IMPLEMENTATION OF DYNAMIC SOURCE ROUTING

Project report submitted in partial fulfillment
of the requirements for the degree of

MASTER OF TECHNOLOGY

in

Computer Science & Engineering

by

PRAVEEN KUMAR
Entry No. 2014MCS2130

SURESH P
Entry No. 2014MCS2144

TANMAY GUPTA
Entry No. 2014MCS2145

Under the guidance of
Dr. VINAY RIBEIRO

Department of Computer Science and Engineering,
Indian Institute of Technology Delhi.
May 2015.

Acknowledgments

First and foremost, we thank to the Power of Almighty for showing us inner peace and for all blessings. Special gratitude to our Parents, for showing their support and love always. We like to acknowledge the constant support provided by the Course Coordinator Dr. Vinay Ribeiro for his consistent motivation in pursuing our project.

We express our sincere thanks to our seniors for providing us their valuable guidance, untiring patience and diligent encouragement during the entire span of this project. We also express our thanks to our friends who helped in testing the project.

PRAVEEN KUMAR (2014MCS2130)
SURESH P (2014MCS2144)
TANMAY GUPTA (2014MCS2145)

Abstract

Mobile Ad Hoc Network is infrastructure less network which manages the network independently. Nodes are always moving in MANET and there is no fixed infrastructure for MANET. We are going to implement Dynamic Source Routing algorithm on MANET. Dynamic source Routing Protocol is efficient routing protocol for multi hop wireless ad hoc network of mobile nodes. Dynamic Source Routing allows the network to be completely self-organizing and self-configuring without any need of existing network.

Dynamic Source Routing uses route discovery cycle for route finding on demand and route maintenance, they both work together for shortest path and also maintain source route to some arbitrary destination in the Mobile ad hoc network. We will use Laptops as mobile nodes in MANET. We will implement DSR in C++ and basic testing will be carried out with 3-4 mobile nodes each of which will run DSR code.

Contents

List of Figures

Chapter 1

Introduction

Dynamic Source Routing (DSR) is a routing protocol used to send data in wireless network designed specially for use in multi-hop ad hoc networks consisting of mobile nodes which is in our case Laptops. DSR forms on demand routes (if previously does not know the route). No prior configuration and organization of network is required. Since Mobile Ad Network is infrastructure less network which manages the network independently each node cooperate with each other i.e. with those who are in the range. Also in MANET nodes move in the network and also join and leave the network for all these condition DSR is best suited. In DSR routes are found on demand so it can work on zero configuration. For sending data from one node to another node in this environment DSR first uses route discovery cycle for finding route from source to destination in which each node will communicate with other node in the range and ultimately find the path. Once the path is found Route Reply is send to the source on the path found.

Dynamic Source algorithm work on demand and does not send any periodic message of any kind. If a source wants to send the data to another node and it does not have route information then Route Discovery will be used by source in which it will just flood the RREQ message to the neighbour with the destination information and special ID. The nodes in the hop will in turn again broadcast the RREQ message to neighbour. When the destination receives the message intended for it copies the addresses of all nodes that packet contains including the source to RREP packet. Then destination node unicasts the RREP packet to C in source route. This packet reaches the source and source gets the path along which it then sends the actual path.

Next Important part of Dynamic Source Routing is route maintenance. Once the route is discovered by source the path will be kept in the route cache in the source and source can send the data in the path saved i.e. source will unicast the data to the destination along the path. If while sending the data sending node finds that the link is broken it generates a RERR packet and send to the source each node in between update the path in their respective cache that the link is broken and source if it has an alternative path will send the data along that path otherwise source can invoke the route discovery again. Both route maintenance and route discovery operate on demand only .

Chapter 2

Algorithm Study

In the current phase of the project we started with the study of the algorithm inorder to get familiarize with it. Various aspects of the algorithm have been studied. During the study of the algorithm our main focus was to understand how the algorithm can be used to implement DSR in MANET as part of the project and also decisions were made on which part of the algorithm to implement and which of them not to.

Route Discovery

Let a source S wants to send data to Destination D. Source will broadcast a RREQ packet which will contain a unique RREQ packet id, destination address, and its own address. When a node receives packet it will first compare the packet destination address to its own address and

1. If node is not destination node
    - It will check whether it has seen the packet already if it has it will discard that packet. This will be done by comparing the incoming packets unique RREQ packet id. Otherwise
    - It will append its address in the packet and broadcast it again.
2. If node is destination node
    - Node will make a new RREP packet
    - It will copy the route record and RREQ ID from the RREQ into RREP packet and also it will append its address in the path field and then node will unicast the packet along the path found.
    - Upon receiving the RREP packet the intermediate node in the path will check the RREP source route and will unicast the packet

to the next node in the source route. Also the node will append its own address in the RREP path field.

- When the source receives the RREP packet it will copy the addresses in the path field to its address cache and will send the data in the path found.

Route Caching

In route discovery when source receives the RREP from the destination it saves the path from the destination to itself in its route cache which it can use immediately and also in future to send data to node which is the path between source and the destination. Also when each node sends the RREQ packet it also finds the path from the source to itself and it stores the path in its cache which it can use when the node needs the same path to send data in future. The same thing can also be done in RREP packet sending.

In case when a node receives the multiple packet for same destination from two different source it can save both of them in its cache. This information about route can be used when one of the path gets broken or damaged. In addition to this route caching can also optimize the route discovery process consider for a scenario where a RREQ request for some destination is received by some intermediate node and that node has the route from itself to the destination instead of sending the RREQ packet by broadcasting the node can append the address from its cache to the RREQ packet and send the RREP packet to the destination.

Route Maintenance

After finding the route source S will send the data to D along the path it has known by route discovery i.e. Source Route. Let Source S have to send the data for Destination D through A, B, C intermediate nodes i.e. source route is S-A-B-C-D.

S will send the data to A and will wait for the acknowledgement from A for the successful delivery of the data. similarly A will send data to B and will receive acknowledgement. Now suppose if B send data to C and no acknowledgement is received from C, B will wait for some predefined amount of time for the acknowledgement to come and if B does not receive the acknowledgement it will send the RERR packet to all the nodes in the path from which it received packet. All the nodes which will receive the packet will update their corresponding route cache for that path and remove the path C-D. If any of the node receiving RERR has an alternate path in its route cache it will try to send the data again in that path if it succeeds the source will update the new path in its cache. Otherwise Source S has to again initiate route discovery.

The route discovery and route maintenance steps involve three types of messages

Route Request (RREQ) Whenever a source node wants to discover route to a destination, it will broadcast RREQ message. This message is then broadcasted by the subsequent nodes until the destination receives this RREQ packet.

Route Reply (RREP) As soon as the destination receives a route request to itself, it originates a route reply (RREP) message and forwards it to the source through the path found in the RREQ packet.

Route Error (RERR) During the packet delivery, if the original path has changed, then the node, that is not able to send the packet, will send a Route Error (RERR) packet to the source (origin) of the packet.

Chapter 3

Modules

Previous Work

In the previous phase after reading and understanding the DSR algorithm and initial implementation, we set Ad hoc network on mobile node(laptop) as it was decided that it was more challenging on Android devices. So we shifted to implementation of DSR on laptop since earlier we faced a problem that a device is not able to connect to two different wireless interfaces at the same time, with a single Network Interface Card (NIC). So we set up Ad hoc network, we found a link for creating an ad hoc wi-fi network on this link, We followed the steps given in the webpage and we were successful in creating ad hoc wi-fi network. Then we repeated the process on 2 other laptops and created the same ad hoc network setup with different IPs as it requires static IP. Then we connected all the 3 laptops with each other on the ad hoc wi-fi network. We then tried to send broadcast messages and we were successful in broadcasting.

Till second phase we were able to connect to multiple devices in the Ad hoc mode and can broadcast messages over the network. So when we broadcast the packets, only those nodes will be able to receive the packet that are in Wi-fi range. Also the IP that we are going to assign will be static. We are using 2 threads one for getting input from the user and the other to respond to the queries from other nodes. According to the request, the query is going to be processed at that node or the query will be broadcasted again.

Interface

We have made an interface which will get input from the user over the terminal. The user can give route request by typing *rreq <IP >* over the prompt. User if he has route information can send data by typing *send <data ><IP >* over prompt, which will send data to the IP mentioned in the query. This interface will run over 1 thread in the user machine and on the other thread queries from the other nodes will be accepted and processed. The node will respond to the queries from other nodes and if the request is not meant for the node receiving the request it will broadcast it or forward it accordingly.

We started implementing route discovery of DSR algorithm. For this we divided it into 3 main modules

1. Network Module
2. Route Discovery Module
3. Routing Table Module

3.1 Network Module

This module deals with sending and receiving packets over the UDP. Code is written for enabling broadcasting of the packets. So packets will be broadcasted over UDP. Broadcast address of Wi-Fi interface is taken through the system calls.

Content in the packets are space delimited. Packet contains a header which contains the information about the type of message i.e. RREQ for route request, RREP for route reply, SEND for sending the data. Header is seperated by the space with the rest of the message. Rest of the packet contains the following four fields separated by a space.

1. Source IP

2. Destination IP
3. Packet ID
4. Route List

Route list will contain the list of IP address of all the nodes along the path through which the packet has travelled. IP addresses in the route list are separated by commas. E.g. MESSAGE SENT TO 10.42.43.3 : 'RREP: 10.42.43.3 10.42.43.2 0 10.42.43.3,10.42.43.2'

3.2 Route Discovery Module

This module is for finding the route from source to destination. This will be initiated from the source which will give the destination IP address in the RREQ command over the prompt. The request from source will be broadcasted to all the nodes in the range of source. Request from the source will have four parameters.

Source IP

This will contain the IP address of the source.

Destination IP

This will contain the IP address of the destination.

Packet ID

This is unique packet ID for a given source node. No other packet (sent by the same node) in the network will be assigned this ID. Here, ID is an integer value.

Route List

Route List will contain the list of IP addresses separated by commas. Every copy of the packet will have different route list. When sending node broadcasts the packet, it appends its IP address to the route list, similarly when another node receives the packet it appends its IP address to the route list. Finally when the destination node receives the packet, it appends its own IP and the packet is sent back based on the route list.

Route Request

Route request received by the intermediate node will be broadcasted to the other node in the network after appending its IP address in the route list. Every node is maintaining a list of pair of *source IP and packet ID*. if a node receives a packet it store the source IP and the packet unique ID to its list, so if it receives a packet with a ID that it has in its list it will discard the packet and will not re-broadcast the packet. So before receiving any packet each node compares the packet id with the packet ids in its list and process only if packet is not received earlier.

Route Reply

Once the destination receives the RREQ packet intended for itself, it sends the same packet after updating its IP in the route list to the node next to it in the route list. Source IP and Destination IP will still be same as the packet that arrived at destination. Now, the packet will be unicasted to the next node in the route list (i.e. predecessor node in the list). Before sending packet each node will update its routing table for the future use of the path information. In a routing table each node will add 2 fields for the nodes that are between source and destination i.e. for each node in the path found it will add node IP and corresponding to that IP it will maintain the IP of the node to which it should send the data inorder to reach the particular node.

If the path found is A-B-C-D-E then node C will maintain information for each node from A to E and corresponding to that it will have the node which it will pick to send data. This routing information is maintained by each node and is updated by the node which are discovered in the path from sender to destination. This routing information also gives neighborhood information i.e. which nodes are directly connected to the node.

The intended packet when it is received by sender, the sender will get the information about the path from the route list and when it wants to send packet it will send the packet to the destination on the discovered path. Sender also updates the routing table.

3.3 Routing Table Module

This module is implemented to store the information of the network in the Routing Table of each node which is found at the time of route discovery. This module is maintained by using map data structure of string , string . Each node will maintain the information of the nodes and the path it should take to reach to the nodes. Initially in all the nodes routing table will be empty, only at the time of route discovery nodes will update their routing table. For example if the path that was found in route discovery was A - B - C - D - E then the routing table in C will have following information.

1. **B B** (for sending to B node that C has to choose is B)
2. **A B** (for sending to A node that C has to choose is B)
3. **D D** (for sending to D node that C has to choose is D)
4. **E D** (for sending to E node that C has to choose is D)

Note that in the above example, the entries B B and D D denote that B and D are directly connected to C.

In the routing table for each node we are not maintaining the complete path information directly; but we are only maintaining the next node it should pick in order to reach to that particular node in the network. The routing information generated at each node will be used in

- Sending data to the destination after the route has been discovered.
- Next route discovery when initiated will use the routing information to find the route, if the next destination for which route has to be discovered is through a node which has the destination path in its routing table it will directly sent to sender the path from its routing table.

Chapter 4

Implementation

DSR algorithm works on source routing i.e. source will tell the list of nodes to go through for sending data to destination. For finding source path, source node initiates route request i.e. it broadcast the RREQ message. After route request route reply is initiated by the destination or some in between node who has the path till the destination, this route reply is send by RREP message. After the source knows about the path it transmits data that is our data transmission phase. If in any phase link gets broken RERR is initiated which is started by the node which last had the data packet and it sends RERR packet to the source. In DSR data transmission phase we are having two acknowledgements one over UDP layer and one MAC acknowledgement.

4.1 Packet Formats

Packets in the DSR like RREQ, RREP, RRER etc. contains fields which are separated by colon(:). Path if it contains multiple nodes then it will be separated by comma. Different packet formats are as follows.

1. **RREQ Packet** RREQ packet is initiated by the node which wants to send the data to destination node whose address it does not have in its routing table RREQ packet

RREQ	**Source IP**	**UID**	**Destination IP**	**Path**

   - **RREQ :** Packet type.
   - **Source IP :** contains IP address of the source node.
   - **UID :** Unique packet ID at source node.
   - **Destination IP :** Contains IP address of the destination node.

- **Path :** List of IP addresses separated by the comma in the order from source to destination.

2. **RREP Packet** RREP packet is sent by the destination node or an intermediate node with the path information for the original source node through which it can send data.

RREP	Source IP	UID	Destination IP	Path

- **RREP :** Packet type.
- **Source IP :** contains IP address of the original destination node.
- **UID :** Unique packet ID at original destination node.
- **Destination IP :** Contains IP address of the original source node.
- **Path :** List of IP addresses separated by the comma in the order from original source to original destination.

3. **Data Packet** This packet is generated by the node after it has got the path from RREP packet or it had path originally in its routing table. This packet contains the message which is intended for the destination.

DATA	Source IP	UID	Destination IP	Message

- **DATA :** Packet type.
- **Source IP :** contains IP address of the original source node.
- **UID :** Unique packet ID at original source node.
- **Destination IP :** Contains IP address of the original destination node.
- **Message :** Message to be sent as a string.

4. **RERR Packet** This packet is initiated by the node when its timer goes off for receiving MACK acknowledgement from the node where it foreword data packet.This can happen if either link is broken or a node has failed. It contains information about path along which the data

has traveled. This is initiated to delete stale entries from the routing table and to find a new path to destination.

RERR	Source IP	UID	Destination IP	path

- **RERR :** Packet type.
- **Source IP :** Contains IP address of the node which initiated the RERR packet.
- **UID :** Unique packet ID at node sending the RERR packet.
- **Destination IP :** Contains IP address of the original source node.
- **Path :** List of IP addresses separated by the comma in the order from original source to node where RERR was initiated.

5. **UACK Packet** Acknowledgement given by the destination node to the original sender node, to indicate the successful delivery of the data packet.

UACK	Original source IP	Original UID	Source IP

- **UACK :** Packet type.
- **Original Source IP :** Contains IP address of the original source node.
- **Original UID :** Unique packet ID at node sending the data packet.
- **Source IP :** Contains IP address of the node which initiated the UACK packet i.e. destination where the data has reached.

6. **MACK** Acknowledgement given by the adjacent node to indicate successful packet delivery. In short, this is simply hop to hop acknowledgement.

MACK	Original Source IP	Original UID

- **MACK :** Packet type.

- **Original source IP :** Contains IP address of the original source node.
- **Original UID :** Unique packet ID at node sending the data packet.

4.2 Algorithm

RREQ initiation

- Initiated by the source node which wish to send data and do not have path in its routing table.
- create the RREQ packet and broadcast after starting the timer
- If timer timed out then try for maximum K times by re-initiating RREQ.
- if timed out for K times then show error and drop all the packets from the queue corresponding to that destination.

RREQ processing

When RREQ packet is received by some other nodes then follow these steps.
IF UID already processed (check by comparing it with elements in processed list)

THEN skip packet.

ELSE

add *<Source IP, UID >* to the processed list.

IF Current node is destination node

THEN append current node IP to path and send RREP to previous node using path with new UID

ELSE IF the path to destination from current node is in routing table of the current node

THEN append Current IP to path and send RREP to previous

node through path and with new UID

ELSE append current IP to path and rebroadcast the packet.

RREP initiation

- Initiated by a node when either path to destination is found in the routing table or when a node itself is destination for which RREQ was intended.
- create the RREP packet with path taken from RREQ packet and unicast according to the path in the RREP packet.

RREP processing

when RREP packet is received by the node in the network follow these steps.

FOR each node that appear to the left of the current node in the path field of RREP packet

make an entry in routing table for that node with predecessor node in path field of RREP being the next hop in routing table .

FOR each node that appear to the right of the current node in the path field of RREP packet

make an entry in routing table for that node with successor node in path field of RREP being the next hop in routing table .

IF Current node is not equal to destination node in RREP(i.e. original source which initiated RREQ).

THEN send the RREP packet to the previous node in the path.

ELSE

stop the timer.

initiate data transmission process.

DATA initiation

When user gives a data send request follow the steps.
IF path to destination is available in routing table.
THEN
    Store the data packet in the queue corresponding to the destination.
    Start the timer for (UACK).
    Send packet to the node found in the routing table to the destination.
    IF UACK is received
        THEN drop the copy of the packet from the queue.
    ELSE IF timer is timed out
    THEN
        IF Number of time out is equal to K
            THEN initiate RREQ
        ELSE
            restart DATA initiation.
    IF MACK is received.
        THEN simply discard.
ELSE
    initiate RREQ.

DATA processing

When a intermediate node receives a data packet follow these steps.

Send MACK to the node from which DATA packet arrived.
IF Packet received for the first time.
THEN
    IF Current node is the destination node.
    THEN
        Consume packet.
        Send UACK.
    ELSE
        Start the timer (MACK).
        Forward the data to the next node towards the destination.

IF MACK is received
THEN drop the copy of the packet from the queue.
ELSE IF timer is timed out
THEN
IF Number of time out is equal to K
THEN
Send RERR to original source.
Remove entry for route to destination from the routing table.
Send RREQ for destination, for sending the data.
ELSE
Restart the timer and resend the packet.
ELSE
Discard Packet.

RERR initiation

RERR is initiated when a link fails or a node fails, so the node which is sending data will have time out and subsequently RERR packet will be send to sender of data packet.

RERR processing

RERR processing will be done by intermediate node who receives RERR packet. Upon receiving RERR packet follow the steps.

- From the Routing table of the node, remove all those entries where next node entry in the routing table is equal to the successor of the current node in the path field of the RERR packet.
- Add the entries in the routing table for all the nodes that are successor of the current node in the path field of the RERR packet.
- **IF** current node is not equal to destination node **THEN** prepend the node IP address and send the packet towards the destination **ELSE** do not forward RERR.

UACK initiation

UACK acknowledgement is sent by the node who has successfully received the data packet intended for the itself for the first time. UACK packet is sent to the sender of the data packet. UACK is not send if a node receives the data packet again with same packet.

MACK initiation

MACK acknowledgement is sent by the node who has received the data packet intended for some other node.

4.3 Proposal

We have implemented the DSR algorithm with some modification. Some of these are mentioned below.

- Instead of using a single acknowledgement we are using two types of acknowledgements. First one is Hop to hop acknowledgement and the other one is source to destination acknowledgement. The internal nodes (except source) upon receiving hop to hop ACK, will delete the packet from the destination's queue. The source node will remove the packet from the queue either when the hop to hop acknowledgement is sent by destination or the source to destination ACK is sent by some internal node.
- A separate queue is maintained for each destination node for which the source wants to send data. Each of these queues store the packets intended to that destination. As soon as a route reply is received by the source for a particular destination all the packets in the queue corresponding to that destination are sent through the path mentioned in the Route Reply.
- When the source has path to destination but the destination does not have path to source (it can happen when destination quits and rejoins

the network), the destination will initiate the route request back to the source so that its routing table can be up to date. Notice that in this case the source might already have received the hop to hop acknowledgement, delete the packet from the queue if the source and destination are directly connected. Otherwise the source will succeed sending the data packet during its next try since during that time destination would have also got path to source.

- In case of link failures or node failures the (internal) node at which the packet is currently residing will try to deliver the packet by initiating route request to the destination. This node will also send a route error message back to source so that the stale information about the destination will be deleted from the corresponding routing tables.

4.4 Flow Chart

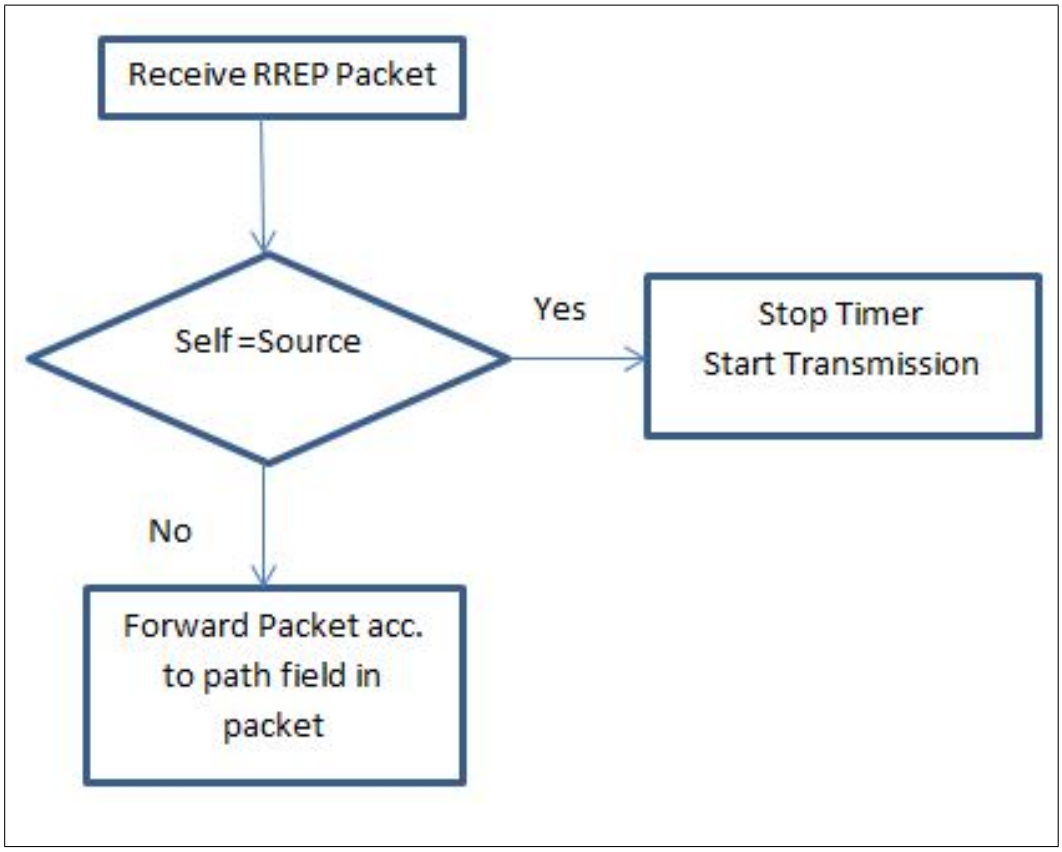

Figure 4.1: Flow chart of receiving RREP packet

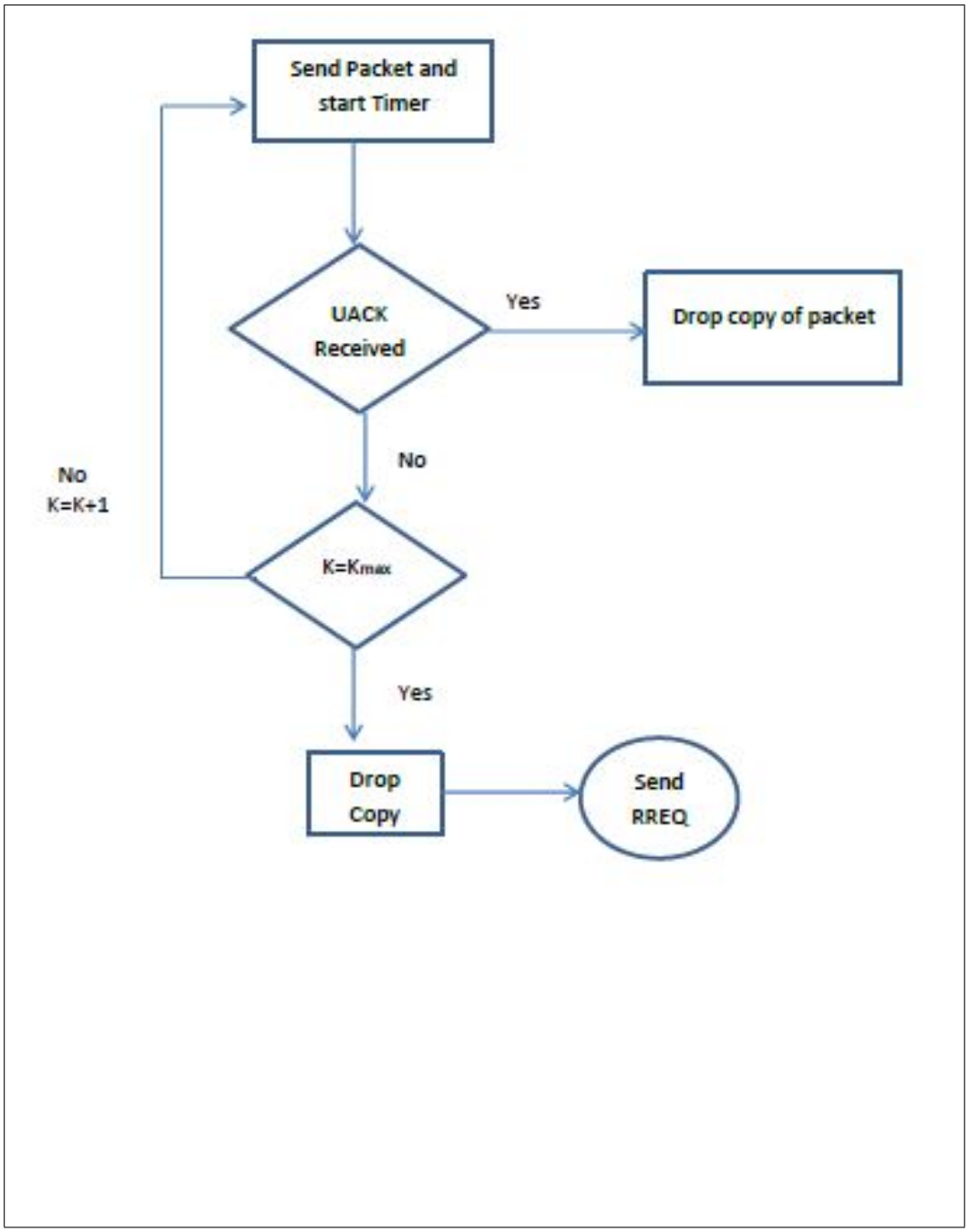

Figure 4.2: Flow chart of sending data packet

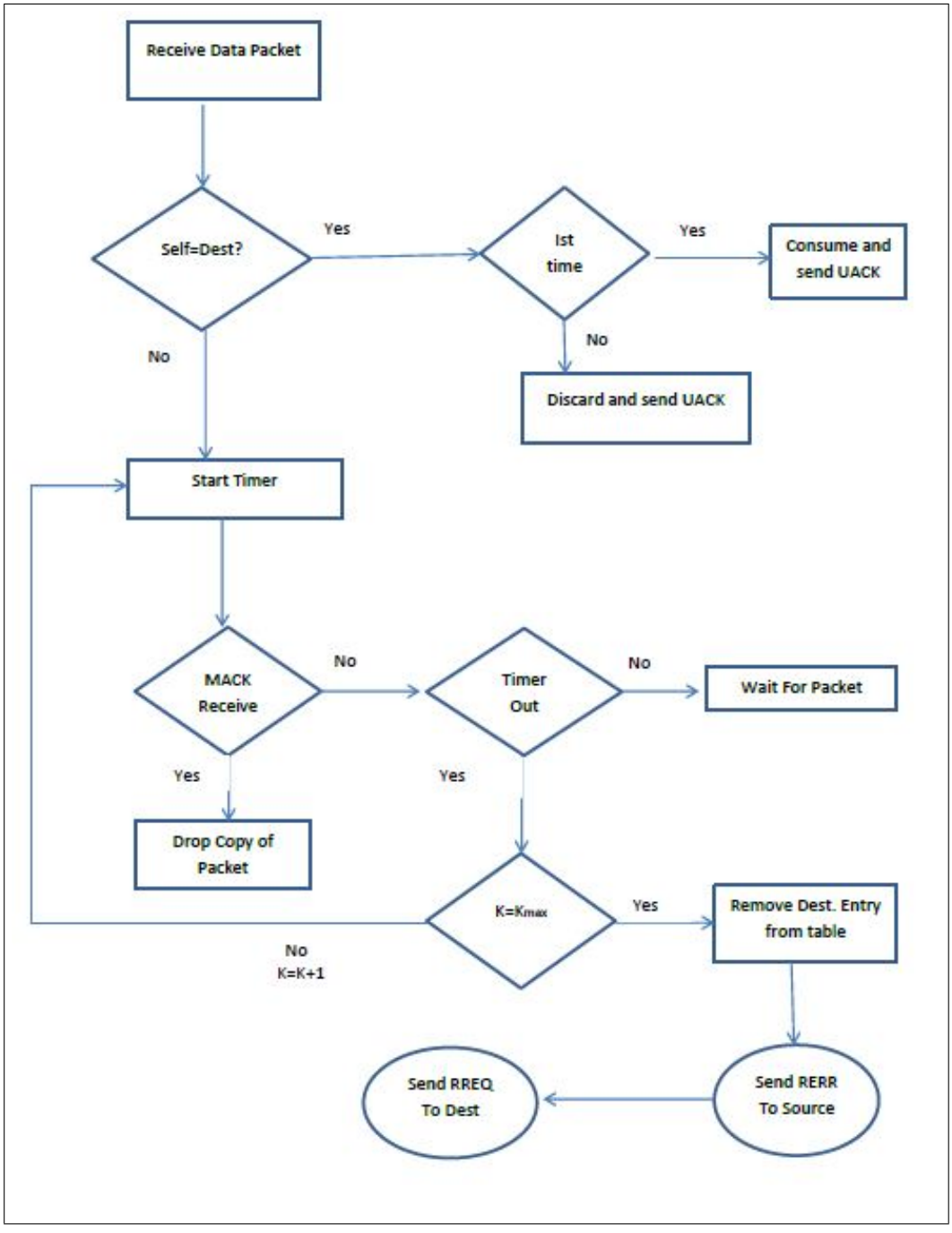

Figure 4.3: Flow chart of receiving Data packet

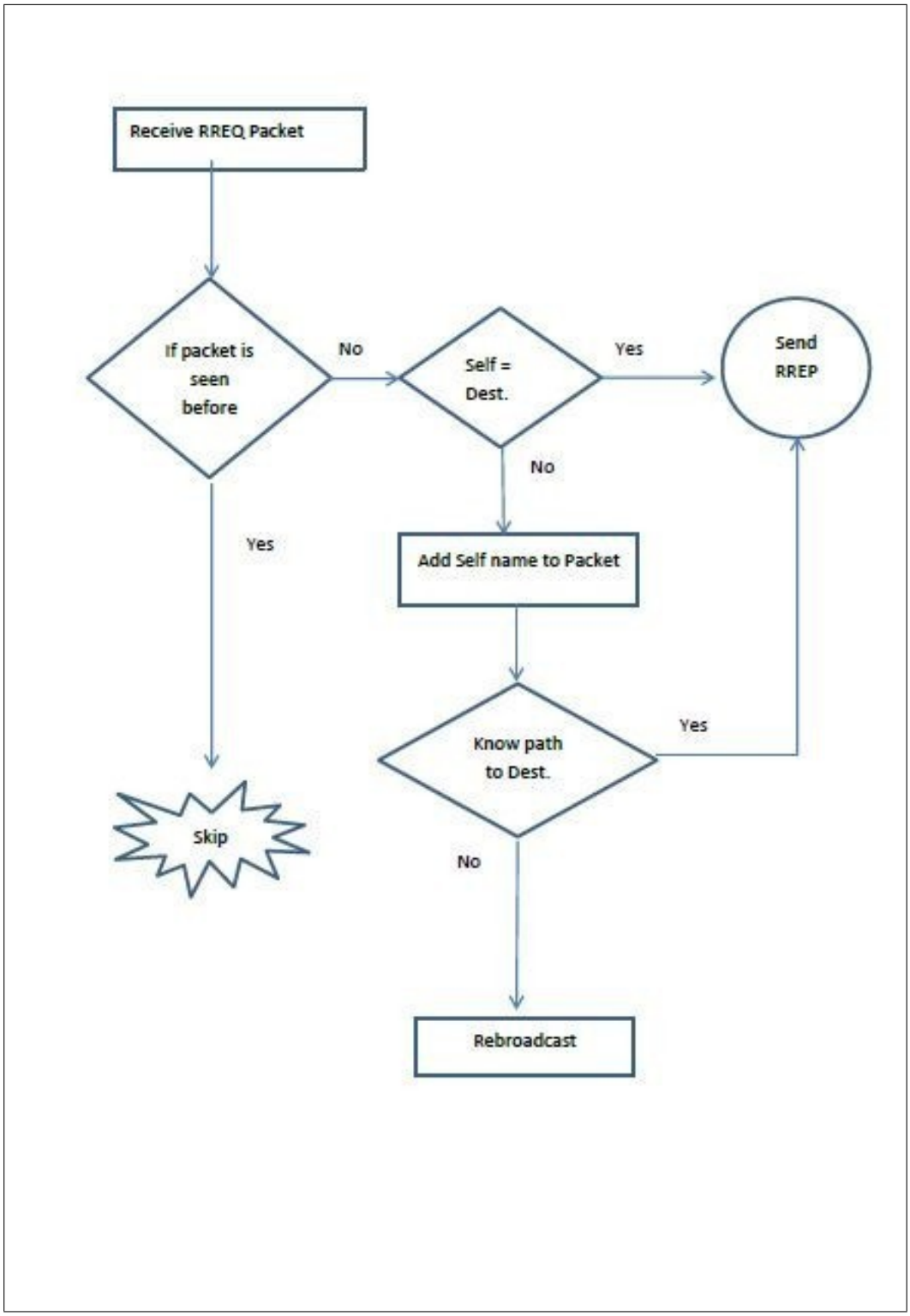

Figure 4.4: Flow chart of receiving RREQ packet

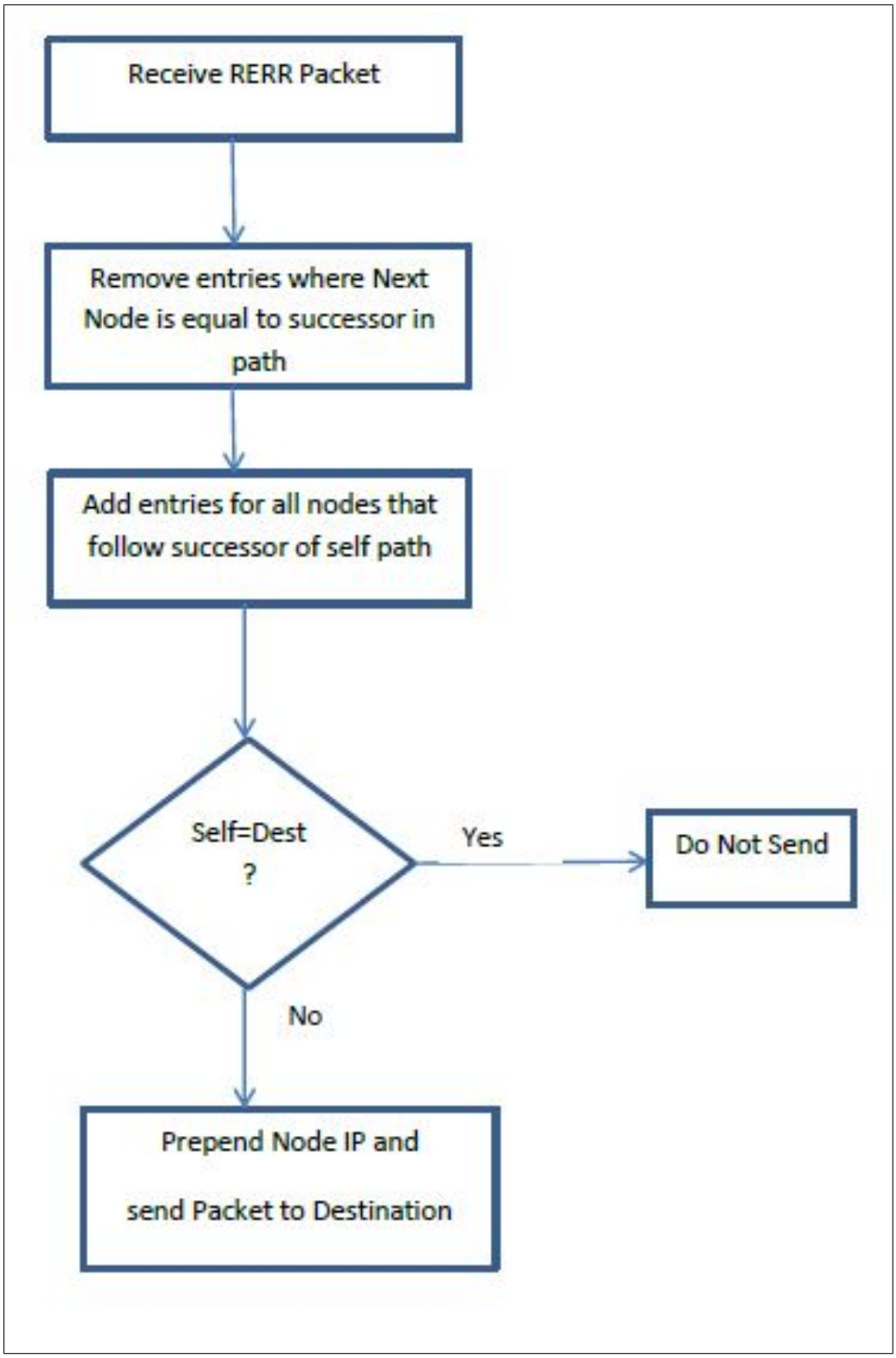

Figure 4.5: Flow chart of receiving RERR packet

Chapter 5

Testing

We tested our DSR implementation on 3 mobile nodes (laptops). In each laptop DSR algorithm was installed. We tested the algorithm in different scenarios. First test was conducted when all the nodes had there Wi-Fi on and all were in the range of each other similarly algorithm was tested for different scenarios that is when one of the link fails or when some new node comes in the network etc. We used Ad-hoc Wi-Fi mode in each laptop. In all the scenarios string messages are sent from one node to another as data. Test is considered successful if a node is able to deliver the message successfully and also getting the required ACK. Also in scenarios deliberately when nodes are failed or removed from the network, error message is displayed to show that message can not be delivered.

Scenario-1 (All the nodes are in range of each other)

In this scenario each node is in the Wi-Fi range of other nodes, that is each node is connected to other node directly. Initially we sent the data from one node to other node, since at start there was no information about the path in the routing table first RREQ was done and path was established successfully and as expected routing tables of the nodes were updated accordingly. Similarly we tried sending data from other nodes and expected result came. That is, routing tables were updated accordingly. Also messages were delivered successfully. After sending data from every node to every other node as expected all the nodes had there routing table updated and after that when data was sent from one node to another node, path was directly taken from the routing table rather than again finding from the RREQ which was correct. So all the tests in this scenario succeeded.

Scenario-2 (All node are connected but one node fails)

We followed from the previous scenario, so all the nodes were connected to each other and also every routing table was updated. Now we failed a node in this scenario and then tried two cases.

1. We sent the data from a node to the node that has been failed. The message was not delivered and according to the algorithm it tried 3 times to send data but each time it failed to send data as node was no longer available. After this, information of the destination node was deleted from the routing table and then 3 times RREQ was initiated but every time it failed as there was no node and finally data was removed from the queue and an error message was shown to the user.

2. In the second case we tried to send the data to other node in the network (which was existing), but as all the remaining nodes were connected, the data was send successfully.

All the tests succeeded in this scenario.

Scenario-3 (A new node comes into the network)

When a new node came into the network, we first sent data between the old nodes only as there routing table were correctly updated. Data was sent through that routing table and new node had no effect on network. After some time we sent data from the new node to one of the existing nodes. As new node had its routing table empty, it sent RREQ first, the path was set up accordingly and the data was successfully delivered. All the nodes in the network (including the intermediate nodes) updated their routing table. After few messages all the nodes had routing information about the other nodes.

Scenario-4 (Two nodes are connected through an intermediate node)

In this scenario node 1 was in range of node 2 and node 2 was in range of node 3. Node 1 and node 3 were not in range of each other, so 2 was the intermediate node. We tried to send the data from node 1 to node 3. As there was no path in the routing table, it sent the RREQ packet and that went to 2, who forwarded to 3 and accordingly RREP was sent. After this, all the nodes' routing table updated. That is, in 1st node's routing table, an entry (3,2) was made indicating that the node can reach node 3 through node 2. After that data packet reached node 3 successfully and ACK was also received by node 1 as expected. As nodes are mobile in MANET after some time we took the node 2 out of the range of node 3 but was still in the range of node 1 and again we tried to send data from node 1 to node 3. As node 1 table was containing the routing information it sent the data packet to node 2 and from node 2 data packet was sent to node 3 which was not in range so message delivery failed after 3 failures and 3 RREQs. It removed the data from the queue and removed the entry of node 3 in the table. Meanwhile node 1 also removed the entry of node 3 as it was unable to send the data. Also, it removed the data from the queue informing the user that the packet cannot be delivered.

In another case we again brought node 3 in range of node 2 and again sent data from node 1 to node 3 this time data was successfully and was delivered to node 3 after some RREQ and RREP messages. After this we removed node 2 from the range of node 1 and tried sending data from node 1 to node 3. But we got error message as expected since node 2 could not be reached from any path from node 1.

Snapshot

```
MESSAGE FROM 10.42.43.1 : 'MACK:10.42.43.3:3:10.42.43.1'
MESSAGE SENT TO 10.42.43.1 : 'S2P'

MESSAGE FROM 10.42.43.1 : 'MACK:10.42.43.3:3:10.42.43.1'
Could not deliver 'S2P' from 10.42.43.3 to 10.42.43.1 after trying 4 times
Packet tranmission failed more than 3 times. Removing route to 10.42.43.1 from routing table. Initiating route request.
MESSAGE SENT TO 10.42.43.3 : 'RERR:10.42.43.2:5:10.42.43.3:10.42.43.2'
Adding packet 10.42.43.2, 6 to processed list
Broadcasting RREQ
MESSAGE SENT TO 10.42.43.255 : 'RREQ:10.42.43.2:10.42.43.1:6:10.42.43.2'

MESSAGE FROM 10.42.43.2 : 'RREQ:10.42.43.2:10.42.43.1:6:10.42.43.2'
10.42.43.2, 6 already processed

MESSAGE FROM 10.42.43.1 : 'RREP:10.42.43.1:10.42.43.2:2:10.42.43.2,10.42.43.1'
UPDATING ROUTING TABLE: 10.42.43.1  ------------>   10.42.43.1
Path found: 10.42.43.2,10.42.43.1
PACKET QUEUE EXISTS. No. OF DATA PACKETS IN QUEUE = 0

MESSAGE FROM 10.42.43.3 : 'RREP:10.42.43.1:10.42.43.2:5:10.42.43.2,10.42.43.3,10.42.43.1'
UPDATING ROUTING TABLE: 10.42.43.3  ------------>   10.42.43.3
UPDATING ROUTING TABLE: 10.42.43.1  ------------>   10.42.43.3
Path found: 10.42.43.2,10.42.43.3,10.42.43.1
PACKET QUEUE EXISTS. No. OF DATA PACKETS IN QUEUE = 0
table
10.42.43.1              10.42.43.3
10.42.43.3              10.42.43.3
DSR >>  send 10.42.43.1 T2P
```

Figure 5.1: Snapshot1

```
Detected IP configurations: IP = 10.42.43.2 and Broadcast = 10.42.43.255
DSR >> Message from 10.42.43.3: 'Test 1'
DSR >> Message from 10.42.43.3: 'Test 2'
DSR >> send 10.42.43.3 Reply Test 1
DSR >> Message with id 2 delivered to 10.42.43.3
DSR >> Message from 10.42.43.4: 'Test 3'
DSR >> send 10.42.43.4 Reply Test 3
DSR >> Message with id 4 delivered to 10.42.43.4
DSR >> table
10.42.43.3              10.42.43.3
10.42.43.4              10.42.43.4
DSR >>
```

Figure 5.2: Snapshot2

```
DSR >> MESSAGE SENT TO 10.42.43.3 : 'P2S'

MESSAGE FROM 10.42.43.3 : 'MACK:10.42.43.1:3'

MESSAGE FROM 10.42.43.2 : 'UACK:10.42.43.1:3:10.42.43.3'
ACK RECEIVED FOR 3 OF 10.42.43.3
Packet acknowledged in queue
ACKNOWLEDGEMENT RECEIVED FOR PACKET 3. Dropping the packet from 10.42.43.3 queue
table
10.42.43.2              10.42.43.2
10.42.43.3              10.42.43.3
DSR >>
MESSAGE FROM 10.42.43.2 : 'DATA:10.42.43.3:8:10.42.43.1:S2P'
MESSAGE SENT TO 10.42.43.2 : 'MACK:10.42.43.3:8'
Adding packet 10.42.43.3, 8 to processed list
Message from 10.42.43.3: 'S2P'
MESSAGE SENT TO 10.42.43.3 : 'UACK:10.42.43.3:8:10.42.43.1'

MESSAGE FROM 10.42.43.2 : 'DATA:10.42.43.3:8:10.42.43.1:S2P'
10.42.43.3, 8 already processed

MESSAGE FROM 10.42.43.2 : 'DATA:10.42.43.3:8:10.42.43.1:S2P'
10.42.43.3, 8 already processed

MESSAGE FROM 10.42.43.2 : 'DATA:10.42.43.3:8:10.42.43.1:S2P'
10.42.43.3, 8 already processed

MESSAGE FROM 10.42.43.2 : 'RREQ:10.42.43.2:10.42.43.1:5:10.42.43.2'
Adding packet 10.42.43.2, 5 to processed list
Route requested to self. About to send RREP.
UPDATING ROUTING TABLE: 10.42.43.2  ------------>   10.42.43.2
Forwarding route list to the previous node
MESSAGE SENT TO 10.42.43.2 : 'RREP:10.42.43.1:10.42.43.2:4:10.42.43.2,10.42.43.1'
```

Figure 5.3: Snapshot3

```
Forwarding route list to the previous node
MESSAGE SENT TO 10.42.43.2 : 'RREP:10.42.43.1:10.42.43.2:4:10.42.43.2,10.42.43.1'

MESSAGE FROM 10.42.43.3 : 'RREQ:10.42.43.2:10.42.43.1:5:10.42.43.2,10.42.43.3'
10.42.43.2, 5 already processed

MESSAGE FROM 10.42.43.3 : 'RREQ:10.42.43.3:10.42.43.1:10:10.42.43.3'
Adding packet 10.42.43.3, 10 to processed list
Route requested to self. About to send RREP.
UPDATING ROUTING TABLE: 10.42.43.3  ------------>   10.42.43.3
Forwarding route list to the previous node
MESSAGE SENT TO 10.42.43.3 : 'RREP:10.42.43.1:10.42.43.3:5:10.42.43.3,10.42.43.1'

MESSAGE FROM 10.42.43.3 : 'DATA:10.42.43.3:9:10.42.43.1:S2P - Node 2 failed'
MESSAGE SENT TO 10.42.43.3 : 'MACK:10.42.43.3:9'
Adding packet 10.42.43.3, 9 to processed list
Message from 10.42.43.3: 'S2P - Node 2 failed'
MESSAGE SENT TO 10.42.43.3 : 'UACK:10.42.43.3:9:10.42.43.1'
table
10.42.43.2              10.42.43.2
10.42.43.3              10.42.43.3
DSR >> send 10.42.43.3 P2S node 2 failed
Adding packet 10.42.43.1, 6 to processed list
Packet added to queue of 10.42.43.3
DSR >> MESSAGE SENT TO 10.42.43.3 : 'P2S node 2 failed'

MESSAGE FROM 10.42.43.3 : 'MACK:10.42.43.1:6'

MESSAGE FROM 10.42.43.3 : 'UACK:10.42.43.1:6:10.42.43.3'
ACK RECEIVED FOR 6 OF 10.42.43.3
Packet acknowledged in queue
ACKNOWLEDGEMENT RECEIVED FOR PACKET 6. Dropping the packet from 10.42.43.3 queue
```

Figure 5.4: Snapshot4

```
MESSAGE SENT TO 10.42.43.1 : 'T2P'

MESSAGE FROM 10.42.43.1 : 'MACK:10.42.43.2:2:10.42.43.1'
Deleting packet with ID 2 from 10.42.43.1

MESSAGE FROM 10.42.43.1 : 'UACK:10.42.43.2:2:10.42.43.1'
ACK RECEIVED FOR 2 OF 10.42.43.1
table
10.42.43.1              10.42.43.1
10.42.43.3              10.42.43.3
DSR >>
MESSAGE FROM 10.42.43.1 : 'RREQ:10.42.43.1:10.42.43.3:2:10.42.43.1'
Adding packet 10.42.43.1, 2 to processed list
Path is taken from cache: 10.42.43.1,10.42.43.2,10.42.43.3
UPDATING ROUTING TABLE: 10.42.43.1  ------------>   10.42.43.1
UPDATING ROUTING TABLE: 10.42.43.3  ------------>   10.42.43.3
Forwarding route list to the previous node
MESSAGE SENT TO 10.42.43.1 : 'RREP:10.42.43.3:10.42.43.1:4:10.42.43.1,10.42.43.2,10.42.43.3'

MESSAGE FROM 10.42.43.1 : 'DATA:10.42.43.1:1:10.42.43.3:P2S'
MESSAGE SENT TO 10.42.43.1 : 'MACK:10.42.43.1:1:10.42.43.3'
Adding packet 10.42.43.1, 1 to processed list
'P2S' added to queue 10.42.43.3
MESSAGE SENT TO 10.42.43.3 : 'P2S'

MESSAGE FROM 10.42.43.3 : 'MACK:10.42.43.1:1:10.42.43.3'
Deleting packet with ID 1 from 10.42.43.3
table
10.42.43.1              10.42.43.1
10.42.43.3              10.42.43.3
DSR >>
MESSAGE FROM 10.42.43.3 : 'DATA:10.42.43.3:3:10.42.43.2:S2T'
MESSAGE SENT TO 10.42.43.3 : 'MACK:10.42.43.3:3:10.42.43.2'
Adding packet 10.42.43.3, 3 to processed list
'S2T' added to queue 10.42.43.2
Message from 10.42.43.3: 'S2T'
MESSAGE SENT TO 10.42.43.3 : 'UACK:10.42.43.3:3:10.42.43.2'
table
10.42.43.1              10.42.43.1
10.42.43.3              10.42.43.3
```

Figure 5.5: Snapshot5

Chapter 6

Conclusion

We successfully implemented the Dynamic source routing over moving mobile nodes(laptops) i.e. MANET. We have implemented DSR with some modifications. The main modules are Network, Route Discovery and Routing Table Module. We used C++ to implement DSR and made an interface for data transmission over MANET using our DSR algorithm. We tested our algorithm on different mobile nodes in different scenarios and we got the expected results.

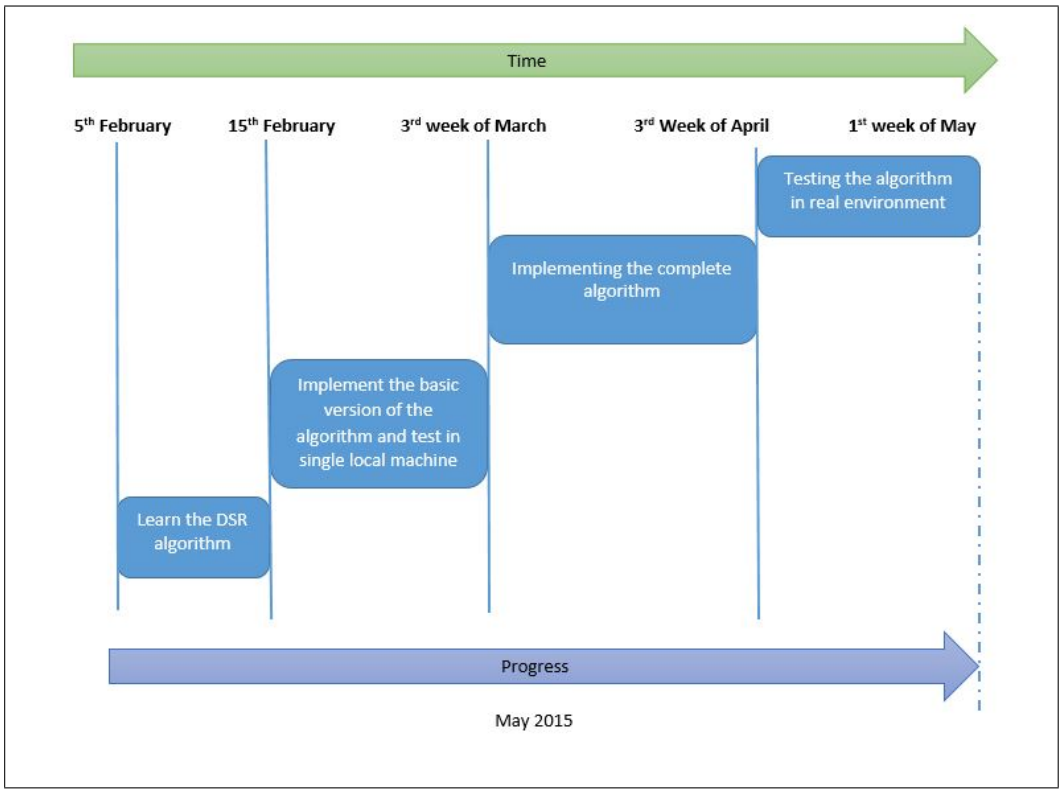

Figure 6.1: Project Progress

Sample code

This code is for processing and forwarding the RREQ, RREP and the data packets.

```
void forwardRouteReply(string source_ip, string destination_ip,
    unsigned long long int packet_id, string route_list)
{
  vector<string> ip_list = StringOperations::split(route_list, ","
     );
  int self_pos = -1;
  int count = ip_list.size();
  int i;
  for(i = 0; i < ip_list.size() - 1; i++)
  {
    if(ip_list[i] == ip_list[i + 1])
    {
      ip_list.erase(ip_list.begin() + i + 1);
    }
  }
  for(i = 0; i < count; i++)
  {
    if(ip_list[i] == ip)
    {
      self_pos = i;
      break;
    }
  }
  for(i = 0; i < self_pos; i++)
  {
       if(ip_list[i] != ip && ip_list[self_pos - 1] != ip)
       {
           cache.updateRoute(ip_list[i], ip_list[self_pos - 1]);
       }
  }
  for(i = self_pos + 1; i < count; i++)
```

```
  {
       if(ip_list[i] != ip && ip_list[self_pos + 1] != ip)
       {
      cache.updateRoute(ip_list[i], ip_list[self_pos + 1]);
       }
  }
  if(destination_ip != ip) // Or self_pos != 0
  {
    string arr[] =
    {
      string(ROUTE_REPLY), source_ip, destination_ip,
          StringOperations::to_string(packet_id), route_list
    };

    string message = StringOperations::join(vector<string>(arr, arr
        +5), DELIMITER);

    sendDataImmediate(ip_list[self_pos - 1], message);
    return;
  }
  else
  {
    if(packet_queues.find(source_ip) != packet_queues.end())
    {
      for(int i = 0; i < packet_queues[source_ip].size(); i++)
      {
        pthread_t th;
        pthread_create(&th, NULL, sendDataPacket, packet_queues[
            source_ip][i]);
      }
    }
  }
}

void forwardRouteRequest(string source_ip, string destination_ip,
    unsigned long long int packet_id, string route_list)
```

```
{
  pair<string, unsigned long long int> pkt_info = make_pair(
      source_ip, packet_id);

  if(processed_packets.find(pkt_info) != processed_packets.end())
       // Already processed
  {
    return;
  }
  processed_packets.insert(pkt_info);

  if(ip == destination_ip)
  {
    sendRouteReply((route_list == "")? ip: ((route_list.find(ip) ==
        string::npos)? route_list + "," + ip: route_list));
    return;
  }
  if(cache.isRouteCached(destination_ip))
  {
    if(route_list == "")
    {
      route_list = ip;
    }
    else if(route_list.find(ip) == string::npos)
    {
      route_list += "," + ip;
    }

    if(route_list.find(cache.fetchRoute(destination_ip)) == string
       ::npos)
    {
      route_list += "," + cache.fetchRoute(destination_ip);
    }
    #ifdef DEBUG
    cerr << "Path␣is␣taken␣from␣cache:␣" << route_list << endl;
    #endif
```

```
    sendRouteReply(route_list);
    return;
  }
  if(route_list == "")
  {
    route_list = ip;
  }
  else if(route_list.find(ip) == string::npos)
  {
    route_list = route_list + "," + ip;
  }
  broadcastRouteRequest(ROUTE_REQUEST, source_ip, destination_ip,
      packet_id, route_list);
}

void forwardData(Data *data, string sender_ip)
{
  pair<string, unsigned long long int> pkt_info = make_pair(data->
      getPacket()->getSource(), data->getPacket()->getPacketID());
  sendDataImmediate(sender_ip, string(MAC_ACK) + DELIMITER +
      pkt_info.first + DELIMITER + data->getPacket()->
      getPacketIDString() + DELIMITER + data->getPacket()->
      getDestination());
  if(processed_packets.find(pkt_info) == processed_packets.end())
  {
    processed_packets.insert(pkt_info);

    if(data->getPacket()->getDestination() == ip)
    {
      cout << "Message␣from␣" << data->getPacket()->getSource() <<
          ":␣'" << data->getPacket()->getContent() << "'" << endl
          << PROMPT_STRING;
      fflush(stdout);

      if(cache.isRouteCached(pkt_info.first))
```

```
    {
      sendDataImmediate(cache.fetchRoute(pkt_info.first), string(
          UDP_ACK) + DELIMITER + pkt_info.first + DELIMITER +
          data->getPacket()->getPacketIDString() + DELIMITER + ip
          );
    }
    else
    {
      sendRouteRequest(pkt_info.first);
    }

    return;
  }
  else
  {
    addToQueue(data);

    if(cache.isRouteCached(data->getPacket()->getDestination()))
    {
      pthread_t th;

      pthread_create(&th, NULL, sendDataPacket, data);
    }
    else
    {
      char id[25];
      sprintf(id, "%llu", next_packet_id++);
      sendDataImmediate(cache.fetchRoute(data->getPacket()->
          getSource()), string(ROUTE_ERROR) + DELIMITER + ip +
          DELIMITER + string(id) + DELIMITER + data->getPacket()
          ->getSource() + DELIMITER + ip);
    }
  }
}
else
{
```

```
    return;
  }
}

void forwardUACK(string orig_src, string orig_id, string orig_dest
    )
{
  if(orig_src == ip)
  {
    unsigned long long int id = atoll(orig_id.c_str());

    cout << "Message␣with␣id␣" << orig_id << "␣delivered␣to␣" <<
        orig_dest << endl << PROMPT_STRING;
    fflush(stdout);

    for(int i = 0; i < packet_queues[orig_dest].size(); i++)
    {
      if(packet_queues[orig_dest][i]->getPacket()->getPacketID() ==
          id)
      {
        packet_queues[orig_dest][i]->acknowledge();
        packet_queues[orig_dest][i]->in_queue = false;
        packet_queues[orig_dest].erase(packet_queues[orig_dest].
            begin() + i);

        break;
      }
    }
  }
  else
  {
    if(cache.isRouteCached(orig_src)) // To avoid pointer error.
    {
      sendDataImmediate(cache.fetchRoute(orig_src), string(UDP_ACK)
          + DELIMITER + orig_src + DELIMITER + orig_id + DELIMITER
          + orig_dest);
```

```
    }
  }
}

void forwardRouteError(string rerr_src, unsigned long long int
    rerr_pkt_id, string data_src, string path)
{
  pair<string, unsigned long long int> pkt_info = make_pair(
      rerr_src, rerr_pkt_id);
  if(processed_packets.find(pkt_info) != processed_packets.end())
      // Already processed
  {
    return;
  }
  processed_packets.insert(pkt_info);
  vector<string> ip_list = StringOperations::split(path, ",");
  cache.removePath(ip_list[0]);
  for(int i = 0; i < ip_list.size(); i++)
  {
    cache.updateRoute(ip_list[i], ip_list[0]);
  }
  if(ip != data_src)
  {
    char id[25];
    sprintf(id, "%llu", rerr_pkt_id);
    sendDataImmediate(cache.fetchRoute(data_src), string(
        ROUTE_ERROR) + DELIMITER + rerr_src + DELIMITER + string(id
        ) + DELIMITER + data_src + DELIMITER + ip + "," + path);
  }
}
```