A NOVEL NODE FAILURE MANAGEMENT FOR MOBILE AD HOC NETWORK USING ANT AGENTS

Ramkumar K.R [#1], Dr.Ravichandran C.S[#2]

[#1]Associate Professor, Department of Computer Applications, Sri Venkateswara College of Engineering
Pennalur, Sriperumpudur,Tamilnadu ,India

[#2]Professor & Head, Department of Electrical and Electronics Engineering,
Sri Ramakrishna Engineering College, Coimbatore, Tamil Nadu, India
[#1] ram@svce.ac.in , [#2] eniyanravi@gmail.com

Abstract-The node failure management is a puzzling task in Mobile Ad hoc Networks because of its dynamic nature. Even a single node or link failure can collapse the entire network. The source nodes are taking all responsibilities to manage node failures in the existing systems, but a link failure could be dealt dynamically with the help of alternative paths to deliver payloads without disturbing the source node. The end to end delay and the population of routing packets could be reduced enormously with an effective node failure management at intermediate level. The payload buffering at intermediate nodes need a special attention to decide the optimal buffer size. Here we propose a novel idea to handle node failures at run time to improve the following factors - high packet delivery ratio, low jitter effect and optimized usage of buffer space in mobile devices.

Keywords: MANETs, Node Failure, Probability, greedy method.

I. INTRODUCTION

An extensive research work is going on routing algorithms of mobile adhoc networks. The basic standards are DSDV[1], OLSR[3], AODV[8], DSR[4] , AntNet, ARA[11] and AntHocNet[10].The ant colony optimization takes a lead role in routing algorithms. The payload is transmitted based on the *pheromone* values calculated by forward and backward ants. The pheromone values is measure of time and queuing delay of a link that is used to calculated the probability of goodness. The beacons (hello packets) are exchanged between neighbours periodically to test the existence. A link fails because of various factors like low battery, barriers of signals, rapid movement of devices and others. A *Rerr* (Route Error) message is sent to source node during link failures, the source node initiates route discovery and path updating program once again from beginning. This regular routine is a lengthy process and node failures are very frequent in mobile adhoc networks [1]. Obviously, it is required to implement an effective frame work to manage node failures at run time. The following chapters are giving the essence of complete work. The basic algorithms and architecture are described in second chapter. The chapter three is describing about the new node failure management and chapter 4 is discussing about results and analysis. The conclusion and future work have been discussed in last chapter.

II. BASIC ALGORITHMS

A. Forward and Backward ants

The forward ants[10] are sent to destination at regular intervals to preserve and to optimize existing routes, as well as to discover new routes. If the forward ant is not engaged to the current node then the node pushes its own IP address and travels further until it reaches destination or MaxHopCount. The duplicate forward ants are destroyed easily with the help of ant and source ids. A forward ant is converted into backward ant after reaching the destination. The rationale of backward ant is to retrace the path of a corresponding forward ant to update pheromone values. It uses the information stored in forward ant and travels in reverse path to change to update routing tables to reflect the current status of network more accurately. It is forwarded via high priority queues that are not used by regular packets.

B. Source node Algorithm

1) Payload preparation:

The new structure named Ant Payload is the combination of forward ant (travels in predetermined path) and a payload to be transferred; in other turn an ant is embedded with payload to give routing map and to discover newer routes in the event of node failure. The second step is to calculate the main attribute "*optimal hop count"* which is 30 % of the total path length; this value is calculated to decide the number of backups that have to be maintained in a path.

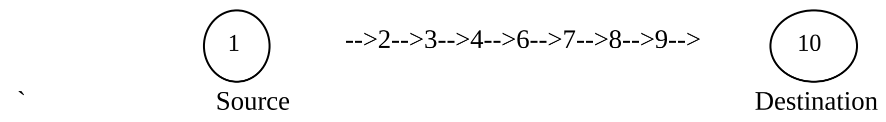

Fig.1. Sample Path

for instance ,if the path length is 10 and 30 % i.e.) 3 is the optimal hop count in which duplicate copies of payload are buffered. These duplicate buffers are used to re transmit packets during the node failures.

The following algorithm infuses a forward ant in a payload and calculates *OptHc*, for instance

Algorithm 1 : Payload preparation

```
Input    : AntPayLoad[index] =   Payload[index] + forwardant(unicast)

Output : $R_{ant}$ - Reverse ant for acknowledgment
---------------------------------------------------------------
Initialize i with zero
for each hop increment i
          begin
// Antpayload is an array, attached with forward ant and   OptHc is calculated

                    AntPayload[i]=attach(Payload[i]+$f_{ant}$[i])
                    $f_{ant}$[i].OptHc= Ceil ((30/100)* length of link))
                    $r_{ant}$[i]=SendPayload(AntPayload[i])

// Resend packets when Senpayload not returns success
  if $r_{ant}$[i].Status!=SUCCESS

                    Resend(AntPayload[i])
  else

                    Delete AntPayload[i]
 end
// Check for the completeness of payload delivery, resend payloads if pending any
for each Item in Payload increment i
 begin
          if( AntPayload[i]!=NULL)

                    startNewTransmission(AntPayload[i])
  else
          continue
 end
```

2) Intermediate Node

The intermediate nodes have three foremost responsibilities

i) Forwarding payload to neighbours

ii) Taking backups [Buffering]

iii) Handling node failures

The proposed algorithm states a new idea to keep buffers at intermediate nodes, these buffers could be used to rebroadcast payload in the occurrence of node failures. The buffers will not be dumped in all intermediate nodes but these are stored in limited 'n' number of nodes which are having alternative paths or last 'n' number of nodes. The *'n'* value is $Opt_{imal}H_{op}C_{ount}$ ie) ceiling value of 30 % of total path length.

C. Algorithm of SendPayload(AntPayload[i])

The working model of this algorithm is given below with a path sampling .

```
Input: AntPayload  i th payload

Output: Acknowledgment
// Check for  error free reception of data
if  ReceivePayload .(Payload[i])==SUCCESS
   then
   begin
 // if payload is not for the current node  then forward payload after buffering payload
  if $f_{ant}$[i].dst!=CurrentNodeID   then
      begin
                    BufferPayload[AntPayload[i]]
                    Extract($f_{ant}$ from AntPayload[i])
                    $f_{ant}$[i].hopcount++
//if the hop count is crossing OptHopcount then    acknowledgement is sent to proper node to delete it  buffers
since  buffers are not going to be maintained in   all nodes.
if $f_{ant}$[i].hopcount >=OptHc   then
             begin
 // Create a backward ant  i.e.) copy of forward ant i
                      Copy($f_{ant}$[i],$b_{ant}$)
// The hop count is decremented by one so that again  it   will    come within the range of optimal hop count and
can  travel  further till it reaches destination.
                    $f_{ant}$[i].hopcount=$f_{ant}$[i].hopcount-1
// The following code sends acknowledgement to the  correct  intermediate node   that has the buffer.  The node
is  identified with the help of stack
                    $b_{ant}$.HopCount=1
                    $b_{ant}$.MaxHopCount=OptHc
                    bant.sourceAddr=CurrentNode
                    bant.destAddr=node from stack
                    entry[perform pop operation for OptHc times]
                    send($b_{ant}$)
                    Forward(AntPayload[i],$N_{list}$)
   else
  // The forward ant which did not cross optimal hop   count travels as normal forward   ant towards
    destination.
                    Forward(Payload[i],$N_{list}$)
   end
  // if a payload reaches destination then send   SUCCESS via res array.
     else if $f_{ant}$[i].dst == CurrentNodeID
        then
         begin
                      res[i]=$f_{ant}$[i]
                      res[i].Status=SUCCESS
                      send res[i] to source node
// Finally  Garbage collection function deletes all buffers in    a path  if any pending buffers exist.
                      GarbageCollection(res[i])
             end
           end
        end
```

1) Practical Model

Step 1: Optimal hop count value is calculated	Hop-count * 30/100
	Pathlength=5
	OptHc= Pathlength * 30/100=Ceil(1.50)=2
	Backup limit=2

Step 2: PL- Payload is transferred from node 1 to node 2.

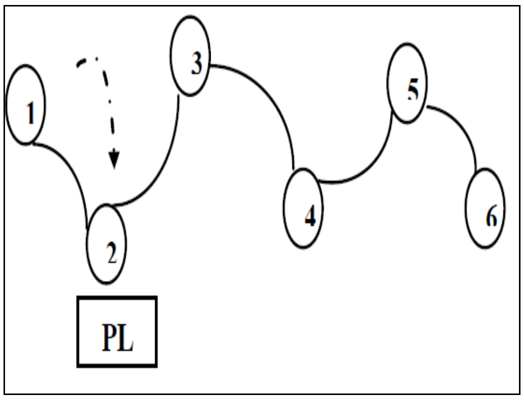

Fig. 1. Payload delivery from Source

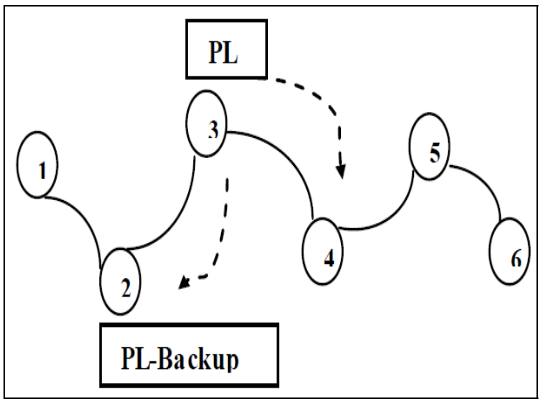

Fig.2. Backup Management

Step 3: PL-Payload travels further ,
hopcount =hopcount+1 && hocount<OptHc

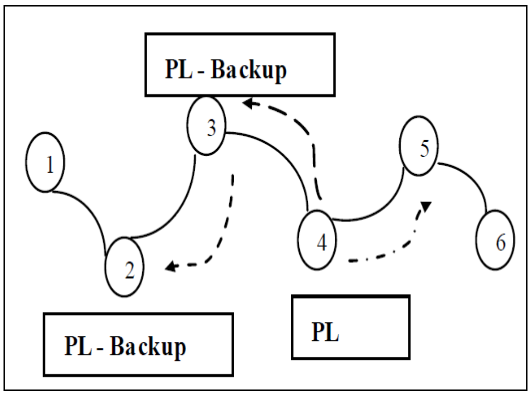

Fig. 3. Second Backup

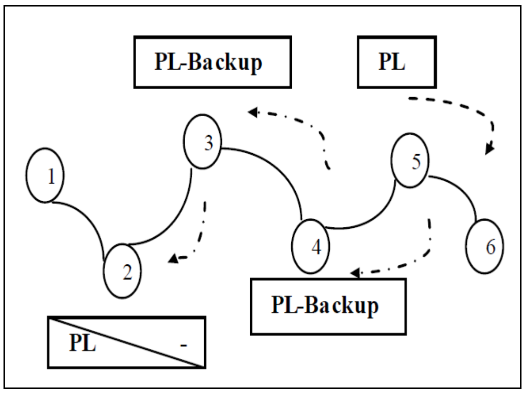

Fig. 4. Backup Deletion

Step 4: After reaching 5 th node the hocount(4)>OptHc so backup is deleted in node 2.
Step 5: Maintains 2 backups at a time

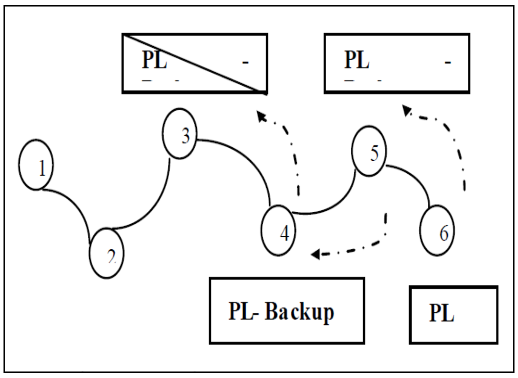

Fig 5: Garbage collection

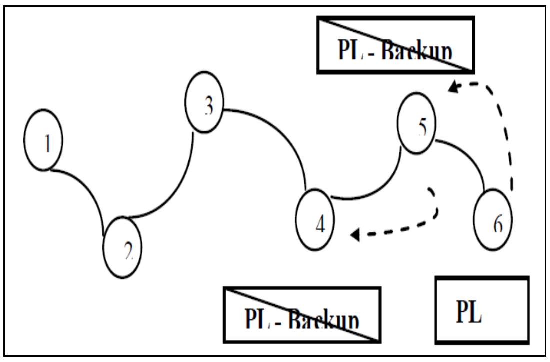

Fig. 6. Garbage collection

III. BUFFER MANAGEMENT

The function makes a backward ant to travel from current node to intermediate node which could be reached in Optimal Hop Count limit. After reaching the intended recipient the backward ant deletes proper buffer entry from Buffer Payload. The number of temporary buffers is limited to OptHc at any point of time in a network.

A. Algorithm of SinglePathTravel (B_{ant})

```
Input: Backward ant swapped from    forward ant.
Output: Delete entry from buffer.

b_ant.HopCount++
if b_ant.HopCount>=b_ant.MaxHopcount   then
            begin
 for i starts with 1 to N  then
            begin
if  b_ant.src == BufferPayload[i] . AntPayload.src     &&   b_ant. Dst == BufferPayload[i].AntpayLoad.dst
             Delete BufferPayload[i]
  end
  else
             SinglePathTravel(b_ant)
 end
```

B. Garbage Collection

It is a simple recursive call that deletes all buffers stored in between source and destination nodes, This function is being executed when a payload reaches the destination successfully.

1) Algorithm of Garbage Collection

```
// It is a recursive call which delete all  buffers in between destination to source
Garbage Collection(Res)
UnipathTravel (Res)
if Res.src!=CurrentNodeID   then
  begin
            delete buffers
            GarbageCollection(Res)
 End
```

C. Node Failure Management Algorithm

During link failures a node cannot transfer payload further, So it checks the previous node for the other possibilities. The previous node is taken from the forward ant stack that may be belonging to these following categories

A) It may be a source node B) It may be an intermediate node with an alternative path

C) It may be an intermediate node which has no alternative path

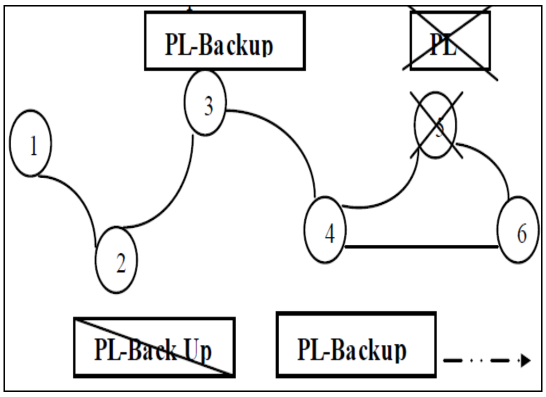

Fig.7. Node failure

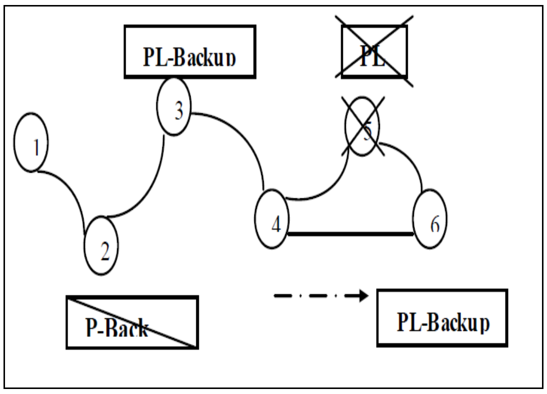

Fig. 8. Node failure recovery

The following algorithm describes about how a node failure is handled, in the above case, the node 5 fails so cannot proceed further and payload is sent from 4 to 6 since it has a backup. It is an intermediate setup to reduce the overhead.

1) Algorithm ForwadPayload

Input : AntPayload, neighbour list
Output: AntPayload is translated to next node without node failure.

```
if isFound($N_{list}$.NextHopEntry for Destination N)==false
  then
  begin
 // get previous node entry from dynamic stack to test the availability of backup
        newNode=pop($f_{ant}$.stack)
 // if newNode is the payload originating source means . start a fresh transaction

 if newNode==SRC then
        begin
                discard($F_{ant}$)
                startNewTransmission(payload[i])
        else
  // if it is an intermediate node then try to send data from there itself.
  if (newNode has an alternative path)
                send buffered data from intermediate node
  else ( if newNode has no alternative path)
                retry from previous nodes until the last backup.
end
end
```

IV. SIMULATION

The simulation is implemented in SWANS simulator with 100 mobile devices in 1000 m^2 area with random waypoint mobility model. The parameters coverage range, propagation delay and *Ber*(Bit error ratio) have been taken from simulator defaults. The backup availability (alternative path availability) of different paths is tested because it is important to have good number of backups to implement ad hoc node failure management. The tested results are categorized in to 2 modules. The first module is spreading of 30 to 40 nodes in 1000 X 1000 meters square. The simulation is executed at different speeds ranging from 1 m/s to 10 m/s to calculate throughput.

NTPA : The Number of time of alternative paths availability.

NTNAAP: Number of times non available of alternative paths

$$BAT = \frac{NTPA}{NTPA + NTNAAP}$$

Equation (1)

The figure 9 shows the throughput value ranges from 0.4 to 0.6 and sometimes alternative paths are not available too. This indicates that 50 % of success rate for buffering payload at intermediate nodes.

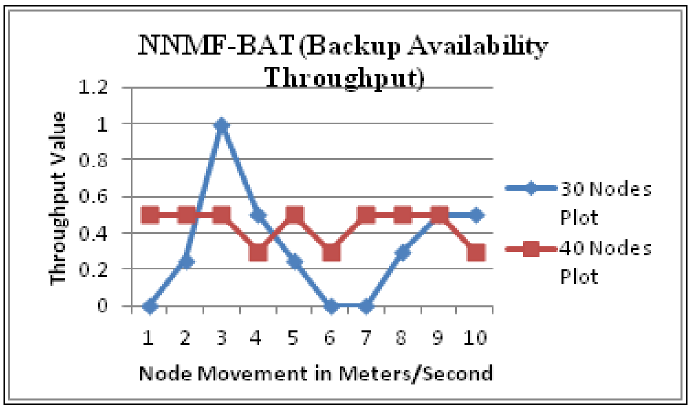

Fig. 9. NNMF-BAT for 30 - 40 nodes

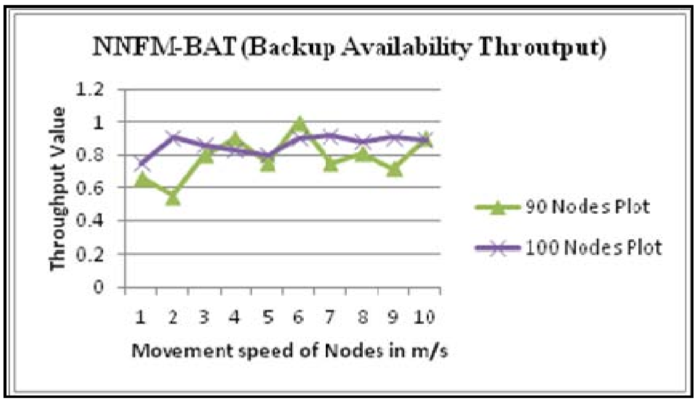

Fig. 10. NNFM-BAT for 90-100 nodes

The graph 10 shows that how throughput reaches the value 1 when 90 to 100 nodes are plotted in the same area 1000 X 1000 meters. Most of the time alternative paths are available so we can implement node failure management algorithm by taking backups at intermediate nodes .

The AODV algorithm is tested with different node failure scenario, 40 to 100 nodes have been plotted in 1000 X 1000 meters area, where node movement speed is slowly increased from 1ms to 10ms. The overall processing from sending error message to discovering newer routes from source node is a circuitous task and consumes more number of hop counts. The detailed simulation results show that maximum it takes 18 hop counts, obviously it takes high end to end delay.

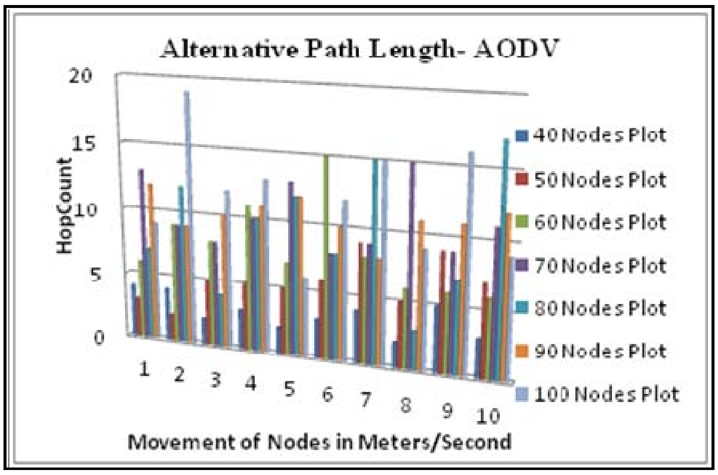

Fig. 12. Alternative Path Length

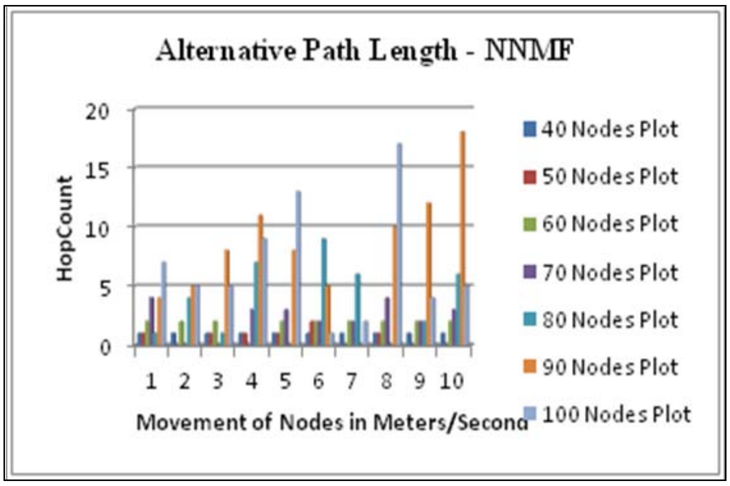

Fig. 13 .Alternative Path Length

The NNMF algorithm is tested in Figure 13. with different node failure scenarios, 40 to 100 nodes have been plotted in 1000 X 1000 meters area, where node movement speed is slowly increased from 1ms to 10ms. The overall processing: from sending error message to discovering newer routes from intermediate node during node failure is a simple task and consumes less number of hop counts.

The following graph shows the average hop count that have been taken to discover new route when a node fails and it shows clearly that NNFM outperforms than AODV in most of the cases.

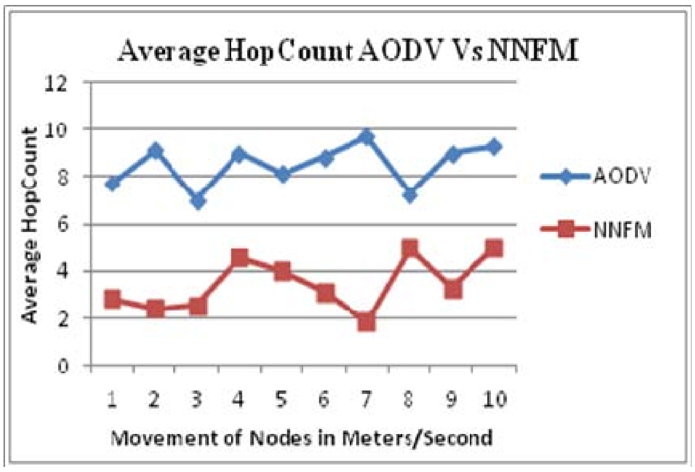

Fig. 14. Average Hop count comparison

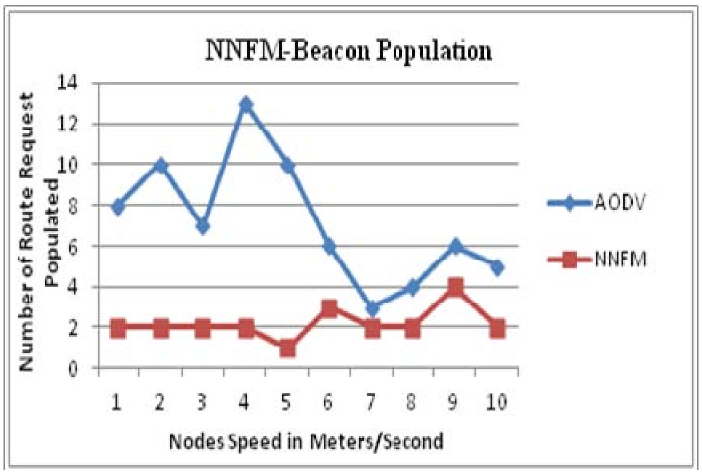

Fig. 15. Beacon Population

The figure 15 shows the number of route request generated to discover newer routes in the event of node failures. It is obvious that NNFM confronts AODV in most of the cases.

V. CONCLUSION AND FUTURE WORK

The complete dependency on source node during node failure is minimized with the help of intermediate nodes. The intermediate nodes hold buffers to retransmit payloads in the event of node failures. The optimal number of buffers, the correct selection of alternative paths and unicast of error messages all have been given as set of algorithms. The simulation results show the effectiveness of these algorithms. The future direction could be the better buffering techniques and evaluation of a heavily loaded node that acts as an intermediate router for several nodes. The performance analysis could be tested in real mobile environment.

REFERENCES

[1] Larry C. Llewellyn, Kenneth M. Hopkinson, Member, IEEE, and Scott R. Graham " Distributed Fault- Tolerant Quality of Wireless Networks " IEEE TRANSACTIONS ON MOBILE COMPUTING, VOL. 10, NO. 2, FEBRUARY 2011.

[2] Goo Yeon Lee and Zygmunt J. Haas, Fellow," Simple, Practical, and Effective Opportunistic Routing for Short-Haul Multi-Hop Wireless Networks" IEEE TRANSACTIONS ON WIRELESS COMMUNICATIONS 1536-1276/11 2011.

[3] Ramkumar, K.R., GaneshKumar, M., Hemachandar, N., Prasadh, M.: D"HPRAAM: Hybrid Parallel Routing Algorithm Using Ant Agents for MANETS. IJET (March 2009) ISSN:1793- 8244,1793-8236.

[4] A.S. Alzahrani and M.E. Woodward, “End-to-End Delay in Localized QoS Routing,” Proc. IEEE Int’l Conf. Comm. Systems (ICCS), pp. 1700-1706, 2008.

[5] P. Yang and B. Huang, “QoS Routing Protocol Based on Link Stability with Dynamic Delay Prediction in MANET,” Proc. Pacific-Asia Workshop Computational Intelligence and Industrial Applications(PACIIA), pp. 515-518, 2008.

[6] R. Ghosh and S. Basagni, “Mitigating the Impact of Node Mobilityon Ad Hoc Clustering,” Wireless Comm. and Mobile Computing,vol. 8, no. 3, pp. 295-308, 2008.

[7] Di Caro, G., Ducatelle, F., Gambardella, L.M.: AntHocNet: An Adaptive Nature-InspiredAlgorithm for Routing in Mobile Ad Hoc Networks, Tech. Rep. No. IDSIA-27-04-2004,IDSIA/USI-SUPSI (September 2004).

[8] C.Perkins, E. Belding-Royer, and S. Das, “Ad Hoc Ondemand Distance Vector (AODV) Routing,” IETF RFC 3561, July 2003.

[9] Ben Liang, Zygmunt J. Haas "Optimizing Route-Cache Lifetime in Ad Hoc Networks" IEEE INFOCOM 0-7803-7753-2/03 2003.

[10] Guine, M., Sorges, U., Bouazzi, I.: ARA-the ant-colony based routing algorithm for MANETs. In: Proc. of IWAHN 2002, pp. 79–85 (August 2002).

[11] S. Nelakuditi, Z.L. Zhang, R.P. Tsang, and D.H.C. Du, “AdaptiveProportional Routing: A Localized QoS Routing Approach,” IEEE/ACM Trans. Networking, vol. 10, no. 6, pp. 790-804, Dec. 2002.

[12] S. Chakrabarti and A. Mishra, “QoS Issues in Ad Hoc Wireless Networks,” IEEE Comm. Magazine, vol. 39, no. 2, pp. 142-148, Feb.2001.