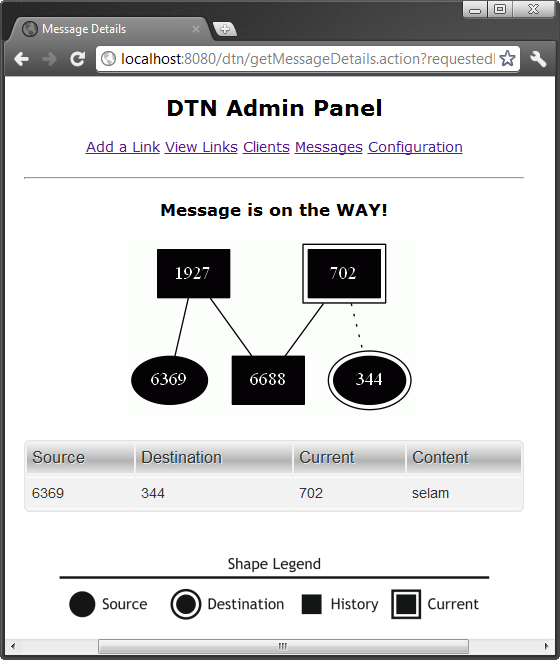


Figure 10: Admin Panel > Message Details Page

### Shape Legend in Views

In both link viewing and message detail viewing, the following legend is used.

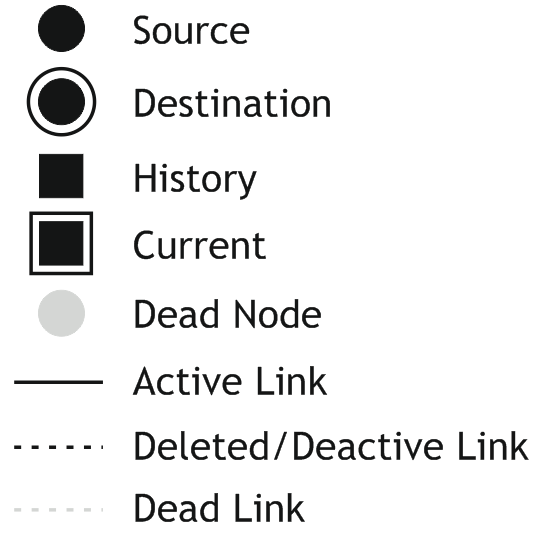


Figure 11: Legend

## Client (Nodes)

The nodes are simple Java programs with a graphical user interface (GUI). At the beginning of the nodes, the node is asking for the server connection information. The node will try to connect and attach itself to the network. Then, it will receive all node list from the server. After that, a GUI like in Figure 12 will be shown to the user.

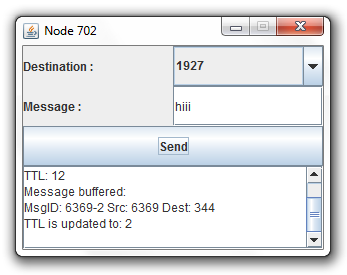


Figure 12: The GUI for Client

GUI has a destination field, which is updated automatically by the server when a new node is connected to the network, a message field and a log area. Log area is used for logging all events that happened in the node. The events are:

* Node attach.
* TTL and algorithms updates.
* Message send and buffer.
* Received message.

Each client has a message buffer. This buffer is used for storing purposes. If no links are available to forward the message, the message is stored here. Each client is also storing a variable for current algorithm and current TTL. At start, client makes api calls and get these variables from server.

### Node Communication

The nodes are communicating with eachother via UDP connection. Each node is opening a UDP socket and listening that socket for incoming messages as a seperate thread. This thread accepts messages from other nodes and also from server. The accepted message types are:

* Normal Message: A message from another node.
* Notify About New Link: A message received from server that a new link is created. After this message is received, message buffer is checked if any of the messages can be forwarded through that link.
* Node Joined: A message received from server that a new node is joined to the network. After this message, node makes an api call to getNodeList and updates the destination field.
* Algorithm Change: A message received from server that the algorithm is changed from configuration page of admin panel. The node makes an api call to getProtocol and updates the current algorithm.
* TTL Change: A message received from server that the TTL variable is changed from configuration page of admin panel. The node makes an api call to getTTL and updates the current TTL.

### Routing

The routing is done in the route function of MessageSender class of Client codes. The epidemic algorithm routing code is shown below in Figure 13.

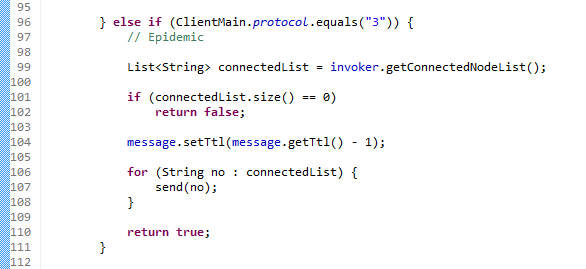


Figure 13: Code Fragment for Epidemic

In the 99th line, we are receiving the list of nodes that is connected to current node. If there are no connected nodes, we are returning false. That will put the current message to the buffer. If there are connected nodes, then we will decrease TTL of the message by one and we are sending the message to each of the connected nodes.

### Unique Message IDs

Each message has a unique ID. An internal counter is maintained by the nodes and this counter is incremented by one at each message sending. Unique ID is generated by appending the node ID and the value of the internal counter together.

# 3 – Conclusion

DTN routing is always a challenging issue since the partial connected structure of the network. In this project, we have implemented and run some scenarios on the DTN. The content of these scenarios and consequences are added in Appendix A. Only some edge scenarios are selected.

One can implement more routing algorithms by adding the implementation to the route function described in Figure 13. Also some more additions must be done in the admin panel configuration page (Figure 8).

Also the operation manual is added in Appendix B for future usage purposes of the project.

# 4 – References

[1]: Utku G Acer; Paolo Giaccone ; David Hay; Giovanni Neglia; Timely Data Delivery in a Realistic Bus Network, Saed Tarapiah

[2]: S. Jain, K. Fall, R. Patra, Routing in a Delay Tolerant Network, SIGCOMM, Aug/Sep 2004

[3]: Amin Vahdat and David Becker. Epidemic routing for partially connected ad hoc networks. Technical Report CS-2000-06, Department of Computer Science, Duke University, April 2000

[4]: http://www.graphviz.org/doc/info/lang.html

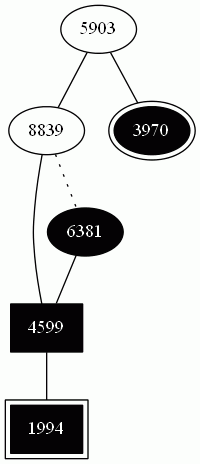
[5]: http://www.graphviz.org

[6]: http://struts.apache.org/2.x/

[7]: http://tomcat.apache.org

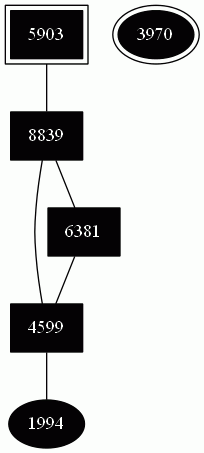
# Appendix A – Scenarios

## Scenario 1



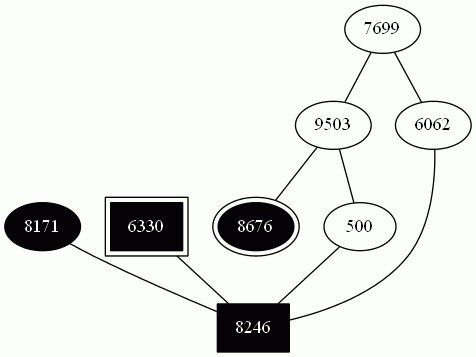
Algorithm: First Contact with History  
TTL: 12  
Source: 6381  
Destination: 3970  
Consequences: The message started its journey from source node which is 6381. It chose the only available node (4599). After that, 4599 forwarded the message to 1994. Now the message stuck in 1994 until another new link will be available. It can’t go back, because 4599 is already in the history list.

## Scenario 2



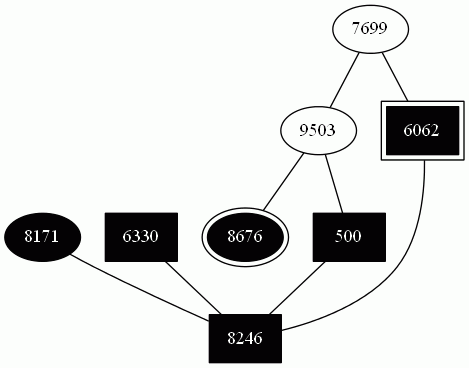
Algorithm: Epidemic  
TTL: 12  
Source: 1994  
Destination: 3970  
Consequences: The message started its journey from source node which is 1994. Since the algorithm is epidemic, now the message is broadcasted to all connected nodes in all stations. But it is still stuck in 5903. Because there is no connection to the destination. In this case, the message is buffered in all of the nodes, until a link is created between one of them and the destination (3970).

## Scenario 3



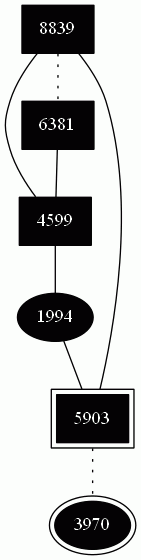
Algorithm: First Contact without History  
TTL: 3  
Source: 8171  
Destination: 8676  
Consequences: The message started its journey from source node which is 8171. It is forwarded and not it is on 6330. But 6330 will not forward the message once again, since TTL of the message is reached to 0 and the message died in the network. The message traversed 3 nodes (including source)

## Scenario 4



Algorithm: First Contact without History  
TTL: 7  
Source: 8171  
Destination: 8676  
Consequences: The message started its journey from source node which is 8171. This case is the same as scenario 3 but this time TTL is 7. But the message tracked a path that it used all TTL until reached to destination and died. The path of the message is 8171🡪8246🡪6330🡪8246🡪500🡪8246🡪6062.

## Scenario 5



Algorithm: First Contact without History  
TTL: 20  
Source: 1994  
Destination: 3970  
Consequences: The message started its journey from source node which is 1994. First Contact without History algorithm has a flaw. Since it will not keep the history of the previous path, if there is no connection to the destination, the message will bounce between the connected nodes until TTL is finished. That will cause the message die unnecessarily. In the figure above, the message is died even though TTL is very big, but the connection to destination is deactive.

# Appendix B – Operation Manual

In order to run the DTN, we need the following programs installed with correct versions and correct places.

* Apache Tomcat 5.5.34 (core)
  + http://tomcat.apache.org/download-55.cgi
* Graphviz 2.28.0
  + http://www.graphviz.org/pub/graphviz/stable/windows/graphviz-2.28.0.msi
* Java Runtime Environment (Version > 1.5)
* WAR file of DTN Project
  + dtn.war (which is sent with this report)
* Node GUI Application
  + client.jar (which is sent with this report)

The folder structure must be in this order:

+ apache-tomcat-5.5.34  
 +bin  
 + Graphviz2.28  
 …  
 …  
 +webapps  
 +dtn  
 …

We can install Graphviz to the folder under apache-tomcat-5.5.34/bin. For satisfying dtn folder, we will use the admin panel of Tomcat. After extracting Tomcat to disk, we can change port of Tomcat if we want (Google will help). Then we have to execute bin/startup.bat to start Tomcat.

The startup must be quick. After that, we can go to a browser and open http://localhost:8080 for the admin panel of Tomcat. From the admin panel, we can deploy our dtn application (Google will help while deploying).

After deploy, now our program is ready to run. Just point http://localhost:8080/dtn from a web browser and we can see the admin panel.

The server is up and running now. We can run some clients, point the server’s IP and port information and if they are correct, client will join to the DTN.