Published : IEEE-International Conference on Signal processing, Communications and Networking, Jan 4-6, 2008. Pp264-268

Building Reliable and Fault Resilient Mobile Ad Hoc Networks

Vipin M, Sankar K, Sarad A V
AU-KBC Research Centre,
M.I.T Campus of Anna University.
Chromepet, Chennai 600044, India.
E-mail: {vipintm, sankar, avsarad}@au-kbc.org

***Abstract*- The nodes in a mobile ad hoc network (MANET) have limited processing power, memory and transmission range. The mobile nodes may dynamically enter or leave the ad hoc network. To improve the resilience of the ad-hoc networks to mobility, node and link failure, we propose an algorithm involving binary trees and a special form of cycles in the network graph subject to constraints in topology and use this information to re-route packets from source to destination when interconnecting links or nodes fail.**

Keywords: Fault tolerance, Wireless Ad hoc networks.

I. INTRODUCTION

Fault resilience in multi hop networks, can be largely improved if each mobile node has extra information on its connectivity to every other node in the network. It is sometimes impractical and usually taxing for each node of a multi hop MANET to determine the entire topology due to the very nature of the ad hoc network. The tradeoff is, for each node to pre-determine and store fewer paths from itself to the destination node. The main idea of the algorithm presented is to create a binary tree using the nodes in the ad hoc network and to introduce a special class of cycles to improve fault resilience. The node represents the mobile communication device and the link indicates the connectivity between any two nodes in the network.

We consider that each node in the MANET is only aware of the existence of its immediate neighbors, i.e. each node is informed of other nodes exactly one hop from it.

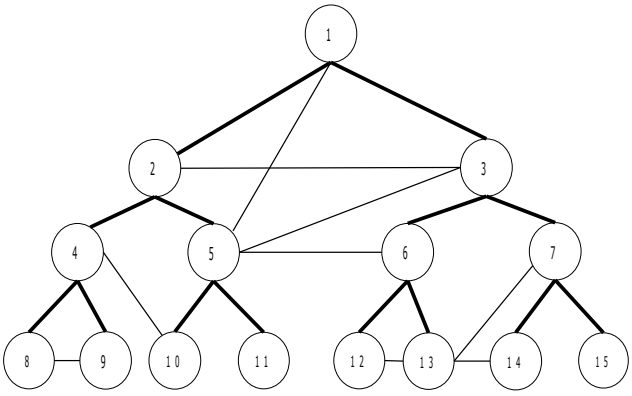


Fig. 1. A Binary Tree Formation

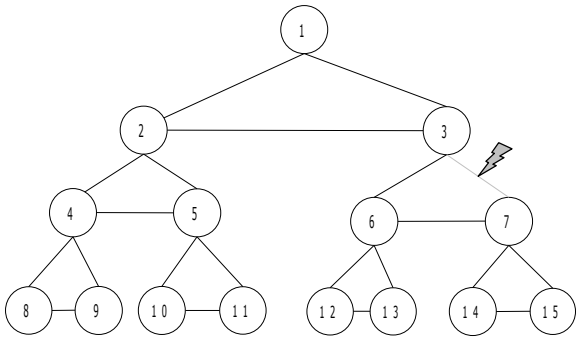
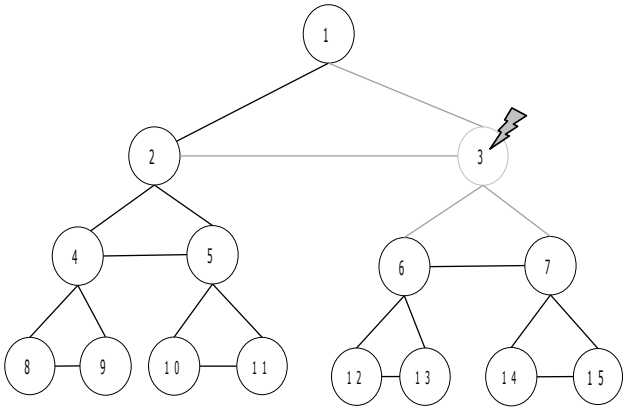


Fig. 2. Link between nodes 3 and node 7 fail

The life time of each node in the ad hoc network is limited by its power source. Careful consideration is to be given to maintain its power consumption at realistic levels. Thus, a check is to be kept on the volume of information required to be transmitted and the complexity of the processes run by the node. The node may also be memory constrained. Hence the algorithm running on the device as well as the transmission overheads between the nodes should be kept minimal. By the very nature of the ad hoc network, new nodes may enter and existing nodes may leave the network at any point of time. The limited transmission range of the node restricts its immediate connectivity to a few neighboring nodes. It might require multiple hops to reach from a source to the required destination in the network.



Published : IEEE-International Conference on Signal processing, Communications and Networking, Jan 4-6, 2008. Pp264-268

Fig. 3. Node 3 fails

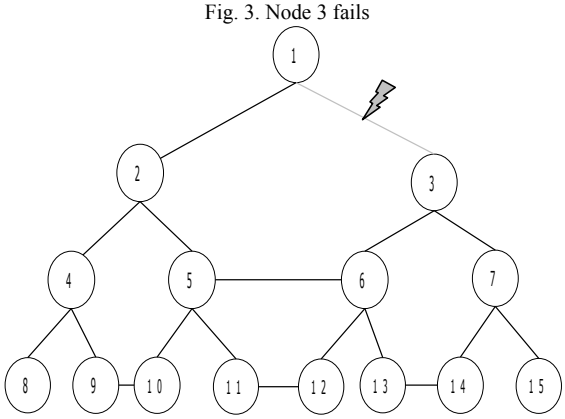


Fig. 4. Link between node 1 and node 3 fails

Let G= (V, E); be the graph where V denotes the set of nodes and E denotes the set of edges. Let v_1, e_1, v_2..., v_{i-1}, v_i, v_{i+1}, …, v_n be an open path of length n, the removal of any intermediate node v_i leaves the graph disconnected, with two disconnected open paths v_1, e_1, v_2...,v_{i-1} and v_{i+1}, …, v_n. We note that the removal of a node is a stronger condition than the removal of an edge, since it leaves all edges incident on the node disconnected. Let v_1, e_1, v_2..., v_{i-1}, v_i, v_{i+1}, …, v_n, e_n, v_1 be a cycle of length n+1. The removal of any intermediate node v_i still leaves the graph connected. Thus, cycles are more resilient to network faults when compared to open paths.

It is possible to run an all pair shortest path algorithm such as Bellman-Ford[1][2] to find the shortest path from every node to every other node in the network. Its distributed variant is still used for routing though it suffers from scalability issues and slow update to change in network topology which makes its use difficult in an ad hoc network, which requires being scalable and changes topology frequently. Link state routing protocols namely, the Open Shortest Path First (OSPF) protocol [3] makes use of Dijkstra's[4] algorithm, a single source shortest past algorithm and runs it separately for each node. It groups together networks into areas and hence reduces the number of nodes for which the algorithm is to be run in the local and global setting. Running OSPF over a MANET is still taxing by nature of the Dijkstra's algorithm from a fault resilience point of view. Link state routing protocols such as the Optimized Link State Routing protocol (OLSR)[5] can be taxing on nodes with low power and memory capabilities. Distance Vector Routing Protocols such as Ad-hoc On-demand Distance Vector (AODV) [6] are also slow in route establishment. We propose an algorithm for fault resilience, keeping in mind the issues and challenges surveyed in [7]. The algorithm we present is based on binary trees and the formation of a special class of cycles in the binary tree unlike source tree routing in [8].

II. BINARY TREE AND CYCLES

We introduce a binary tree based algorithm for MANET to improve fault resilience. The key idea is to construct a binary

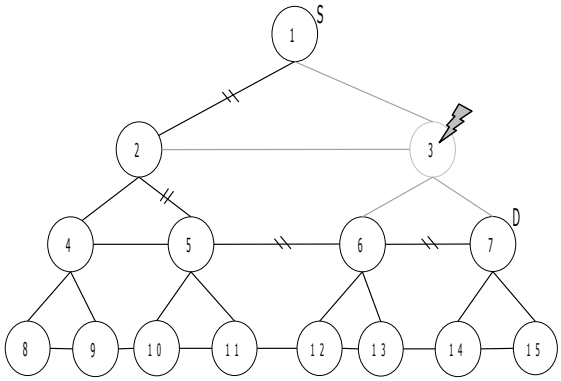


Fig. 5. Graph with nodes of Type 1 and Type 2

tree with the nodes in the network topology, as illustrated in Fig. 1. The removal of an edge from the binary tree results in the formation of two disjoint sub-trees. Each intermediate node in the sub-tree has a vertex degree 3. The removal of a node from the binary tree results in the formation of utmost three disjoint sub-trees where each of its internal nodes is of vertex degree 3. It is necessary that the communication network remains connected in the event of multiple links or nodes failing.

We introduce cycles in the binary tree by joining edges at the same tree level-namely Type 1 and Type 2 cycles. These cycles are introduced to improve connectivity with in the sub-tree and in between sub-trees. Fig. 2 and Fig. 3 illustrate Type 1 cycles and show its resilience to link and node failure. Fig. 2 shows a link failure between node 3 and node 7. We observe that the graph remains connected due to the link introduced between node 6 and node 7. Fig. 3 shows a failure at node 3. This results in disconnecting the graph into two disjoint components. To prevent this, we introduce Type 2 cycles. Fig. 4 illustrates Type 2 cycle and shows the failure of the link between nodes 1 and node 3. We observe that the network graph remains connected due to the link between nodes 5, 6 and nodes 11, 12.

Let us take up the case of node 3 failing in Fig. 5, which has cycles of both Type 1 and Type 2. Let node 1 be the packet source and node 3 be the packet destination. When node 3 fails, the packet is re-routed via 1→2→5→6→7 or through any connected path to the destination. The failure of node 3 still leaves the graph connected. The introduction of

Published : IEEE-International Conference on Signal processing, Communications and Networking, Jan 4-6, 2008. Pp264-268

the cycles reduces the number of hops from a source to destination when a link or node fails, whether it is down the graph or across the graph.

The root node of the binary tree is a randomly picked polling node in the network graph In Fig. 1, the root node is picked and two least weight edges are chosen and connected (when the nodes are in transmission range) with its neighborhood nodes. Any parent node always picks the two least weight edge connected nodes to expand the binary tree. Binary tree expansion proceeds in a breadth first search manner, where the tree is expanded breadth wise, to obtain the required binary tree from the graph. The binary tree is numbered in level wise ascending order.

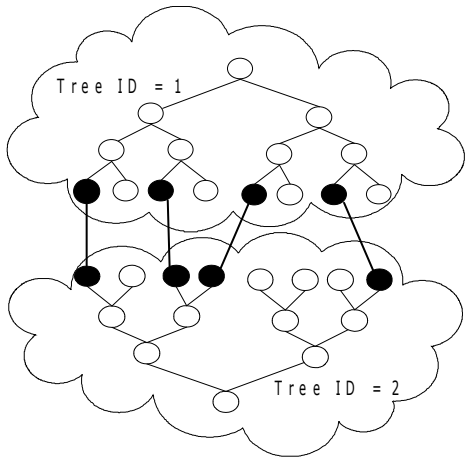


Fig. 6. Interconnecting trees

The node number at height 0 (root) is $[2^0]$, at height 1 is $[2^1,2^2)$, at height 2 is $[2^2,2^3)$ and at height h is $[2^h,2^{h+1})$. For any node numbered n, its left child is numbered $2.n$ and right child is numbered $2.n+1$. We can use this information to connect the nodes at the same level to get cycles of Type 1 and Type 2, as illustrated in Fig. 5.

III. THE TOPOLOGICAL DATABASE

Since each node in the binary tree is aware of its immediate neighbors, we make use of an adjacency matrix to propagate the required topology database, consisting of the binary tree and the cycles of Type 1 and Type 2, to all the nodes. If the nodes are numbered between 1 and n, an $n*n$ adjacency matrix of size n^2 bits is created. The links connecting the nodes in the graph are undirected. Hence it is sufficient to store the upper triangular matrix of the adjacency matrix alone. The size of the adjacency matrix now reduces to $n(n+1)/2 = (n^2 + n)/2$ bits. The use of adjacency matrix allows O(1) time retrieval of links between nodes.

Three passes of message passing is required for all nodes to get the required topological database. The first pass is called Binary_Enumeration, the second pass is called the Binary_Enumeration_Reply and the third pass is called the Binary_Broadcast.

In Binary_Enumeration, the height of the binary tree is fixed to be h. Enumeration starts from the root node (polling node). Each node informs the child nodes, their node number. Type 1 and Type 2 cycles are connected during Binary_Enumeration. Each node is aware of the node number it wants to connect to, to form Type 1 and Type 2 cycles. This is because nodes at height h are numbered $[2^h,2^{h+1})$. Each node searching for these cycles sends a broadcast packet to its immediate neighbor requesting a response in case it is the node number which the broadcast node is trying to connect to. The nodes at the same level wait for a minimum amount of time, so that they are communicated their node numbers by the parent nodes before they start broadcasting amongst its immediate neighbors looking for a particular node number.

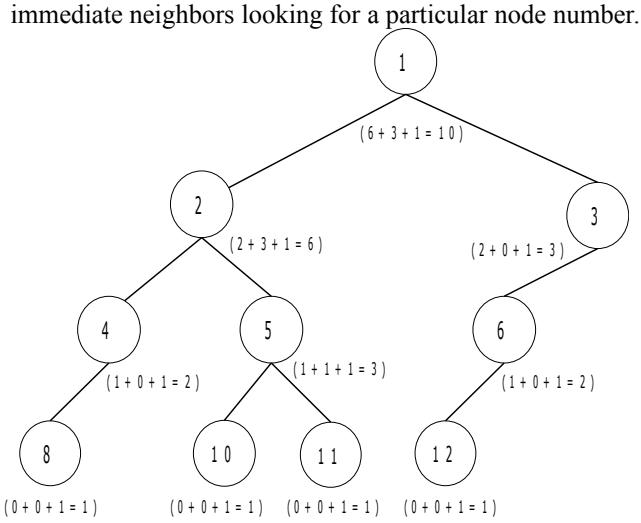


Fig. 7. A binary tree with node count

In Binary_Enumeration_Reply, the adjacency matrix of the nodes starting from the leaf level is updated with its parent and child nodes information, this process is repeated at each level and the adjacency matrix is send up the tree, all the way to the root node. Since the node at each level also knows the nodes to which Type 1 and Type 2 cycles are formed, this information is added to the adjacency matrix, before being sent to its parent node. The border nodes (Shaded in Fig. 6) of a given tree will also send the Tree ID of the other trees to which it is connected to, along with the adjacency matrix that is propagated up the tree. In Fig. 6, the border nodes with Tree ID=1 connect with the border nodes of Tree ID=2. When this procedure is done, the root's adjacency matrix holds the required topological database and the border node information for any given tree that it is connected to.

In Binary_Broadcast, the root's adjacency matrix along with the border node information is propagated down the tree to all its nodes. All the nodes in a given tree are now aware of

Published : IEEE-International Conference on Signal processing, Communications and Networking, Jan 4-6, 2008. Pp264-268

all the border nodes for another given tree. The border node information propagated during Binary_Broadcast can now be used to find a suitable path to the desired tree.

Since the binary tree may not be complete, node numbers may be missing, as illustrated in Fig. 7, by nature of the numbering used for connecting Type 1 and Type 2 cycles. The size of the adjacency matrix grows exponential with increase in height of the tree. If the binary tree height is restricted to say h=8, the nodes are numbered in between 1 to 255. It takes a little over 4 KB to store the adjacency matrix, taking the upper triangular matrix alone. One way is to restrict the height of the tree is to create a number of disjoint binary trees of small fixed height, each with a different Tree Identity (Tree ID) and also interconnect between nodes with different Tree ID. Another possible way is to efficiently reduce the size of the adjacency matrix with a five pass message passing. Let n be the number of nodes in the binary tree, whose intermediate nodes may or may not have two child nodes. Then, the size of the adjacency matrix can be reduced to $(n^2+ n)/2$ bits.

This is done by a second numbering of the nodes in the binary tree. Fig. 7 illustrates the procedure. The nodes have three addends-the first addend is the count of the number of nodes in its left sub-tree, the second addend is the count of the number of nodes in its right sub-tree and the third addend is the count of the center node, which is always 1. Counting starts from the leaf nodes. For e.g. let us start with node 8. It has 0 nodes in the left sub-tree, 0 nodes in the right sub-tree and 1 center node. This information is propagated to its parent node. Since the left sub-tree to node 4 is at node 8 and node 4 has no right sub-tree, the node count for node 4 becomes (1+0+1=2). Similarly node 10 and node 11 have a node count of 1 and node 5 have a node count of 3. For node 2, the left sub-tree is at node 4 and right sub-tree is at node 5. Hence node 2 gets a node count of (2+3+1) =6. Similarly node 12 gets a node count of 1, node 6 gets a node count of 2 and node 3 gets a node count of 3. Hence the node count of node 1 becomes (6+3+1). Now node 1, the root node knows that it has 6 nodes to its left and 3 nodes its right. Hence the numbering [2, 7] is reserved for its left sub-tree and [8, 10] is reserved for its right sub-tree. The left child of node 1 is numbered 2 and the right child is numbered 8. Node 2 knows it has 2 nodes to its left and 3 nodes to its right. Hence the numbering [3, 4] is reserved for its left sub-tree and [5, 7] is reserved for its right sub-tree. The same procedure is recursively followed for the left and right sub-trees. The nodes in Fig. 7 numbered 1, 2, 4, 8, 5, 10, 11, 3, 6, 12 gets new node numbers 1, 2, 3, 4, 5, 6, 7, 8, 9 and 10 respectively. I.e. when the procedure terminates the n nodes in the binary tree is numbered from 1 to n.

A five pass messaging may be used to deriving the required topological database, incase we do not want to restrict the height of the tree to a small value and yet do so in a memory efficient way, from the size of the adjacency matrix point of view. This is however done at a cost of two extra passes when compared to three pass messaging described earlier. The first pass of message passing is named as Binary_Join_Cycle, the second pass of message passing as Binary_Node_Count, the third pass of message passing as Binary_Second_Numbering, the fourth pass as Binary_Consolidate_Toplogy and the fifth pass as Binary_Broadcast_Topology. In Binary_Join_Cycle, the height of the binary tree is fixed at a desired height h. Enumeration starts from the root node. Each node informs its child nodes, their node number. For any node numbered n, its children are numbered 2.n and 2.n+1. Type 1 and Type 2 cycles are connected during Binary_Join_Cycle and this is done in the same manner it is done in Binary_Enumeration of three pass messaging. In Binary_Node_Count, the leaves initiate the node counting as illustrated in Fig. 7. In Binary_Second_Numbering, the nodes are given a second numbering from 1 to n. In Binary_Consolidate_Toplogy, each node starting from the leaf level put its parent and child nodes information into the adjacency matrix and passes it on to its parent node. Since the node at each level knows the nodes to which Type 1 and Type 2 cycles are formed, it can simply request the other node for its new node number and update this information in the adjacency matrix. The border node of a given tree will also send the Tree ID of the other trees to which it is connected to, along with the adjacency matrix that is send up the tree. In Binary_Broadcast_Topology, the adjacency matrix of the root node holding the required topology database with respect to the second numbering along with border node routing information is propagated down the tree to all its nodes.

	a	b	c	d	e
a	0	1	1	0	0
b	1	0	0	0	1
c	1	0	0	1	0
d	0	0	1	0	1
e	0	1	0	1	0

Fig. 8. Adjacency matrix of a graph

Since utmost five entries in each row of the adjacency matrix is 1 and the remaining are all 0's, a simple and efficient algorithm such as Run Length Encoding or a sparse matrix compression algorithm such as in [9] may be used for compressing the adjacency matrix. Any node can use the adjacency matrix to look up all existing paths that are joined by the binary tree and the cycles associated with it. If the cost metric had been propagated as a cost matrix in the same manner the adjacency matrix was obtained, it would also be possible to calculate the cost from any node to any other node in the tree depending on the path chosen, within the premise of the binary tree and its associated cycles.

Published : IEEE-International Conference on Signal processing, Communications and Networking, Jan 4-6, 2008. Pp264-268

For e.g. Fig.8 illustrates the adjacency matrix of a graph with 5 nodes- namely *a*, *b*, *c*, *d* and *e*. To find a path from node *a* to node *d*, the adjacency matrix is looked up and the immediate neighbors of node *a* is found to be node *b* and node *c*. Node *b*'s immediate neighbor is node *e* and node *e*'s immediate neighbor is node *d*. The path *a*→*b*→*e*→*d* is found. Similarly, node *c*'s immediate neighbor is node *d* and the path *a*→*c*→*d* is found. The algorithm proposed can be used to re-route packets either when there are link or node failures.

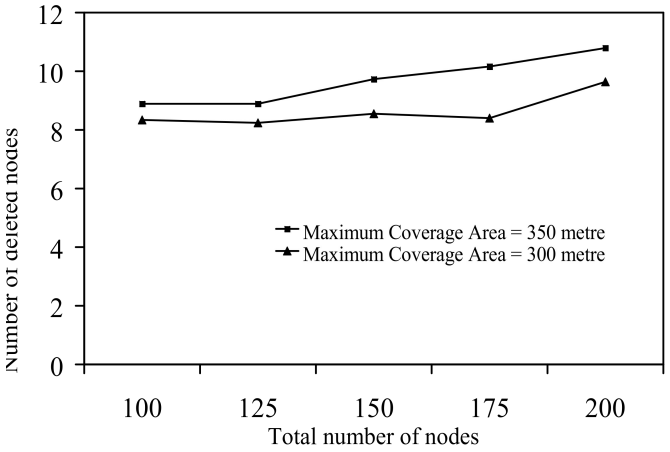


Fig. 9. Total number of nodes vs. Number of deleted nodes

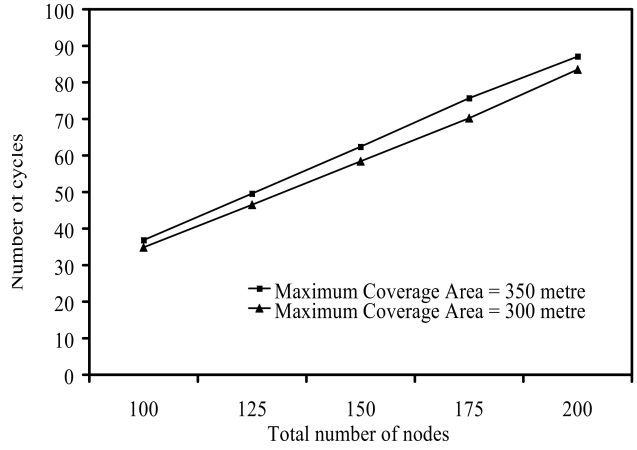


Fig. 10. Total number of nodes vs. Number of cycles

It may also be initiated even otherwise but routing cost may not be optimal even though two least weight edges are used to find the child nodes when the binary tree is build. When an orphaned node associates itself with a node that is already a part of the tree, the node that is part of the tree can act as a proxy to the orphaned node until the tree is re-enumerated. Each time a node becomes unreachable, this information is broadcast to all other nodes, similar to route maintenance in [10]. When a specified number of nodes are unreachable, the tree is re-enumerated.

IV. EXPERIMENTAL RESULTS

The simulation test bed was setup. A good pseudo random generator[11], with uniform distribution properties was chosen. A square area of required size was picked and a given number of random ad hoc nodes were plotted within the area. Each node was given a power range between *power_minimum* and *power_maximum*. The power range is a function of distance in the square area. Each node is designated a random power with in the power range. A binary tree was enumerated; cycles of Type 1 and Type 2 were connected using nodes in each others power range.

TABLE I
Number of nodes deleted vs. Number of nodes unreachable

Number of nodes deleted for a sample of 100 nodes	Number of nodes deleted for a sample of 200 nodes	Number of Unreachable nodes
20	24	5
41	45	10
57	62	15
59	67	17
-	72	18
-	76	20
-	75	22
-	-	23

The parameters chosen for simulation are based on [12]. All experiments are done for a simulated area of 1 square kilometer. Each node was pseudo randomly assigned a power range (transmission range) with *power_minimum*=50 metre. The maximum coverage area described in the plot is the value of *power_maximum*. The plot in Fig. 9 gives the count of average number of nodes required to be deleted randomly, so that 2 or more nodes in the connected network becomes unreachable. The experiment was done for an ad hoc network of 100, 125, 150, 175 and 200 nodes. The plot given is the average for 100 trials.

Fig. 10 is a plot of the total number of nodes vs. total number of cycles (Type 1 and Type 2 cycles), with *power_minimum*=50 metre. We notice that the number of cycles increases with increase in total number of nodes under experiment. The plot given is the average for 100 trials.

TABLE I gives the average number of nodes required to be deleted so that a given number of nodes in the network is unreachable. The value of *power_minimum*=50 metre and the value of *power_maximum*=350 metre. The first entry in the table states that, for a sample of 100 nodes in an ad hoc network, it requires an average of 20 random nodes to be deleted to get an average of 5 unreachable nodes in the network. A similar entry for an ad hoc network with 200

Published : IEEE-International Conference on Signal processing, Communications and Networking, Jan 4-6, 2008. Pp264-268

nodes is also given. The experimental output is the average of 100 trials. For the 100 node ad hoc network considered, the deletion of more than an average of 17 random nodes leaves most of the network unreachable. This is the point where the network connectivity begins to deteriorate rapidly. Similarly, for the 200 node ad hoc network considered, the network connectivity deteriorates rapidly when an average of more than 22 random nodes is deleted.

V. CONCLUSION

We proposed a binary tree based algorithm involving a special class of cycles to improve fault resilience taking into consideration the properties of ad hoc networks. The experimental results carried out show good resilience to network failures.

ACKNOWLEDGMENT

The suggestions and comments of T Selvam and K Vijayalakshmi were of considerable assistance.

REFERENCES

[1] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms*, Second Edition. MIT Press and McGraw-Hill, 2001.

[2] Aaron Kershenbaum. *Telecommunication Network Design Algorithms*, First Edition. McGraw-Hill, Inc., 1993.

[3] Moy, J, OSPF Version 2. Internet Request For Comment 1247(1991).

[4] Michael T. Goodrich, Roberto Tamassia. *Algorithm Design: Foundations, Analysis, and Internet Examples*. Wiley, 2001.

[5] T. Clausen, P. Jacquet, Optimized Link State Routing Protocol (OLSR). Internet Request for Comment 3626(2003).

[6] C. Perkins, E. Belding-Royer, S. Das, Ad hoc On-Demand Distance Vector (AODV) Routing. Internet Request for Comment 3561(2003).

[7] S. Mueller, Ghosal, "Multipath routing in mobile ad hoc networks: Issues and challenges", Invited paper in *Lecture Notes in Computer Science*, Edited by Maria Carla Calzarossa and Erol Gelenbe, 2004.

[8] J.J. Garcia-Luna-Aceves and M. Spohn, "Source-tree routing in wireless networks". In *Proc. IEEE ICNP 99*, 7th International Conference on Network Protocols, Toronto, Canada, 1999.

[9] Bruce J. McKenzie, Timothy C. Bell, "Compression of sparse matrices by blocked rice coding*". IEEE Transactions on Information Theory*, Volume 47, Issue: 3: 1223-1230 (2001)

[10] D.B. Johnson, "Routing in ad hoc networks of mobile hosts". In Proc. of the Workshop on Mobile Computing Systems and Applications, pages 158-163, 1994.

[11] M. Matsumoto, T. Nishimura, "Mersenne twister: a 623-dimensionally equidistributed uniform pseudorandom number generator", *ACM Transactions on Modeling and Computer Simulation*, Volume 8, Issue 1, January 1998.

[12] Valeriy Naoumov, Thomas R. Gros, "Simulation of large ad hoc networks". MSWiM 2003: 50-57