OPTIMIZED DYNAMIC SOURCE ROUTING PROTOCOL

A project report submitted by
P.S.Sathya Narayanan (122015094)

Towards the fulfillment of award of completion of project in
Computer Networks

ACKNOWLEDGEMENTS

First and foremost, I thank the **Almighty** for helping me to gain support in all forms to finish my project successfully. I express my sincere thanks to **Dr. S.Vaidhyasubramaniyam, Vice Chancellor, SASTRA DEEMED TO BE UNIVERSITY** who provided all facilities which were required for this project. **Dr. R. Chandramouli, Registrar**, for permitting me to do this project as a part of my curriculum.

With deepest gratitude we would like to thank **Dr. S. Swaminathan, Dean- Planning and Development**, for letting us to do the project in the campus. We would like to acknowledge with much appreciation **Dr. R. John Bosco Balaguru, Dean-Sponsored Research**, who gave us to permission to use all required equipment and necessary materials to complete the project.

I am fortunate to have **Dr. N. Hemavathi ,as my class teacher.** Her valuable assistance and supervision guided me towards the successful completion of the project. Lastly, I thank all the technical and non-technical staffs of Information Technology Department and my parents for their constant support throughout my project.

BONAFIDE CERTIFICATE

This is to certify that the project work entitled **"Optimized Dynamic Source Routing Protocol"** is a bonafide record of the work carried out by **P.S.Sathya Narayanan (122015094)**

A Student of Third year B.Tech., Information Technology, in fulfillment of the **Project for Computer Networks** from the **SASTRA DEEMED TO BE UNIVERSITY, Thirumalaisamudram, Thanjavur - 613401, during the year 2018-2022.**

NAME OF THE **Class Teacher** : **Dr. Hemavathi N**

SIGNATURE:

Project Viva-Voce held on _____

Examiner –I
Examiner-II

CONTENTS

.....CHAPTER-I.....

1) INTRODUCTION

1.1) ABSTRACT

KEYWORDS: only ONE at a time,PATH ,REDIRECTION not done

We all know what routing and a routing protocol is, we may also be knowing that Proactive, reactive and hybrid are the 3 methods available to perform the routing protocol . Here I will be covering the flaws of Dynamic Source Routing Protocol . It's one of the reactive routing protocols. We will be **limiting the no. of transmissions made by a node to only ONE at a time. This is done by transmitting the packet to only the least weighed link at that time. If that path unfortunately has some breaks of links in it then we will be returning to the previous node and passing the packet to the next least weighed link.** Finally we have an **array named PATH which stores the path of the transmission of that packet.** Thus by doing this we also avoid the annoying REDIRECTION process for a simple link break (REDIRECTION might also be done for an irrelevant path).

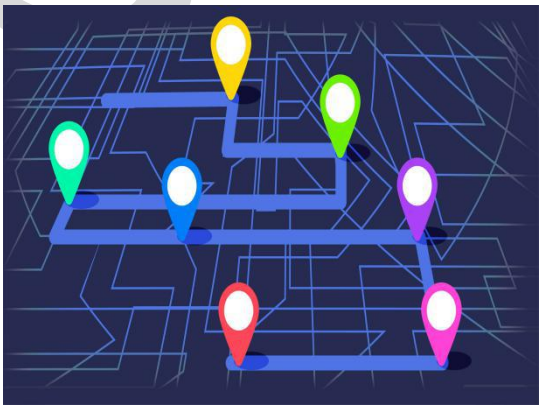

Fig 1.1.1 ROUTING

……CHAPTER - II……

2) DYNAMIC SOURCE ROUTING PROTOCOL

2.1) DEFINIATION

Dynamic source routing protocol is one of the **routing-protocols** used in the world wide level. In this protocol we forward the packet to all of the surrounding nodes in each and every step. We keep doing this until we reach the destination node. A brief explanation is given along with an example in the next section.

2.2) EXPLANATION WITH EXAMPLE

DYNAMIC SOURCE ROUTING MODEL

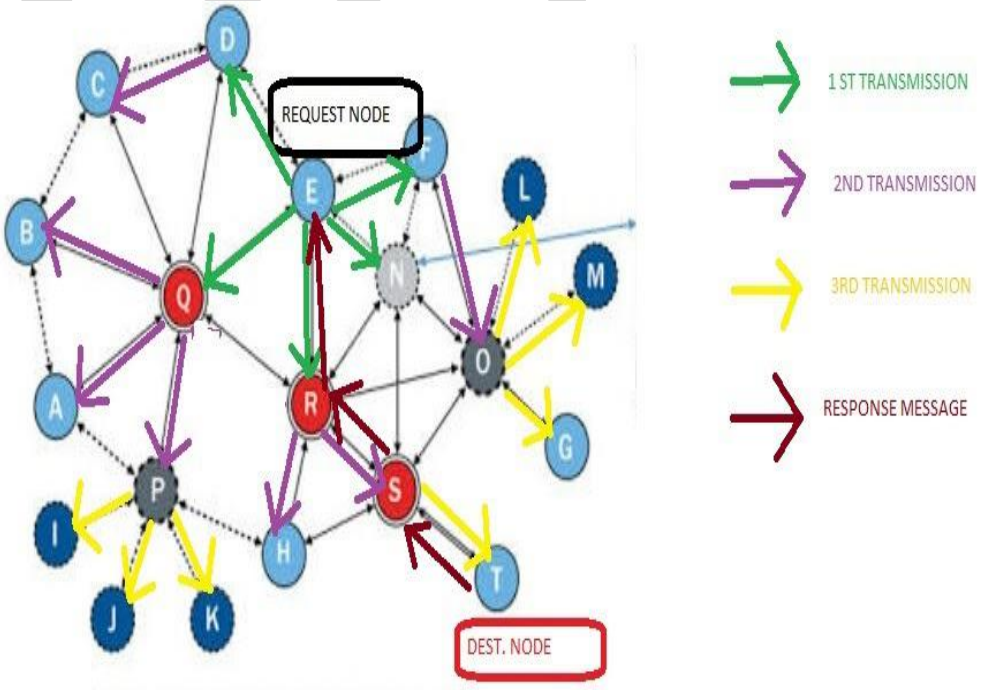

Fig. 2.2.1 DSR SYSTEM

EXPLANATION

In the above eg. we see that E is the requesting node and T is the destination node. The green,purple and yellow coloured arrows denote the 1st,2nd and 3rd pass of the request messages and here at first the node E sends it's request to all of the neighbouring nodes of it namely F,D,Q,R and N which is denoted in green colour. Then the nodes F sends it to O ,the node D to C, the node Q to B,A,P and finally the node R to H and S (i.e all the nodes pass that messages to their neighbouring nodes), those message passes are denoted by purple colour and finally the node P to I,J,K . The node O to L,M and G and the node S to T (denoted by yellow colour arrows) and finally we find the destination node T so finally we reach the destination and then now as we know the route between the source (req. node) and dest. Node the dest. Node T sends the response message to E via. The path T->S->R->E (denoted by brown colour arrows).

Fig 2.2.2 MOBILE PHONES (AN AD-HOC APPLICATION) OF DSR

……CHAPTER - III……

3) FLAWS ON EXSISTING PROTOCOL (DSR)

3.1) DEFECTS OF DSR

We see that here there are unnecessarily a lot of message transmission as each and every node passes the message to all of their neighbouring nodes . As all the transmission paths need to be traced every time and stored always there is a lot of power,energy and memory wastage and it too takes too much time and traffic. We also see that if there occurs a link-break in between the **DSR** is unable to repair itself. To avoid all these **unnecessary** wastage of essential parameters and for the property of **self-repairing** I have given a try by adding some Overheads in the already existing DSR routing protocol which we will be looking in this project of mine.

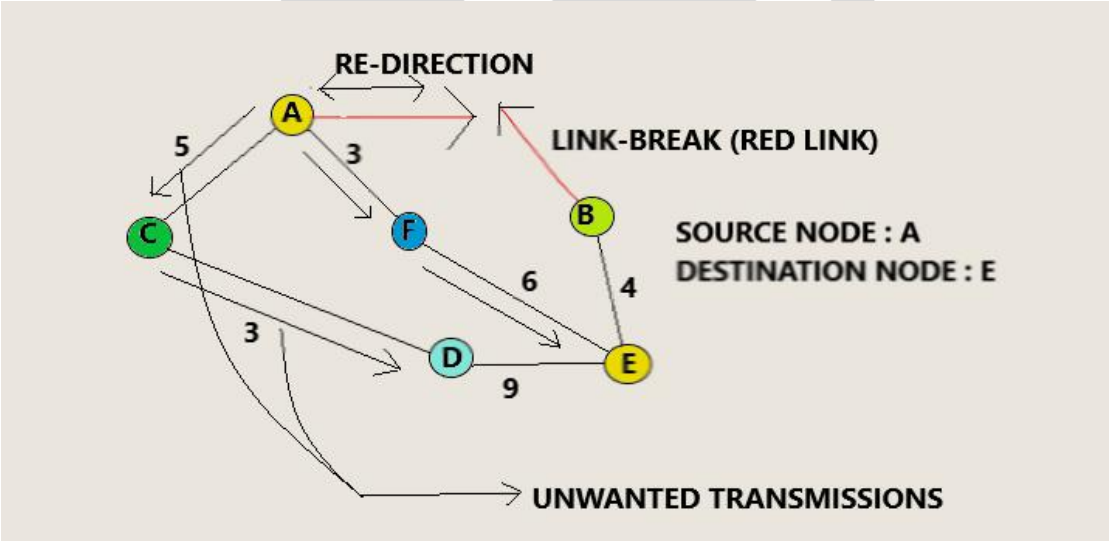

Fig 3.1.1 LINK-BREAKS & RANDOM TRANSMISSIONS

3.2) RECTIFICATIONS

I will be getting the Connection Network in form of a graph and the Link Weights in-between them in a one more graph format. The Requesting node and destination node will be got as input. From Requesting node we check weather there is a **LINK in-between the Requester and destination node**, If there is a link as per our expectation we directly transmit the packet into that node via that existing link between them. If not we look for the **link with LEAST-WEIGHT from the requester node and transmit the packet in that direction** and from that

new node once again the Least weighing link is found and packet transmission is done in that direction. We do this **until the destination node is found.** If any **link breaks** occur in-between the packet is **transmitted in the reverse direction** (to the previous node), **then the NEXT LEAST WEIGHING link** from that node is used for packet transmission.

3.3) FLAWS FIXED

1) The **Random packet transmission in every direction is avoided** which leads to Saving of power and also contributes a huge amount in Network traffic avoidance.

2) **Self-healing of transmission path done** to tackle link-breaks occurred unfortunately.

3.4) SOFTWARE USED

I have used C++ in the implementation of this methodology. I chosen this because it gives us the ability to use class (Object Oriented Programming Language) in it, Functions can also be used to do separate tasks. Thus **encapsulation, binding of various methods within a Single Class and the use of Functions motivated me towards the use of C++** in my Implementation.

Fig 3.4.1 SOFTWARE USED

4) OPTIMIZED DYNAMIC SOURCE ROUTING (ODSR) ALGORITHM

4.1) ALGORITHM

- Initialize arcn[no_of_nodes][no_of_nodes] and brcn[no_of_nodes][no_of_nodes] to be '0' in all their entries and b1[no_of_nodes][no_of_nodes] and b2[no_of_nodes][no_of_nodes] to be ∞ in all their entries. A array named path of required size is declared to store the transmission path of a packet.

a = 1
com = 0
h = $\infty - 1$
dir_link = 0
out_of_range=0
req_dst_same=0
addnodecn (src,dest,free,wt) *// to create network graph*
1) brcn[src][dest] = brcn[dest][src] = 1
2) b1[src][dest] = b1[dest][src] = wt
3) If free equal_to 1 *// free = 1 => break-free link*
    1) arcn[src][dest] = arcn[dest][src] = 1 *// free = 0 => link-break*
    2) b2[src][dest] = b2[dest][src] = wt

printconn ( ) *// Prints the network / link graph*
1) print "The Connection Network is…"
2) for i=0 upto no_of_nodes
    1) for j=0 upto no_of _nodes
        1) print " "
        2) print brcn[i][j]
    2) print "\n" (newline)
3) print "The Link Weights in-between are…"
4) for i=0 upto no_of_nodes
    1) for j=0 upto no_of _nodes
        1) print " "
        2) print b1[i][j]2) print "\n" (newline)

```
balance (c)              // finds the next node for packet transmission
1)  l =0
2)  for j=0 upto no_of_nodes
    1)  if b2[c][j] equal_to h
        1)  l = 1
        2)  arcn[c][j] = arcn[j][c] = 0
        3)  b2[c][j] = b2[j][c] = ∞
    2)  if l equal_to 1
        1) return c
    3)  if j equal_to no_of_nodes - 1
         1 ) if (a-2) less_than 0    //checks if source node lost from Network
             1 ) return -1
         2 ) c = path[a - 2]
         3 ) a = a - 2
         4 ) return c
 direct_link (req,dest)   //checks if there is a direct link b/w req. and dst.
 1)  path[0] = req
 2)  path[1] = dest
 3)  dir_link = 1
 packet_path (req,dest)   // traces transmission path of packet
 1)  if req equal_to dst
        1)  path[0] = req
        2)  path[1] = dst
        3)  req_dst_same = 1
        4)  return
 2)  if req less_than no_of_nodes - 1 or dst less_than no_of_nodes - 1
        1)  out_of_range = 1
        2)  return
 3) if arcn[req][dest] equal_to 1
     1) direct_link (req,dest)
     2) return
 4) c = req
 5) path[0] = c
 6) goto transmit
 7) transmit : for j=0 upto no_of_nodes
                       1 ) if b2[c][j] less_than h //finds the least-distant node
                            1 ) h = b2[c][j]
                       2 ) if j equal_to no_of_nodes - 1
```

1) c = balance(c)
2) if c equal_to -1
1) return
3) path[a] = c
4) break
6) if path[a] equal_to dest
1) com = 1
7) if com not_equal_to 1
1) a = a+1
2) h = $\infty - 1$
3) goto transmit

refresh ( ) *// refreshes the Network to a new one once again*

1) h = $\infty - 1$
2) com = 0
3) a = 1
4) for i=0 upto no_of_nodes
1) for j=0 upto no_of _nodes
1) arcn[i][j] = arcn[j][i] = 0
2) b1[i][j] = b2[i][j] = ∞

printpath ( ) *// Prints the transmission path*

1) if (a-2) less_than 0 and dir_link != 1 or if out_of_range equal_to 1
1) print "Necessary Node not in Network"
2) return
2) print "The Packet is transmitted in path as"
3) for i=0 upto a+1
1) print path[i]
2) if i less_than a
1) print "-->"
4)refresh( )

Fig 4.1.1

4.2) ADVANTAGES

- As we **transmit packets through only the least weighed link** we save energy and make the transmission more efficient.

- If there is a **link-break occurred we traverse along the next least weighed link which leads to self-healing of link-breakage** and also prevents packet/Datagram loss to the maximum extent unless there is some other internal fault among the nodes.

- **We avoid Network traffic** as much as possible because we transmit only in the feasible routes where destination node detection is possible.

4.3) SOURCE - CODE

```
#include<iostream>

using namespace std;

class conn                                                    // class used to bind
all func() together

{

      public:

                  static const int no_of_nodes=4;              // Here : no_of_nodes
assumed to be 4
```

```
int arcn[no_of_nodes][no_of_nodes],brcn[no_of_nodes][no_of_nodes],b1[no_of_nodes][no_of_nodes],b2[no_of_nodes][no_of_nodes];

int path[5];

int c,a=1,h=999,i,j,l,com=0;                                    // initial initializations

bool dir_link=0,out_of_range=0,req_dst_same=0;                  // initial initializations

conn()

{

    for(i=0;i<no_of_nodes;i++)

    {

        for(j=0;j<no_of_nodes;j++)

        {

            arcn[i][j]=0;

            brcn[i][j]=0;

            b1[i][j]=1000;          // Here : 1000 is assumed to be INFINITY

            b2[i][j]=1000;

        }

    }

    for(i=0;i<5;i++)

        path[i]=0;

}

void addnodecn(int src,int dst,int free,int wt) //getting the node connections

{
```

```
        brcn[src][dst]=brcn[dst][src]=1;
        b1[src][dst]=b1[dst][src]=wt;
        if(free==1)
        {
            arcn[src][dst]=arcn[dst][src]=1;
            b2[src][dst]=b2[dst][src]=wt;
        }
}
int balance(int c) // to get the next transmitted node of a packet
 {
        l=0;
        for(j=0;j<no_of_nodes;j++)
        {
            if(b2[c][j]==h)
            {
                l=1;
                arcn[c][j]=arcn[j][c]=0;
                b2[c][j]=b2[j][c]=1000;
                c=j;
            }
            if(l==1)
               return c;
            if(j==3)
```

```
                {
                    if((a-2)<0)          //checks if source node lost from
Network
                       return -1;
                    c=path[a-2];
                    a=a-2;
                    return c;
                }
            }
        }
        void direct_link(int req,int dst)    // used if there exist a free link
in-between req. and dst.
        {
            path[0]=req;
            path[1]=dst;
            dir_link=1;
        }
        void packetpath(int req,int dst)      // used to trace the path of a packet
        {
             if(req==dst)
             {
                 path[0]=req;
                 path[1]=dst;
                 req_dst_same=1;
                 return;
```

```
}
if(req>(no_of_nodes-1)||dst>(no_of_nodes-1))
{
      out_of_range = 1;
      return;
}
if(arcn[req][dst]==1)
{
      direct_link(req,dst);
      return;
}
c=req;
path[0]=c;
goto transmit;
 transmit: for(j=0;j<no_of_nodes;j++)     // transmit : used to do packet transmission
                  {
                        if(b2[c][j]<h)
                           h=b2[c][j];
                       if(j==3)
                       {
                         c=balance(c);
                         if(c==-1)
                            return;
```

```
                    path[a]=c;
                        break;
                    }
                }
            if(path[a]==dst)
                com=1;
            if(com!=1)
            {
                a=a+1;
                h=999;
                goto transmit;
            }
        }
        void printpath()                    // to print the packet's path as traced
        {
if(((a-2)<0&&dir_link!=1&&req_dst_same!=1)||(out_of_range==1))
            {
                cout<<"\n Necessary Node not in the Network";
                return;
            }
            cout<<"\n The Packet is transmitted in path as: ";
            for(int i=0;i<a+1;i++)
            {
```

```
                cout<<path[i];
                if(i<a)
                cout<<"->";
            }
            refresh();
        }
        void printconn()        // used to print initial network and their
respective link weights
        {
            cout<<"\n THE CONNECTION NETWORK:\n";
            for(int i=0;i<no_of_nodes;i++)
            {
                for(int j=0;j<no_of_nodes;j++)
                {
                    cout<<"\t"<<brcn[i][j];
                }
                cout<<"\n";
            }
            cout<<"\n THE LINK WEIGHTS IN-BETWEEN THE NODES IN
THE NETWORK:\n";
            for(int i=0;i<no_of_nodes;i++)
            {
                for(int j=0;j<no_of_nodes;j++)
                {
                    cout<<"\t"<<b1[i][j];
```

```
                }
                cout<<"\n";
            }
        }
        void refresh()    // used to refresh the network after all transmissions over
        {
            h=999;
            com=0;
            a=1;
            for(i=0;i<no_of_nodes;i++)
            {
                for(j=0;j<no_of_nodes;j++)
                {
                    arcn[i][j]=brcn[i][j]=0;
                    b1[i][j]=b2[i][j]=1000;
                }
            }
        }
};
int main()
{
    conn obj;
    obj.addnodecn(0,1,0,2);    // addnodecn( src, dst, break-less-link, link-weight )
```

```
        obj.addnodecn(0,2,1,3);
        obj.addnodecn(2,3,1,1);
        obj.addnodecn(3,1,1,5);
        obj.printconn();
        obj.packetpath(0,1);
        obj.printpath();
}
```

4.4) OUTPUT

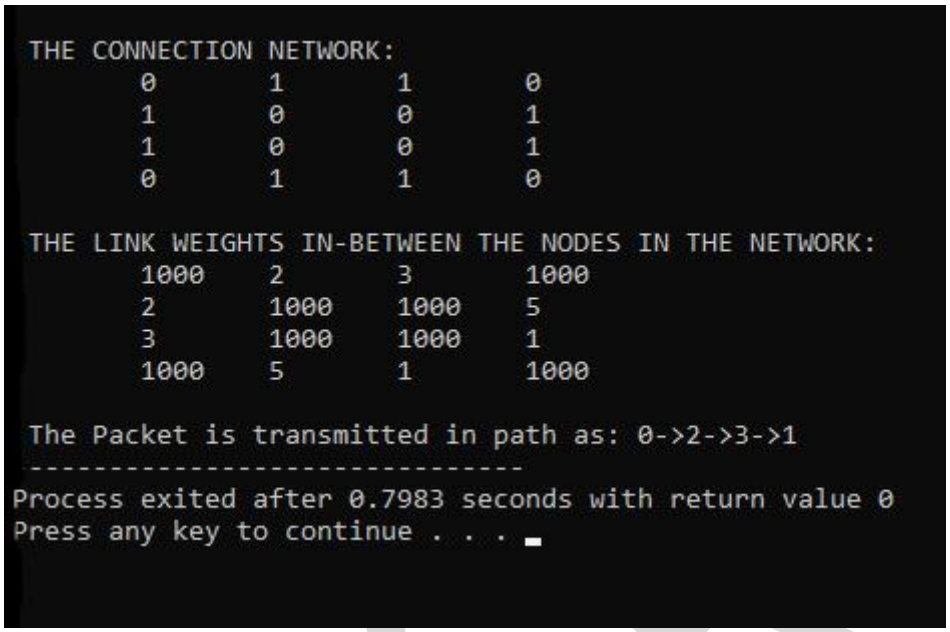

Fig 4.3.1 OUTPUT (RUN-IN DEV CPP)

.....CHAPTER - V.....

5) EFFICIENCY FACTOR

5.1) EFFICIENCY

The Efficiency of a routing protocol is termed as it's ability to reach the destination node in traversing a less distance and less power consumption. ***The lesser the no.of distance traversed before reaching the destination node the more the Efficiency*** of that certain routing algorithm in that particular network.

5.2) EFFICIENCY FACTOR

The efficiency factor tells the level of efficiency of a particular routing algorithm over a specific network of nodes for a particular transmission.It can be denoted by ***ρ.*** It can be calculated by using the formula :

$$\rho = 1 - (1 / (100 - d_p)) \text{ [if and only if } 0 < d_p < 100]$$

$$\rho = 0 \text{ [if } d_p = 100 \text{ or if } d_p = 0]$$

Where d_p is the **Percentage of distance traversed by a packet before it is delivered to the destination node.** It can be calculated by the formula :

d_p = (Total distance traversed by a packet before reaching the destination node * 100) / (Total weights of all edges in network)

5.3) EFFICIENCY FACTOR CALCULATION

The Efficiency factor can be calculated as we have done in the following example :

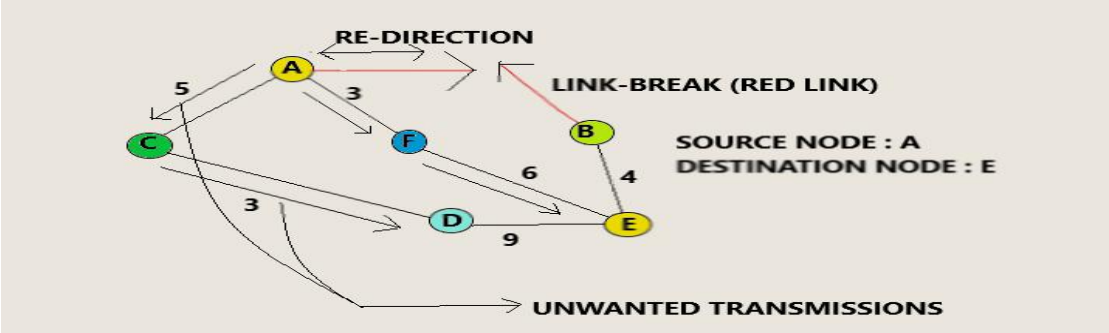

FIG 5.3.1

5.3.1) d_p calculation

The **d_p** for the **traditional DSR algorithm** on the network given in **fig 5.3.1** can be calculated by using the above given formula :

Here :

The Total distance travelled by a packet before reaching the destination node = 5 + 3 + 3 + 6 = 17

Total weights of all edges in the network is

5 + 3 + 3 + 6 + 9 + 4 = 30

Thus on applying these values onto the formula we get :

(17 * 100) / 30 = **56.66 %**

Now the **d_p** for **Optimized Dynamic Source Routing (ODSR)** algorithm over the given network of nodes in **fig 5.3.1** is :

The Total distance travelled by a packet before reaching the destination node = 3 + 6 = 9

Total weights of all edges in the network is

5 + 3 + 3 + 6 + 9 + 4 = 30

Thus on applying these values onto the formula we get :

(9 * 100) / 30 = **30.00 %**

We see that the d_p value has been significantly reduced for the ODSR algorithm. Similarly the d_p values for DSR and ODSR algorithms were noted on different Networks and the results are shown in **fig 5.3.1.1** :

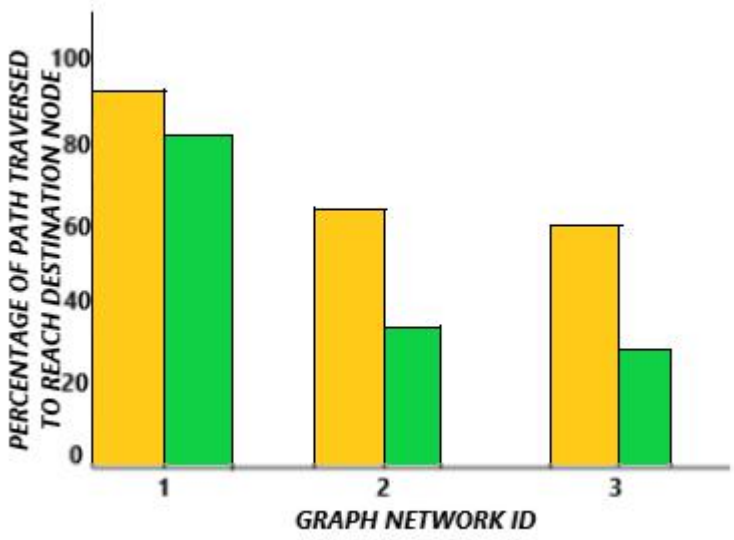

FIG 5.3.1.1

The ***X - axis*** used to plot the ***Network no.*** and ***Y - axis*** used to plot the ***d_p value in percentage*** and the Yellow colored bars refer to the d_p of Traditional **DSR** algorithm and Green bars refer to the d_p of **ODSR** algorithm in **fig 5.3.1.1.** Thus we see a ***significant decrease in d_p of ODSR*** in mostly all sorts of Network.

5.3.2) d_p for LINK-BREAKS

If there are Link-breaks at a distance r from a node then add 2r to the Total distance travelled by a packet from that node. We do so because if there occurs a link break at a distance r from a node then the packet travels r distance and after that it'll realize the link break and be returned to the transmitter node , so on returning it travels a distance r and before itself it has travelled a distance of r so

r + r = 2r

Thus **we add 2r** to the total distance travelled.

5.3.3) ρ CALCULATION

Now the Efficiency Factor can be calculated by the above given formula for **ρ as d_p value is greater than 0 and lesser than 100** and it is :

$$\rho = 1 - (1 / (100 - d_p))$$

We substitute values for both DSR and ODSR algorithms and get values as

ρ_{dsr} = 1 - 1 / (100 - 56.66)

ρ_{dsr} = 0.97

ρ_{odsr} = 1 - 1 / (100 - 30)

ρ_{odsr} = 0.98

Thus we see that there is only a slight increase in **ρ** value as we are able to decrease only one transmission for the above network **(fig 5.3.1)** for transmission between **A -> E ,** but for much larger networks and other combinations of transmissions in the same network we were able to reduce a good and considerable amount of transmissions in number which increased the **ρ** value for ***ODSR.***

We know that ***ρ is only for a single set of transmission so we need to take ρ for nC_2 times*** as ρ values need to be found for all transmissions there would be a combination of nC_2 transmissions where ***n => no. of nodes*** in the network considered. We consider only one Source and Destination at a time so we take combination for considering n nodes 2 at a time so it gives nC_2 ***after finding ρ for nC_2 times we take average of ρ by adding all ρ values and dividing it by n. The final ρ value is considered as the efficiency-factor*** of a routing algorithm over that network. For the network in **fig 5.3.1** the values are :
As there are **6 nodes n => 6**

$$nC_2 = 6C_2$$

$$6C_2 = 6\ !\ /\ (\ (6 - 2)\ !\ *\ 2\ !\ )$$

$$\mathbf{6C_2 = 15}$$

Thus we have 15 combinations so we have **15 values for ρ.**
We find those values of **ρ** by following the **steps shown in 5.3.1 and 5.3.2** .
The link break for the Network is considered to be occuring at a distance of 1 from node A and 2 from node B.
The **ρ** values for **Traditional DSR algorithm** are :
ρ_{dsr}=(0.97+0.98+0.98+0.98+0.66+0.91+0.97+0.98+0.98+0.98+0.96+0.97+0.98+0.96+0.98) / 15
ρ_{dsr} = 0.94
The **ρ** values for **Optimized DSR algorithm** are :
ρ_{odsr}=(0.98+0.98+0.98+0.95+0.98+0.97+0.97+0.98+0.98+0.98+0.98+0.95+0.98+0.98+0.98) / 15
ρ_{odsr} = 0.97

Thus we see that **$\rho_{odsr} > \rho_{dsr}$** . So we conclude that **ODSR is better than DSR.**

5.4) SPECIAL CASES OF d_p

- If **d_p = 0** the packet needed to be transmitted is **not at all left from source node** (The Source node may be isolated from the Network If all the links are broken from it).

- If **d_p = 100** then the packet is **transmitted to all the edges** and all the nodes has at-least once received that packet.

CONCLUSION

As an Engineering student I have used my knowledge and understanding on the subject of Computer Networks and tried to find the loopholes in the DSR Routing protocol. I have succeeded in finding some of them and also used my ability of Problem Solving and Solution Designing to find a way to get rid of the flaws in this DSR algorithm. I have implemented it. I share my hearty THANKS to my **Sastra Deemed To Be University** and my Computer Networks faculty **Dr.N.Hemavathi** for guiding us and helping all of us by sharing her precious knowledge by teaching this subject and granting this chance as a GEM to us for exploring and creating something and learn many things from this project. In this project we optimize the best route discovery by traversing only through the LEAST WEIGHED links and thus Power will also be saved along with avoidance of unwanted Transmission traffic.

This Algorithm suggested by me can be used in various Applications of DSR like : -

- **Mobile ad-hoc Networks (MANET)**
- **Wireless ad-hoc Networks (WANET)**
- **Vehicular ad-hoc Networks (VANET)**
- **Flying ad-hoc Networks (FANET)**
- **Wireless Sensor Networks (WSN)** etc…

REFERENCES

1) A modified route discovery approach for DSR
https://www.researchgate.net/publication/322658030_A_MODIFIED_ROUTE_DISCOVERY_APPROACH_FOR_DYNAMIC_SOURCE_ROUTING_DSR_PROTOCOL_IN_MOBILE_AD-HOC_NETWORKS

2) Performance Evaluation and Comparison of DSR,MDSR and RDSR routing protocols available Online at www.ijcsmc.com.

3)**A Study on DSR Routing Protocol in Ad-hoc Network for Daily Activities of Elderly Living**
https://www.researchgate.net/publication/331610239_A_Study_on_DSR_Routing_Protocol_in_Adhoc_Network_for_Daily_Activities_of_Elderly_Living

4) Simulation of ad-hoc networks with DSR protocolhttp://netlab.boun.edu.tr/papers2/Iscis2001-DSR-TamerDEMIR+.pdf

5) A Survey paper on DSR in ad-hoc networks
http://ijsrd.com/Article.php?manuscript=IJSRDV2I10021