A Congestion Controlled Multipath Routing Algorithm Based On Path Survivability Factor

A.K. Daniel[1], Shishir Dwivedi[2], Tarun Verma[3], Pankaj Kumar Dubey[4]

Department of Computer Science and Engineering
M.M.M. Engineering College Gorakhpur (Uttar Pradesh)

ABSTRACT

Mobile wireless ad hoc network has emerged as the self organized wireless interconnection for the various applications in random topology. However achieving reliable transmission in mobile wireless network is crucial due to change in the network topology caused by node mobility. Congested links are also an expected characteristic of such a network as wireless links inherently have significantly lower capacity than hardwired links and are therefore more prone to congestion. The protocol adapts quickly to routing changes when host movement is frequent, yet requires little or no overhead during periods in which hosts move less frequently. This protocol finds alternate mechanism that will find out optimal and reliable paths in terms of congestion and number of hop counts. It will use the brighter part of DSR algorithm. Also the route maintenance process is different from that of DSR because concept of **delay time** and **alert packets** is used.

Keywords: RRP (Route Request Packet), **UPD** (Update Packet), alert packet, delay time, stream array, survivability factor

1. INTRODUCTION

Mobile wireless ad hoc network are communication network built up of collection of mobile devices which can communicate through wireless links. Routing is the task of directing packets from source to destination. This is particularly hard in mobile wireless ad hoc networks due to the mobility of the network elements and lack central control. Source routing is a routing technique in which the sender of a packet determines the complete sequence of nodes through which to forward the packet; the sender explicitly lists this route in the packet's header, identifying each forwarding "hop" by the address of the next node to which to transmit the packet on its way to the destination host. Source routing has been used in a number of contexts for routing in wired networks, using either statically defined or dynamically constructed source routes. The protocol presented here is explicitly designed for use in the wireless environment of an ad hoc network. When a host needs a route to another host, it dynamically determines one based on cached information and on the results of a *path discovery* protocol. Dynamic source routing protocol offers a number of potential advantages over conventional routing protocols such as distance vector in an ad hoc network. First, unlike conventional routing protocols, our protocol uses no periodic routing advertisement messages, thereby reducing network bandwidth

Battery power is also conserved on the mobile hosts, both by not sending the advertisements and by not needing to receive them.

It uses *dynamic source routing* to route packets in an ad hoc network. Source routing is a technique in which the source node determines the entire sequence of nodes a packet has to pass through, when it is being transmitted from source to destination. The source node puts the list of addresses of all nodes in the header of the packet, so that the packet is forwarded to the destination through those specified nodes. However source routing can be done statically or dynamically, this does it dynamically. This is done using a procedure called *path discovery*. Whenever a node has packet to send to some other node, the first node initiates the path discovery. Each node maintains a cache called *path cache* to store the routes it has gathered to different destinations. As the nodes in an ad hoc network move from place to another, some of the existing links break and the routes in the path caches of the nodes must be modified. This is done using a procedure called *path maintenance. Path discovery* and *path maintenance* are two main mechanisms used by this protocol. Section 2 of this paper deals with previous works, section 3 covers the proposed mechanism, finally section 4 addresses the conclusions.

2. PREVIOUS AND RELATED WORKS

2.1 Qualities of a good routing algorithm

•Provides loop-free routes
•Provides multiple routes (to alleviate congestion)
•Establishes routes quickly (in case of link failure)

2.2 DSR

DSR [1, 2, 3, 5] is an algorithm that can be used in mobile wireless networks for routing purpose. But it provides only single path routing, although, it could be amended to support multipath routing. More significantly, it suffers from a scalability problem due to the nature of source routing. As the network becomes larger, control packets (which collect node addresses for each node visited) and message packets (which contain full source routing information) also become larger. Clearly, this has a negative impact due to the limited available bandwidth.
Each mobile host participating in the ad hoc network maintains a *route cache* in which it caches source routes that it has learned. When one host sends a packet to another host, the sender first checks its route cache for a source route to the destination. If a route is found, the sender uses this route to transmit the packet. If no route is found, the sender may attempt to discover one using the *route discovery* protocol.
Route Maintenance is the mechanism by which a node sending a packet along a specified route to some destination detects if that route has broken, for example because two nodes in it have moved too far apart. DSR is based on *source routing*: when sending a packet, the originator lists in the header of the packet the complete sequence of nodes through which the packet is to be forwarded. Each node along the route forwards the packet to the next hop indicated in the packet's header, and attempts to confirm that the packet was received by that next node; a node may confirm this by means of a link-layer acknowledgment, passive acknowledgment, or network-layer acknowledgment. If, after a limited number of local retransmissions of the packet, a node in the route is unable to make this confirmation, it returns a ROUTE ERROR to the original source of the packet, identifying the link from itself to the next node as broken. The sender then removes this broken link from its Route Cache; for subsequent packets to this destination, the sender may use any other route to that destination in its Cache, or it may attempt a new Route Discovery for that target if necessary. Unlike routing protocols using distance vector or link state algorithms, DSR uses dynamic source routing which adapts quickly to routing changes when host movement is frequent.

3. DETAIL DESCRIPTION OF PROPOSED ALGORITHM

Proposed mechanism operates on mobile wireless networks. This algorithm inculcates good qualities of DSR algorithm for selecting optimal routing path. Our proposed algorithm uses an *alert packet* and *UPD packet*.

3.1 DELAY TIME

Delay time ( ) is 2.5 times the average time required by the alert packet to traverse the most distantly separated nodes in the network (i.e. diameter of the tree) at the condition of average congestion level i.e. to the number of hops necessary for a packet to reach from any host located at one extreme edge of the network to another host located at the opposite extreme, as the *diameter* of the network.
In case of very good conditions the time taken by a message to travel to and fro between two nodes is at least 2 times the time taken by the packet to travel in one direction. So, in ideal conditions a source is supposed to get acknowledgement after 2 times the time packet takes to travel in one direction. But in real due to traffic it is not the case. So, we have assumed that the source gets the acknowledgement after approximately 2.5 times the time taken by the packet to travel in one direction.
Delay time will be governed by following factors:

- Traffic
- Distance between two farthest pair.

3.1.1 Traffic:
Traffic is the average congestion that can exist in longest path of the network for timely delivery of alert packet in worst case (i.e. highly congested network).

3.1.2 Distance between Two Farthest Pair
It is quite obvious that the average time taken by the alert packet to reach from any destination to any source will always be less than or equal to the average time taken by the alert packet to traverse the longest distance possible in the same network i.e. delay time for the diameter of the network will be maximum.

3.2 ALERT PACKET

Alert packet is a type of acknowledgement packet. It is small size packet of fixed length (i.e. its size will not increase as it visits node). It is sent from destination to source at regular intervals of time (*delay time*) on the same path that has been selected for sending data packets (in reverse order). Its purpose is to tell source that the current working path is still valid.

3.3 STORAGE MECHANISM USED FOR THE ALGORITHM

This algorithm uses variant of initial phase of DSR algorithm (i.e. route construction phase) for deciding all possible paths between given source and destination. Every node has a *path cache*. *Path Cache* is a kind of buffer that holds information about paths emanating from that node as a source. When a node has a packet to send to some destination and does not currently have a route to that destination in its *Path Cache*, the node initiates *Path Discovery* to find a route; this node is known as the *source* of the path discovery, and the destination of the packet is known as the *destination*. The source transmits a *ROUTE REQUEST PACKET (RRP)* as a local broadcast, specifying the target and a unique identifier from the source.

Each node possesses a *stream array(S)***. S**ize of this array is equal to number of adjacent neighbours to that node.

For determining the size of the stream array, let a node ***i*** broadcasts a small packet to its adjacent neighbours. Now, all nodes which receive this packet respond by sending another small packet to node ***i***. The number of respond packet the node ***i*** receives is equal to the size of the stream array. This process is done at regular intervals of time. Initially all elements of this array are assigned to ***null***. Each element can have only two values either 0 or 1. "*0*" for *upstream* link from neighbour node to given node and "*1*" for *downstream* link from given node to neighbour node. Here, downstream link from ***i*** to ***j*** (or upstream link from ***j*** to ***i***) doesn't mean that the link ***i->j*** will be used for transferring packet from ***i*** to ***j***. It only means that the path from the source to destination will contain a link ***i->j*** i.e. data packet will be transferred from ***i*** to ***j*** and the acknowledgement will follow link ***j*** to ***i***.

3.4 PATH DISCOVERY

Whenever a node has a packet to send to another node, it checks its *path cache* first. If there is no existing route for the required destination node in the path cache, the source node broadcasts a *route request* with a *sequence number* and *destination address*. The *sequence number* identifies a particular route request. Each node maintains a table called *request table*, which maintains the information about all the route requests the node has received before. The source node makes a note of the route request it is has sent in the request table.

Let ***i*** be the node that broadcasts RRP and ***j*** be one of the adjacent neighbours of ***i***. Whenever node ***i*** broadcasts a RRP it updates only those elements of its stream array that have ***null*** value present in it by inserting 1 in $\mathbf{S_{ij}}$ where ***j*** belongs to one of the neighbours of ***i*** . Correspondingly, neighbours that will receive RRP will update their stream array by inserting 0 in $\mathbf{S_{ji}}$ where ***j*** receives packet from ***i***. Now, ***j*** will also broadcast this received RRP to all its adjacent neighbours and updates only those elements of its stream array which have ***null*** value present in it. After receiving RRP, neighbours of ***j*** will respond in similar manner as described above. So, as RRP travels it appends node id of all the visited nodes until destination node is found.

When a node receives the route request, it checks its stream array. And if the node is the destination of the route request, it sends a route reply back to the source node by simply copying the path from the RRP. If the node is an intermediate node, it checks its path cache. If it has a route to the destination in its cache, the node sends a route reply to the source node by copying the path from its cache and the path from the RRP. If the intermediate node does not have any available route to the destination in its path cache, it rebroadcasts the RRP.

After receiving the first route reply, source sends all the data packets destined to the destination using the path retrieved from the route reply.

Additional route reply that source node gets will act as an **alternate** to the current working path.

The route reply packet that destination sends contains these three parametric values of that path:

Number Of Hops Travelled (N): As route reply packet travels back it counts number of nodes traversed.

Traffic Load (T) At Any Node:

Traffic Load (T) = (Outgoing traffic at that node) - (Incoming traffic at that node).

Now outgoing and incoming traffic at any node may be calculated by the size of output and input buffer of that node. Note that input buffer may act as output buffer and vice versa. So,

Node store in its file the information about the traffic load by using values 0 or 1. Every node applies above formula to calculate traffic load and assign 0 or 1 depending upon:

0: for negative traffic load i.e. incoming traffic is more than outgoing traffic (or traffic at the node is more) and

1: for positive traffic load i.e. outgoing traffic is more than incoming traffic (or traffic at the node is less).

As route reply packet travels from destination to source it extracts this value from every travelled node. Finally, when it reaches the source node it has the traffic load value of all the nodes (that can exist) of corresponding paths. So, more the number of zeros in any path, more the traffic in the path and less the number of zeros in the path, less the traffic.

Bandwidth (B): As route reply travels from destination to source it visits all nodes and links joining those nodes that come in path. So, as it travels it stores the bandwidth of that link in it which has minimum value among all traversed links. Let this value be **B**.

We stored the minimum bandwidth value of all links that the route reply traverses so as to guarantee that at least **B** amount of bandwidth is available in the path.

Now, **Survivability Factor** for every path will be calculated on the basis of above three parameters.

SF=function(B,T,N)

More the number of hops (N) less survivable is the path. As the number of hops increase number of links will also increase so chances of link failure is more. More the traffic load (T) in the path less survivable is the path. More the bandwidth (B) more survivable is the path.

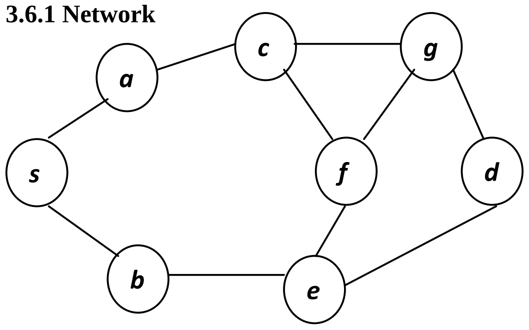
On the basis of SF we shall prioritize the paths i.e. more the SF more the priority and more survivable the path is. After the determination of the first path, node will start sending the packets and for all subsequent route replies it shall calculate SF for them and prioritize them accordingly.

3.5 TREE

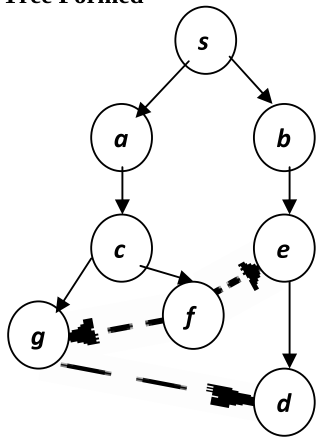
Now, the directed tree will be constructed, which will contain all the possible paths between source and destination. For the very first path a tree structure will be formed, and for all the subsequent route replies only those nodes and links will be created that are not present in current tree, if a node is already present in the tree then a threaded link will be formed. So, ultimately the outcome will be a directed tree. In this tree the source will be the root of the tree and destination will always be one of the leaf nodes. This tree will contain all the possible paths from the source to the destination. This tree will be stored in the root cache of the source node.

3.6 PROTOTYPE MODEL

3.6.1 Network



3.6.2 Tree Formed



3.6.3 Values In The Stream Array Of Each Node

For Node ***s***, $S_{sa}=S_{sb}=1$

For Node ***a***, $S_{ac}=1$ and $S_{as}=0$

For Node ***c***, $S_{cg}=S_{cf}=1$ and $S_{ca}=0$

For Node ***b***, $S_{be}=1$ and $S_{bs}=0$

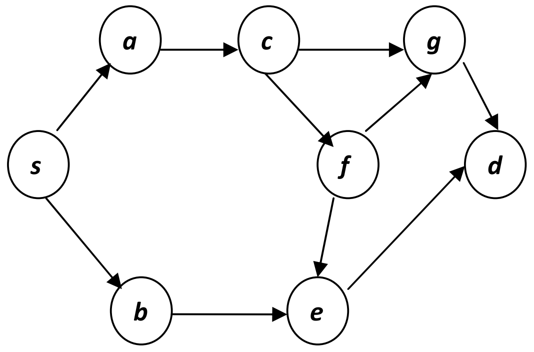
For Node ***e***, $S_{ed}=1$ and $S_{eb}=S_{ef}=0$

For Node ***f***, $S_{fe}=S_{fg}=1$ and $S_{fc}=0$

For Node ***g***, $S_{gd}=1$ and $S_{gc}=S_{gf}=0$

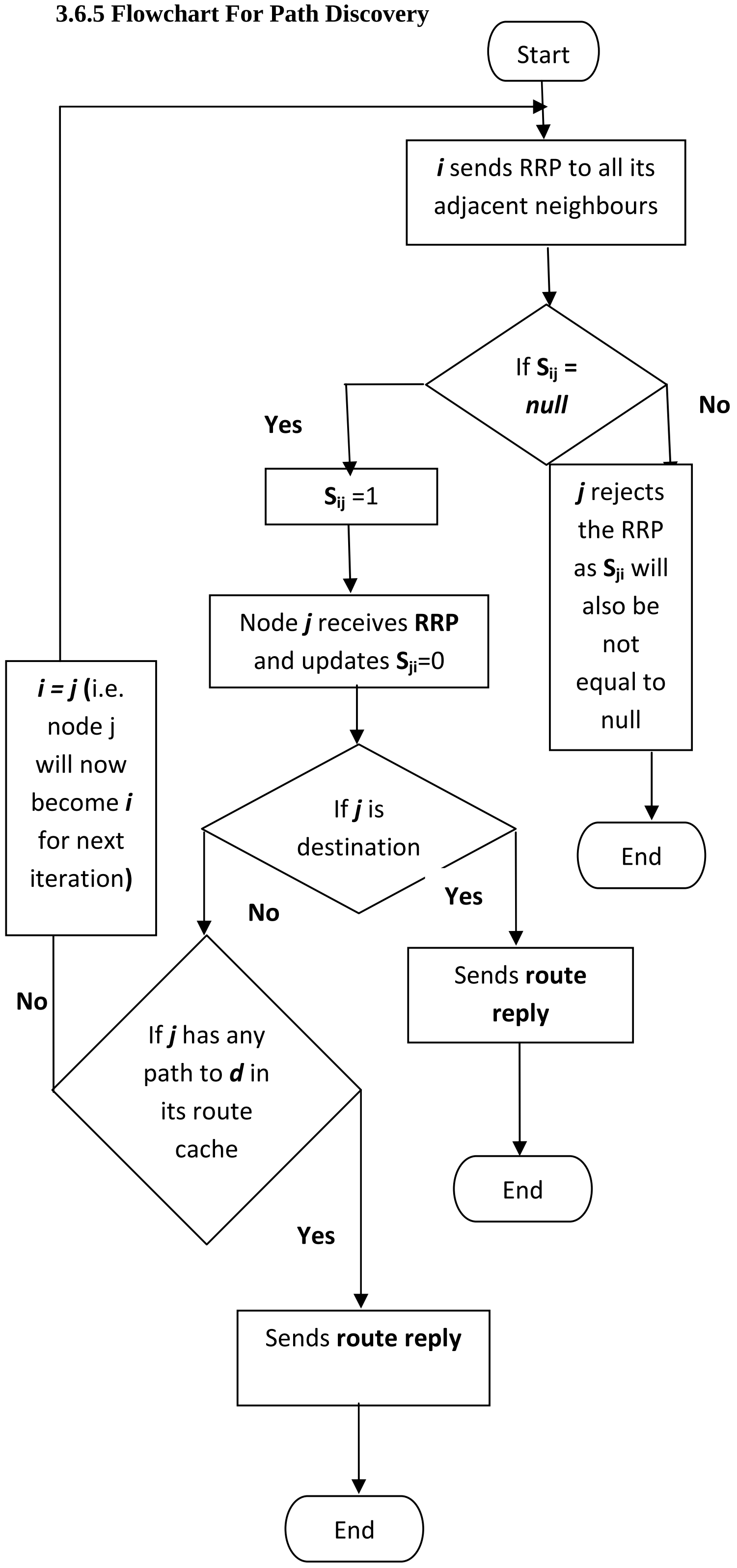
For Node ***d***, $S_{dg}=S_{de}=0$

3.6.4 Paths Created



1. ***s->a->c->g->d***
2. ***s->a->c->f->g->d***
3. ***s->a->c->f->e->d***
4. ***s->b->e->d***

3.6.5 Flowchart For Path Discovery


Start
i sends RRP to all its adjacent neighbours
If S_{ij} = null
Yes
No
S_{ij} =1
j rejects the RRP as S_{ji} will also be not equal to null
Node j receives RRP and updates S_{ji}=0
End
If j is destination
No
Yes
i = j (i.e. node j will now become i for next iteration)
No
If j has any path to d in its route cache
Sends route reply
End
Yes
Sends route reply
End

3.7 PATH MAINTENANCE

Whenever a node ***i*** sends a data packet to the next hop, it waits for the acknowledgement from the receiving node ***j***. Node ***i*** may not receive the acknowledgement packet for some reason. The reason may be due to the failure of link present between ***i*** and ***j*** or due to high congestion present in that link. If the node ***i*** detects that the link to the next hop (i.e. ***j***) is in error; it sends the data packet to previous node in that path (i.e. ***k***) and checks for the availability of any path from that node to destination in its path cache.

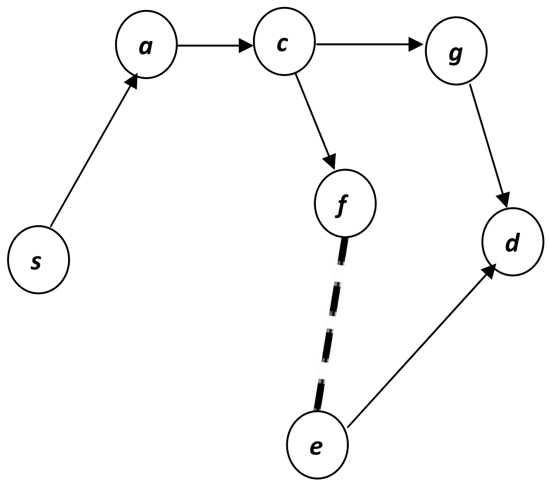
Two cases arise:
(1) If it finds any path it changes the route map of the data packet and also sends an UPD packet to the source ***s*** informing about the particular link failure and amended path. After receiving the UPD packet the source deletes all paths that contain failed link. Also it updates newly chosen path as the current working path.

(2) If path is not found it simply sends the UPD packet to the source informing about the particular link failure. Now source does the processing according to the conditions given below.

Again two cases arise:
(i) If this UPD packet reaches the source before the expiry of the delay time, source deletes all those paths that contain the failed link. Now the source chooses the alternate path having the highest priority from its path cache.
(ii) If delay time expires before the arrival of UPD packet source ***s*** chooses alternate path that have highest priority from its path cache and sends the data packet through it. When the UPD packet arrives at source, the source checks whether failed link is contained in the new chosen path or not and simultaneously it deletes all paths that contain failed link. If the failed link is found source annuls the current transaction in chosen path and selects the path having the highest priority (highest value of survivability factor) as the new working path and sends data packet through it. And if the failed link is not found in the newly selected path then the source continues to send data packets through the newly selected path.

If source doesn’t have any alternate path then it again starts the path discovery process.

3.7.1 Example

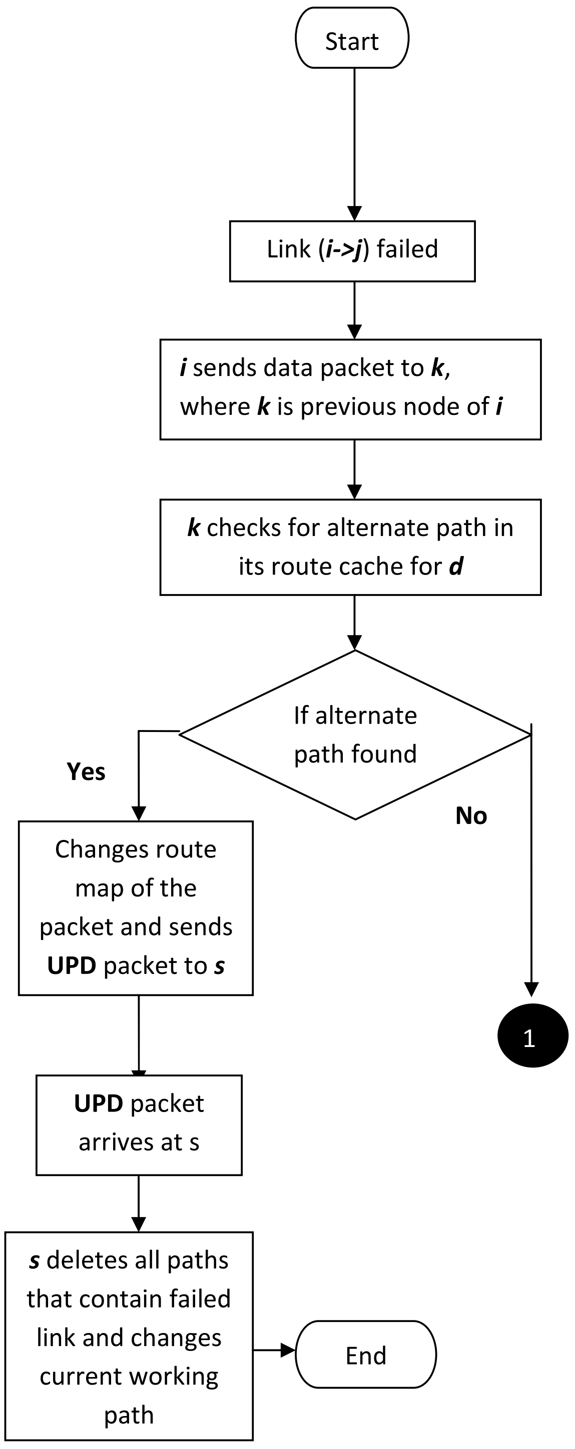


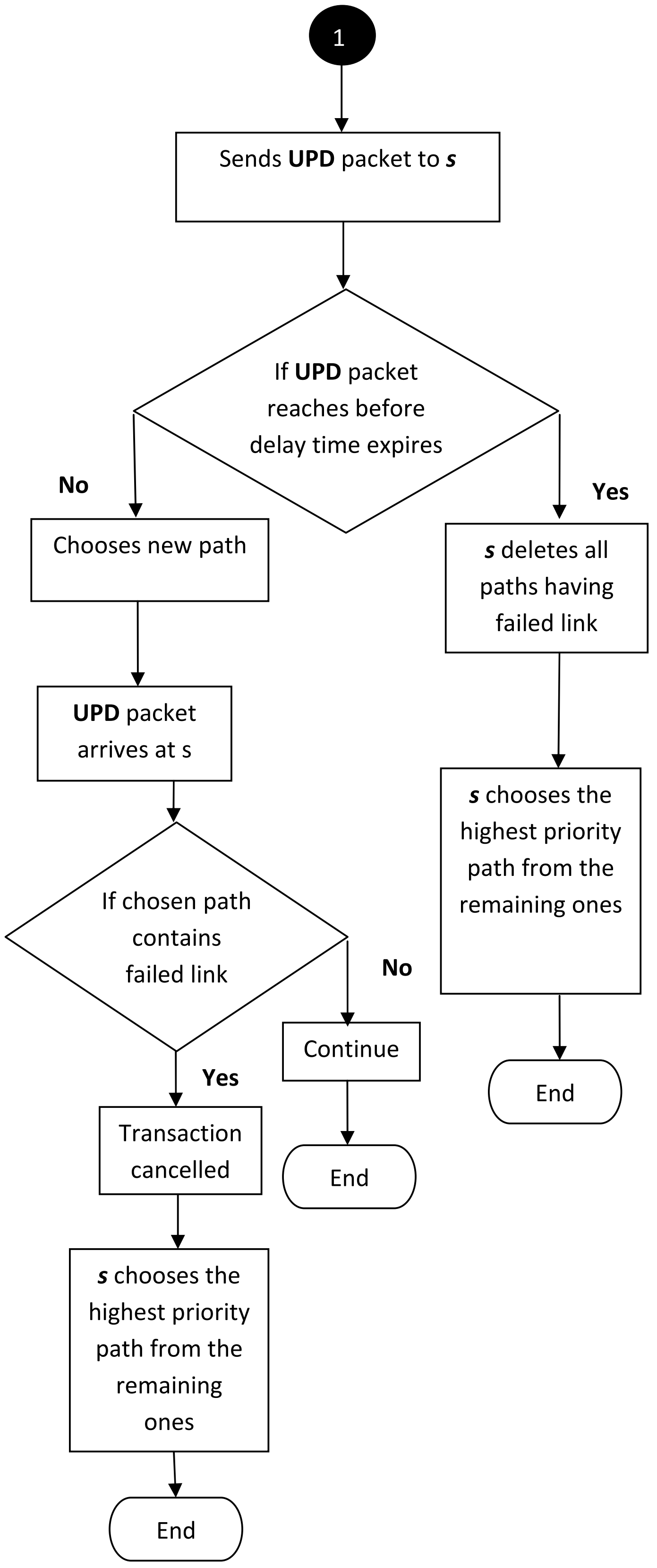
Dashed line between nodes ***f*** and ***e*** shows the the link ***f->e*** is broken.

Here, node ***f*** is ***i***, node ***e*** is ***j***, and node ***c*** is ***k***.

After link failure path cache of ***c*** is checked for presence of any path to ***d***. If found, then data packet is sent to ***d*** through that path and a UPD packet is sent to ***s*** for informing it about the new path and also of particular link failure. The source ***s*** will then delete all the paths containing the failed link. If not found then only UPD packet is sent to ***s*** informing it about the link failure. In this case again, if the UPD packet arrives at ***s*** before the delay time expires, the source will delete all the paths containing the failed link. Also, the path with maximum value of survivability factor will be chosen from the remaining ones. But if the delay time expires before the UPD packet arrives at ***s***, the source ***s*** will choose new path having highest value of survivability factor and send data packet through it. Now, when UPD packet arrives source ***s*** checks whether the new chosen path contains failed link or not. If not ***s*** continues to send packets through the selected path and also deletes all paths having the failed link present in it from its path cache. If yes then it cancels the current transaction and also deletes all paths that contain failed link. Then, it chooses the new path having the highest value of survivability factor.

3.7.2 Flowchart For Path Maintenance


Start
Link (*i->j*) failed
i sends data packet to *k*, where *k* is previous node of *i*
k checks for alternate path in its route cache for *d*
If alternate path found
Yes
No
Changes route map of the packet and sends **UPD** packet to *s*
1
UPD packet arrives at s
s deletes all paths that contain failed link and changes current working path
End


1
Sends **UPD** packet to ***s***
If **UPD** packet reaches before delay time expires
No
Yes
Chooses new path
s deletes all paths having failed link
UPD packet arrives at s
s chooses the highest priority path from the remaining ones
If chosen path contains failed link
No
Yes
Continue
End
Transaction cancelled
End
s chooses the highest priority path from the remaining ones
End

4. CONCLUSIONS

The algorithm proposed here finds multiple paths (if exist) from a source to destination. The above algorithm extracts merits from DSR, exploits it by augmenting additional features of alert packets and delay time. The intended path between any given pair shall be achieved in timely and efficient manner with a frugal usage of resources. A very useful data structure tree has been employed for the accomplishment of the set of objectives of the algorithm. Besides giving multiple paths it identifies the exact location of the link failure. Also as the paths are arranged in order of decreasing survivability factor (how much survivable or reliable the path is), the path with more value of survivability factor is chosen first (in case of link failure) i.e. the reliability of the chosen path will be more. So, performance will be good.

5. REFERENCES

[1] David B. Johnson. Routing in Ad Hoc Networks of Mobile Hosts. In *Proceedings of the IEEE Workshop on Mobile Computing Systems and Applications (WMCSA'94)* , pages 158–163, December 1994.
[2] David B. Johnson and David A. Maltz. Dynamic Source Routing in Ad HocWireless Networks. In *Mobile Computing*, edited by Tomasz Imielinski and Hank Korth, chapter 5, pages 153–181. Kluwer Academic Publishers, 1996.
[3] David B. Johnson , David A. Maltz , Yih - Chun Hu, and Jorjeta G. Jetcheva. The Dynamic Source Routing Protocol for Mobile Ad Hoc Networks. Internet-Draft, draft- ietf-manet-dsr-07.txt, February 2002.
[4] Josh Broch, David B. Johnson, and David A. Maltz. The Dynamic Source Routing Protocol for Mobile Ad Hoc Networks. Internet-Draft, draft-ietf-manet-dsr-03.txt, October 1999. Earlier revisions published June 1999, December1998, and March 1998.
[5] David B. Johnson. Routing in Ad Hoc Networks of Mobile Hosts. In *Proceedings of the IEEE Workshop on Mobile Computing Systems and Applications,* pages 158–163. IEEE Computer Society, December 1994.
[6] C.- K. Toh. A Novel Distributed Routing Protocol to Support Ad–Hoc Mobile Computing. In *Conference Proceedings of The 1996 IEEE Fifteenth Annual International Phoenix Conference on Computers and Communications*, pages 480–486, March 1996.
[7] Sunil Taneja and Ashwani Kush "A Survey Of Routing Protocols In MANET" published in IJIMT 2010.

AUTHORS PROFILE:

1. A.K. Daniel
Associate Professor
M.M.M. Engineering College,
Gorakhpur-273010,
Uttar Pradesh, INDIA

2. Shishir Dwivedi
B.Tech. Final Year CSE
M.M.M. Engineering College,
Gorakhpur-273010
Uttar Pradesh, INDIA

3. Tarun Verma
B.Tech. Final Year CSE
M.M.M. Engineering College,
Gorakhpur-273010,
Uttar Pradesh, INDIA

4. Pankaj Kumar Dubey
B.Tech. Final Year CSE
M.M.M. Engineering College,
Gorakhpur-273010,
Uttar Pradesh, INDIA