NOVEL METHOD FOR DESIGNING SELF-REPAIRING PROTECTION TREES IN MESH NETWORKS

G.V.N.Kishore¹, K. Devi Priya²

¹M.Tech student, Department of C S E , Aditya Engineering College, author1@email.com
²Assistant Professor, Department of C S E , Aditya Engineering College, A.P.India, devipriya.kamparapu@aec.edu.in

Abstract

Protection trees have been used in the past for restoring multicast and unicast traffic in networks in various failure scenarios. In this paper we focus on shared self-repairing trees for link protection in unicast mesh networks. Shared protection trees have been proposed as a relatively simple approach that is easy to reconfigure and could provide sub-second restoration times with sub-optimal redundancy requirement. The self-repairing nature of this class of protection trees may make them an attractive option for cases where dynamic changes in network topology or demand may occur. In this paper, we present heuristic algorithms to design a self-repairing protection tree for a given network. We study the restorability performance of shared trees and examine the limitations of such schemes in specific topologies, such as cases where long node chains exist. Using extensive simulations with thousands of randomly generated network graphs. We compare redundancy and average backup path length of shared protection trees with optimal tree designs and non tree designs. We also apply our algorithms to the problem of designing the protection tree in a pre-designed fixed-capacity network, and study the performance of shared protection trees in this scenario under different network loads and link utilization levels.

Index Terms: Mesh Tree, Node, Protection, Source

1. BACKGROUND

Network failure recovery has been an important subject of research in the field of network design and service reliability for more than two decades. The large volume of traffic (data, voice, video on demand, etc.) carried by backbone networks draws special attention to the issue of network recovery and protection against node and link failures, because interruption of such huge traffic flow (and, consequently, the offered user services) could cripple businesses and cost millions of dollars. Fast restoration of traffic after failure is essential, whether the failure is caused by a fiber cut, node failure or higher layer service point failure. At the physical layer ring-shaped designs for optical backbone networks, e.g. SONET UPSR and BLSR rings [1] are used commonly. However more recently, particular attention has been paid to mesh networks, by which we refer to networks in which at least one node is connected to three or more other nodes. Mesh networks in particular address the scalability issues of ring-based architectures, because in mesh networks links and nodes can be added or upgraded solely based on traffic demand without imposing a certain physical topology. Failure recovery schemes in mesh networks are generally categorized as Path restoration schemes and Link protection schemes [2]. In path restoration schemes, failed connections are restored individually by their source nodes through new end-to-end routes. In link protection schemes, an alternative local path between the end nodes of the failed link is found through the network, and all connections on the failed link are switched in bundle to the local detour. The term link is used in a broad sense here; it could refer to a multi-fiber span, a single fiber, a single wavelength on a fiber, or even a higher layer logical connection. As such, different wavelengths on a fiber may be re-routed on different paths. Path and link protection schemes could use dedicated spare capacity for each backup path, or share the spare capacity on each link among all backup paths that traverse it.

In general, link protection could potentially provide faster recovery service than path restoration because there is no need to inform the source node of each individual connection, or to re-compute the end-to-end path from the source node. This factor could become even more important in backbone networks where each fiber might carry thousands of connections between different source-destination pairs. On the other hand, studies have shown that end-to-end path restoration could provide more capacity efficiency and reduce the required redundancy in the network [3]. In practice, link
protection schemes are preferable for quick restoration of physical layer communication in backbone networks, while path restorations can be deployed at the internetworking layer of the network.

1.1 Related work

We focus our attention on those studies that are more relevant to the subject of this paper, namely the design of tree-based link protection schemes. Therefore, this review does not cover huge amount of prior research on path restoration algorithms. Specifically, we briefly review mesh survivable design techniques, ring and cycle-based techniques, and then discuss applications of unicast trees in network service restoration and dynamic reconfiguration.

Research on mesh survivability schemes has been conducted for about two decades by now. At first, such research efforts focused on using network digital cross connects to re-route the connections of a failed link over k shortest-paths either from the source node or between the end nodes of the failed link. Distributed protocols such as the Self Healing Network (SHN) [4] have been proposed to eliminate the need to maintain a central link-state database. Such approaches allowed sharing of spare capacities for backup paths of different links, essentially assuming that the probability of two concurrent link failures was low. The problem of optimizing backup paths to minimize redundancy (often referred as Spare Capacity Assignment – (SCA) has been formulated as an Integer Linear Programming (ILP) in several studies under different constraints [5–9]. For the SCA problem it is assumed that the network topology and the working capacity of each link are known, and then the shared spare capacity is optimized by finding the best backup paths for each unit of working capacity. SCA optimization formulations are NP-complete and thus difficult to solve except for small networks. However, such optimal solutions are often used as yardsticks for evaluating the performance of equivalent heuristic methods.

More recent proposals have employed cycle-based protection in which backup paths for network links are arranged on one or more graph cycles [10–13]. This approach would provide certain advantages, such as easier migration from ring-based SONET to a mesh network, as well as faster restoration speed if the cycles are pre-connected.

The equivalent SCA design problem for protection cycles has been formulated as an ILP with cycle enumeration (pre-processing a list of potential cycle candidates) [14]. This formulation adds a new constraint to the unrestricted optimal mesh survivable design; that the backup paths should be selected from links on a pre-selected group of cycles. As a result, the redundancy requirement of a cycle-based design tend be higher than the optimal mesh design. The ILP approach to cycle design is also NP Complete, with exponential computational times reported in [14]. Heuristic design algorithms were proposed in [15,16] to reduce computational times by limiting the number of cycle candidates at the cost of higher redundancy.

The high computational complexity of designing optimal mesh and cycle protection schemes has generated some interest in alternatives with less computational complexity and with simpler manageability; i.e. the ability to reconfigure, scale, maintain and regionalize network wide protection scheme with ease while still providing sub-second restoration speed with sub-optimal redundancy requirements. Network trees provide an alternative option for pre-planning backup paths with reasonable computational complexity. Trees are local by nature and changes on one branch have limited impact on other branches and higher layers in the hierarchy. This fact also provides simpler handling of multiple-link failure or node failures with tree-based protection schemes. A number of telecommunication protocols for constructing spanning trees already exists in various layers of today’s networks and can be modified for construction of protection trees. Network state information and databases for tree-based algorithms have already been developed and deployed in the networks.

It is easy to grow, modify, add branches or repair a spanning tree using common protocols currently deployed in the mesh networks. Furthermore when static vs. re-configurable networks are being considered, shared protection trees can be managed in a self-repairing manner, where a disconnected node can reconnect itself to the protection structure using primarily local state information and without the need to recompute the complete set of backup paths for each link. Such computation can be done in advance and stored in the node so that no routing computation would be necessary after failure. On the other hand, it is expected that shared-capacity unicast protection trees in general may require more network redundancy than an equivalent shared-capacity optimal mesh or cycle-based designs. One reason is that a tree provides one backup path for each link while a cycle could provide two backup paths (clockwise and counter clockwise) for each non-cycle link and thus more sharing of spare capacity is possible [12].

Spanning trees have been used widely for routing and protection of traffic in telecommunication networks, for instance, for maintaining connectivity between bridges that connect subnets [17], and in ATM multicast path restoration [18,19], where node or edge-disjoint spanning trees could be used for providing the main and backup routes for multicast traffic from source to sink nodes. Core-based shared tree [20] has also been used for shared protection of multicast traffic.
Multicast networks primarily use spanning trees for maintaining reachability among nodes in the network. However in the case of unicast link protection the problem is more complicated as traffic patterns are bidirectional between each pair of nodes, as opposed to multicast traffic to/from one source node. In this research we were interested in a class of shared protection trees, proposed in [22], that are used for link protection in a unicast network. In this approach, backup paths for network links are provided by a pre-computed shared-capacity protection tree.

The underlying method is based on assigning one primary parent node and one or more backup parent nodes for each network node, in a way that the backup paths for network links are organized in a spanning tree structure. This approach provides a reasonably fast restoration time, because aside from typical cross-connect switching along the pre-determined path, no additional time is necessary for exchange of failure information, routing table update or making routing decision after failure.

To understand the mechanism of self-repairing shared protection tree, consider Fig. 1a. Here a spanning tree is constructed in a mesh network. Tree links are shown in thick lines, and non-tree links are shown in dashed lines. The tree provides the backup paths in the network by assigning primary parent nodes to each node; for example, node C is the primary parent node for nodes E and F etc. The traffic on link EF can be protected via backup path CE–CF, and traffic on the link BF is protected via backup path AB–AC–CE. Now suppose the tree link CF fails (Fig. 1b). Node F has lost its connection to its primary parent on the protection tree.

![Fig-1 : Dynamic reconfiguration of protection tree.](image)

It now has to switch to its predefined backup parent node, for instance node B. The tree has thus reconfigured itself and the traffic on link CF is re-routed through BF–AB–AC. This dynamic reconfiguration also effectively updates the backup path for link EF, now to BF–AB–AC–CE. The network links must be capacitated properly to have enough protection capacity for rerouting of the traffic of those links that they protect.

The self-repairing spanning tree can reconfigure itself quickly in case of topology changes such as addition of new links or nodes, or multiple links and node failures. In particular, a detailed study of the performance of self repairing spanning tree in multiple-failure scenarios, (including comparison with the performance of other schemes in such scenarios) has been presented in [26].

The results show that with its operational simplicity, the self-repairing tree is able to provide a viable alternative to other proposals, in particular based on the fact that the complexity of its operation does not increase with the number of failures while optimal mesh or cycle-based approaches typically require re-calculation of backup paths to keep up with multiple simultaneous or subsequent failures.

The study in [21] suggests that proper selection of the root node and placement of the links with large capacities higher up in the tree structure -resulting in a hierarchical capacity protection tree- could improve the performance of this restoration scheme. Restorability of this scheme depends on the existence of primary and backup parent nodes for every network node, which is not necessarily guaranteed even if the network is two-edge connected.

The reason lies in the self-repairing characteristic of this tree-based scheme. In case of a tree link failure which essentially disconnects a node from its primary parent node on the tree, the downstream node must repair the protection tree through an adjacent backup parent node for quick restoration time. However, there are cases where such backup parent nodes may not exist, for instance in a node chain. The worst case is a ring topology, in which the restorability of a self-repairing unicast tree could approach as low as 1/(N – 1), in which N is the total number of nodes in the ring. An enhancement to backup parent selection was proposed in [22] to guarantee 100% restorability by allowing non-adjacent backup parent nodes; however the modified scheme was no longer self-repairing, thus unable to handle multiple failures within the tree.

We proposed dynamic shortest backup path calculation for handling multiple failures, which prolongs the restoration time. In Section 4 we will show the impact of chain topologies on the overall network restorability of self-repairing trees and
examine the cases where only partial restorability could be achieved by the self-repairing shared tree.

The main objective of this paper is to propose design methods to construct the self-repairing shared-capacity tree in a given mesh network. While the general concept has been described in [21] and the optimization problem has been addressed in [22], capacity efficient design heuristics that could be applied to large networks had not been studied before. Among the contributions of this paper, we show that a triangular-based tree could reduce the required spare capacity on the tree. While the idea of triangular trees had been briefly explored in [23] to show that it reduced average backup path length, it was applied to design of capacitated self-repairing trees. We present heuristic algorithms to choose the primary and backup parent nodes of each node. We propose and compare several tree growing approaches (presented as node placement criteria) and show in which scenarios each could be used. Another contribution of this paper is the study of the restorability limitation of such protection trees. We study the design of self-repairing protection trees in both Spare Capacity Allocation (SCA) and Fixed Capacity scenarios. We also offer simulation results for spare capacity assignment problem and compute the performance of the protection scheme in terms of restorability, redundancy and average backup path length, as well as exploring the limitations of the self-repairing protection trees.

2. MODELS AND ASSUMPTION

The literature for study of protection schemes: the network will be two-edge connected (otherwise full network restorability could not be achieved); the network will be represented by a connected graph with N nodes and M links and average nodal degree of D; we also assume that parallel physical links between two nodes have been consolidated into a single logical link. Each network would be designed with minimum spare capacity required to achieve maximum single failure restorability.

Some general characteristics of our algorithm are as following:
1. The nodes with nodal degree of two should be placed preferably either at the root or the leaves of the tree. The rationale is that otherwise, an intermediate node with one primary parent node and one child node will be left with no adjacent edge to a potential backup parent node.

2. The tree edges are chosen in a manner that each node remains adjacent to at least one non-tree edge, if possible. This is to facilitate the selection of backup parent nodes for protection of tree links.

3. Our tree construction algorithm grows the tree by adding nodes to it in an ordered way decided by pre-determined sorting criteria that we call node placement method.

The tree design algorithm is described in the following:

Protection Tree Primary Parent Selection Algorithm

(1) Create Vector of Nodes V, empty tree node vector U and empty tree edge vector T.

(2) Sort V based on node placement criteria

(3) U(1) = V(1) (Add the root node vr to the tree node vector)

(4) Repeat the following loop of steps until all nodes are added to the tree node vector U

(4.1) Going from top to bottom in V, choose the first node vi that is not a member of tree node vector U

(4.2) Create vector L of all members of the vector V that are in U and are connected to vi with non-tree links. L is thus a subset of V and follows the same sorted order of nodes.

(4.3) Going from top to bottom in L, choose the first node lk that has at least two non-tree links adjacent to it, unless:

- lk is the root node of the tree, or
- no node with the above condition exist in the tree, AND all other remaining (non-tree) nodes have a nodal degree of two.

In each of the above cases, choose the first entry lk in L.

(4.4) if an lk was found in the previous step:

- designate lk as primary parent of vi.
- add (lk, vi) to T.

(4.5) if reached the bottom of the node vector, reset the pointer to the top of the node vector.

(4.6) Return to Step 4.

#Backup Parent Selection Algorithm

(5) For each node vi without backup parent, do:

(5.1) Sort all neighbours based on their tree distances

(5.2) Exclude neighbours that are in the subtree of vr.

(5.3) Choose the neighbour that:

- Its connecting edge is not in the protection tree
- Does not have a backup parent node
- Is not the root

(5.4) If such neighbour is found, assign it as backup parent of node x, and assign node x as backup parent of this neighbour.
(5.5) If not found, choose the neighbor with minimum distance that its connecting edge is not in the protection tree. Assign it as the backup parent node of node x.

(5.6) End of the loop.

The first part, the primary parent node selection algorithm, basically constructs the spanning tree in the network. The second part, the backup parent node selection algorithm, chooses those non-tree links that will be used for tree repair in case of a tree link failure. The tree node placement criteria (Line 2 in the above algorithm) is used for sorting the nodes of the network in the order that they will be added to the tree. The top node on the list would become the root of the tree. Note that in the above algorithm a node cannot be added to the tree if none of its neighbors are in the tree yet. A possible scenario here is a node whose rank in the node placement criteria is higher than all its neighbors. We call such nodes degraded nodes. The algorithm skips such nodes in each sweep of the node vector first, and once the top-down sweep of the node vector is completed, the algorithm resets the node pointer back to the top of the vector to check for degraded nodes. This loop continues until all nodes are added to the tree.

We consider two different criteria here: One capacity based methods (Average Adjacent Capacity (AAC), and one topological method (nodal degree (ND). The Nodal Degree or distance from center criteria would attempt to create a tree with shorter backup paths and more capacity sharing. The capacity-based methods would attempt to improve the restorability by placing the large capacity nodes higher up in the tree hierarchy.

In discussing the complexity of the node placement criteria in the following, we assume that a typical heap sorting algorithm with maximum complexity of \(O(N \log N)\) is used where applicable.

2.1. Nodal Degree (ND)

In this method the nodes in the protection tree are sorted in descending order based on the nodal degree of the nodes. Therefore, the tree becomes wider rather than deeper, and restoration paths on the tree will be shorter. The ND method requires computation of nodal degrees for each node and then sorting the vector of nodes accordingly, and has a complexity of \(O(N^2 + N \log N) = O(M + N \log N)\).

2.2. Average Adjacent Capacity (AAC)

We define the AAC parameter as the mean value of link capacity on the adjacent links connected to each node. Clearly, node placement based on AAC criteria favors nodes that have no low capacity link adjacent to them. For a predesigned network with fixed capacity, this method provides a fairly good chance to ensure existence of sufficient spare capacity for restoration in pre-designed scenarios by pushing up nodes that have a higher average capacity rather than those with one very high capacity link and several low capacity ones. The AAC node placement algorithm requires a further comparison of nodal degrees to insure that those with nodal degree of two would be placed either at root or leaf nodes, if possible. However because the computation of average adjacent capacity has the same order of complexity as nodal degree, the total computational complexity of the AAC node placement algorithm will also be \(O(M + N \log N)\).
Implementing protection trees in fixed-capacity networks In this scenario we examine how a protection tree could be implemented in a pre-designed fixed-capacity network to achieve maximum restorability. In this scenario the network under study might be an operating network that may not have any specific protection scheme in place or may have been using a different scheme than a protection tree.

We then have to determine how we could organize the available spare capacity of the network into a protection tree. Obviously the objective of the protection tree design algorithm would be different from the capacity assignment scenario in Section 2. While in that case we were trying to minimize the required spare capacity that had to be assigned in the network, in this case we would like to maximize the restorability that we could achieve using the available spare capacity of the network.

Total network restorability could be limited by several factors, for instance network load, availability and distribution of spare capacity, and effectiveness of the algorithm in utilizing spare capacity to provide backup paths. For example, total network load might be moderate enough that would leave sufficient spare capacity in the fixed-capacity network, but that spare capacity might not be present at locations where we need it. Alternatively, the fixed-capacity network may have enough spare capacity for protection should an overall optimization problem is solved, but a heuristic algorithm that focuses on local failure recovery might be less efficient in utilizing the available spare capacity and thus might fail to provide full restorability.

All these factors would have an impact on the restorability of the network. In order to select the protection tree in a fixed-capacity network, we make some minor adjustment to the design algorithm. Here in capacity-based node placement methods (AAC and MAC), we use the amount of protection capacity on network links for node sorting, not the working capacity. The purpose here is to maximize the use of spare capacity through highly shared nodes.

Also in selection of primary parent node, instead of connecting each node to its highest ranked neighbour in the sorted vector of nodes (as we did in Section 3), here we choose the upstream node of the adjacent link with the largest spare capacity as the primary parent of this node. This arrangement increases the possibility that those links with larger spare capacities end up on higher branches of the protection tree.

Another difference is in assigning the backup parent nodes. While pair-wise triangles as would provide better capacity efficiency for the spare capacity assignment problem, in the fixed-capacity case the spare capacity is already determined and the objective is to increase restorability, which the triangular sub tree approach would not necessarily provide as it does not take the available spare capacity of the links into account. Therefore in the fixed capacity scenario, our algorithm chooses as backup parent of each node the upstream node of the link with the largest spare capacity from the set of eligible adjacent links of this node. All previous requirements about the backup parent node not being a child or a primary parent of this node, still hold.

Fig-4 : Mesh Formation Using AAC Method

3. PERFORMANCE EVALUATION

We studied the performance of the protection tree in fixed-capacity networks using random networks scenarios. The graphs were generated based on the same algorithm as given below. In order to narrow down the number of parameters in this problem, we limited our study to those graphs where the unlimited spare capacity assignment with protection tree design would have provided full restorability. In order to model the fixed capacity scenario, the total capacity of each link was set to the OC-1/STS-1 size. For populating the working capacity of the links we generated DS-1 demands between randomly selected pairs of nodes in the network, and routed them based on two parameters: maxLinkLoad, the maximum allowed load of a link, and networkLoad, the ratio of total working capacity to total link capacity of the whole network. In order to get a realistic model of the dynamic operation of the network, a routing algorithm was implemented to route individual demands on the minimum weight shortest available path as long as the ratio of working to total capacity on each segment of the path remained...
4. CONCLUSION

We presented an algorithm to design self-repairing hierarchical protection trees for protection of unicast traffic in mesh networks. Such protection trees provide the flexibility of simple, dynamic reconfiguration for sub-second restoration while providing a sub-optimal redundancy requirement. We studied different node placement methods for construction of the tree and showed the benefits and drawbacks of each method in terms of complexity and restorability performance in different scenarios. We computed the complexity of our design heuristics and showed that it had scalable execution time.

Our simulation results with more than 12000 random graphs showed that the scheme could provide full restorability in over 99% of the cases if no 3-node chain existed in the network. However when node chains exist, only partial restorability could be achieved. The restorability performance of the algorithm was particularly poor in near-ring networks with average nodal degrees near 2. Our analysis of redundancy results indicates that the heuristics presented here could achieve redundancy results within 20% of optimal tree design, but it the required spare capacity is higher than optimal mesh and protection cycle designs. That would present a trade-off between the complexity and capacity efficiency of the design.

The average and maximum backup paths reported by our design showed dependency on network radius, which increases logarithmically with network size in a random mesh network. We furthermore applied our algorithm to the case of a pre-designed network with fixed capacity to show how the available spare capacity of the network could be utilized for better restorability of the protection tree.

REFERENCES


