Redundant Trees for Preplanned Recovery in Arbitrary Vertex-Redundant or Edge-Redundant Graphs

Muriel Médard, *Member, IEEE,* Steven G. Finn, *Member, IEEE,* Richard A. Barry, and Robert G. Gallager, *Life Fellow, IEEE*

***Abstract*—We present a new algorithm which creates redundant trees on arbitrary node-redundant or link-redundant networks. These trees are such that any node is connected to the common root of the trees by at least one of the trees in case of node or link failure. Our scheme provides rapid preplanned recovery of communications with great flexibility in the topology design. Unlike previous algorithms, our algorithm can establish two redundant trees in the case of a node failing in the network. In the case of failure of a communications link, our algorithm provides a superset of the previously known trees.**

***Index Terms*—Graph theory, multicasting, network recovery, network robustness, routing, trees.**

I. Introduction

THE STUDY of self-healing networks, which automatically restore their functionality in the event of a failure, is prompted by the increasing reliance on high-speed communications and the requirement that these communications be robust in the case of certain failures. A failure may arise because a communications link is disconnected or a network node becomes incapacitated. Failures may occur in military networks under attack [26], as well as in public networks where failures, albeit rare, can be extremely disruptive [61]. For high-speed networks, rapid recovery from failure is important, as even a short down time may entail the loss of much data. Survivability may be considered to be a component of quality of service (QoS) [44], [47]. We present a tree-based algorithm applicable to optical wavelength-division multiplexed (WDM) systems [42], SONET, asynchronous transfer mode (ATM), or any protocol which allows the use of tree routings and redundancy for recovery from failures.

In this section, we present an overview of the criteria for our algorithm and briefly illustrate how these criteria relate to ATM and SONET. The study of self-healing networks is often classified according to the following three criteria (e.g., [8], [16]): the use of link (line) rerouting versus path (or end-to-end) rerouting, the use of centralized computation versus distributed computation, and the use of precomputed versus dynamically computed routes. A succint comparison of the different options can be found in [75, pp. 291–294] and in [30]. Our algorithm considers path rerouting which is preplanned and centrally computed. Other important criteria are the logical connections built by our algorithm and the network topologies on which our algorithm can operate.

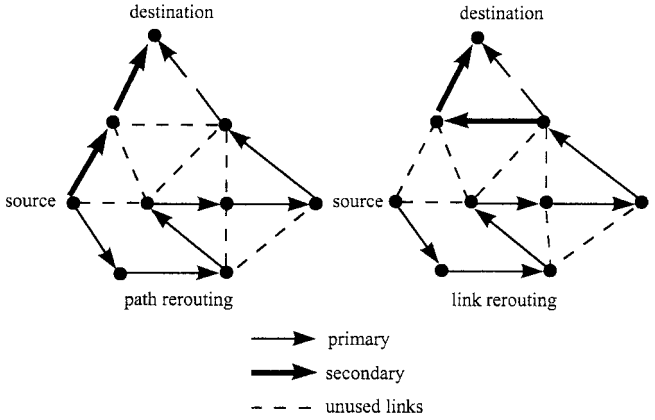


Fig. 1. Path and link rerouting. The failed link is shown as being interrupted.

A. Path Rerouting

Our algorithm implements path rerouting, i.e., in case of a failure which leaves a node disconnected from the primary route, a backup route, which may or may not share nodes and links with the primary route, is used. Link rerouting usually refers to the replacement of a link by link(s) connecting the two end nodes of the failed link. Fig. 1 shows path and link rerouting. The efficiency of path protection versus link rerouting in terms of the number of connections is considered in [17], [21], [66]. An extension or our algorithm to link rerouting is mentioned in Section III-C but is outside the scope of this paper.

For ATM, path rerouting performed by the private network node interface (PNNI) tears down virtual circuit (VC) connections after a failure, and the source node must then establish a new end-to-end route. Backup virtual paths (VP's) may be predetermined [34] or selected jointly by the end nodes [40]. Link rerouting in ATM usually involves a choice of new routes by nodes adjacent to the failure [2], [36]. In SONET,

Manuscript received April 25, 1997; revised February 12, 1998, July 29, 1998, and March 11, 1999; approved by IEEE/ACM TRANSACTIONS ON NETWORKING Editor G. Sasaki. This work was supported by DARPA ITO.

M. Médard is with the Coordinated Science Laboratory, University of Illinois Urbana-Champaign, Urbana, IL 61801 USA (e-mail: medard@comm.csl.uiuc.edu).

S. G. Finn is with Advanced Networks Group-Group 65, MIT Lincoln Laboratory, Lexington, MA 02173 USA.

R. A. Barry is with Sycamore Networks, Chelmsford, MA 01824 USA.

R. G. Gallager is with the Laboratory for Information and Decision Systems, Massachusetts Institute of Technology, Cambridge, MA 01273 USA.

Publisher Item Identifier S 1063-6692(99)08525-8.

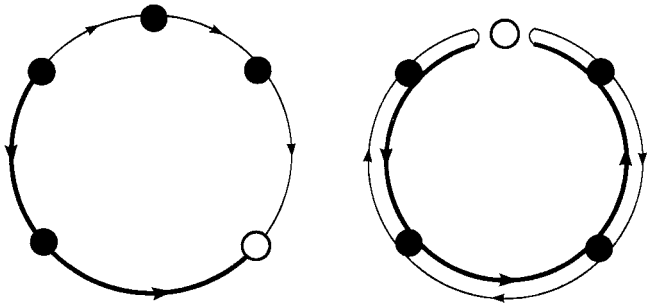


Fig. 2. UPSR and BLSR—thin lines show the primary paths and thick lines show the backup paths. The white nodes are the failed nodes.

path protection is usually performed by a unidirectional path-switched ring (UPSR), as illustrated in Fig. 2. An UPSR has two live streams, which are degenerate examples of trees. Link rerouting is performed in a SONET bidirectional line self-healing ring (BLSR) (see Fig. 2).

B. Preplanned Route Computation

Preplanned route computation is commonly termed automatic protection switching. Preplanned schemes generally offer better recovery speeds than dynamic approaches, which wait until a failure occurs before seeking alternate routes for the disrupted traffic and which generally involve software processing [46], [65]. Note, however, that there is thus a tradeoff between spare capacity and speed of restoration [32], [12].

Source routing, which is used by ATM PNNI, can be preplanned [28] or partially preplanned [44]. In SONET, both BLSR or UPSR are preplanned. SONET restoration time is specified to be under 60 ms. Recovery can be achieved in tens of milliseconds using optomechanical add–drop multiplexers [62], [55] and in a few microseconds using acousto-optical switches [18], [77]. These speeds can be contrasted with the 2-s restoration time goal [76], [55], [32] commonly set for dynamic distributed restoration using digital cross-connect systems (DCS) for ATM or SONET/SDH [27], [79], [22], [53]. Dynamic centralized path restoration for SONET ([25]) may even take minutes [76], [9], [7]. The performance of several algorithms is given in [11], [5].

C. Centralized Computation

The scheme we present is centralized, although some possibilities for distributed computation are briefly mentioned in Section III-C. A centralized scheme can never be less efficient than a distributed scheme in the number of connections which can be restored, but a centralized scheme requires a central processor with global knowledge of the network. If a centralized scheme is also preplanned, then the delay associated with setting up backup routes does not affect the restoration time because the delay is incurred up front at connection setup time. For high-speed fiber networks, connection setups are rare, and therefore the delay incurred at the beginning of a connection is acceptable as long as its is of the order of the delay to set up the connection. For ATM, centralized approaches have been proposed using virtual [54] or actual rings [37]. SONET uses centralized preplanned computation for setup of the primary and backup connections for UPSR and BLSR.

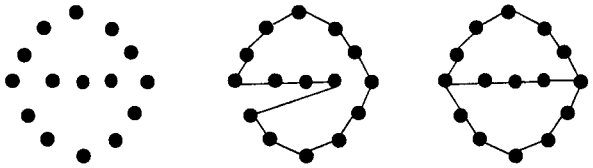


Fig. 3. An example of vertex topology where a ring-link topology is not minimum cost.

D. Logical Tree Connections

We create tree routings, such as would be used for multicast or incast applications. Multicast trees are used for ATM [64], [33], [1], [35], [78] and by tag-routing schemes, such as the Multiprotocol Label Switching standard [45]. Moreover, multicast is supported by SONET and is an attractive application in WDM networks [42], [31], [3]. Rather than use rerouting, which is based at the source [14] or at intermediate nodes [38], we create two trees such that one is a backup to another. Section II discusses the applicability of different redundant tree approaches. For ATM, several preplanned tree protection strategies are presented in [73].

E. Topological Requirements

The network topology, i.e., the physical layout of nodes and links, must be such that recovery is physically possible. Topological requirements are important because they determine whether a mechanism may be used in existing networks and because they may impact significantly the cost of constructing new networks or extensions to existing networks. The building blocks of SDH/SONET networks are generally self-healing rings (SHR's) and diversity protection (DP) [69], [67], [50], [49], [58]. SHR's are UPSR's or BLSR's, while DP refers to physical redundancy, where a spare link (node) is assigned to one or several links (nodes) [75, pp. 315–325]. Using only DP and SHR's is a constraint which has cost implications for building and expanding networks [71]. Fig. 3 shows a node configuration, a ring constructed on those nodes, and the edge-redundant topology with the minimum total link length. We may see that the ring is not the least-cost edge-redundant topology, if cost is proportional to total link length.

Even if we do not restrict ourselves to SHR's and DP, the network must satisfy certain topological requirements. If we wish to reroute traffic after the failure of any single link, the graph obtained by mapping the network nodes to vertices and the links to edges must be at least edge redundant, i.e., two-edge connected. If we wish to recover from an arbitrary node failure, then the graph must be at least vertex redundant, i.e., two-vertex connected. Our algorithm works on arbitrary redundant graphs.

F. Overview of the Paper

Our algorithm constructs two trees in such a fashion that the elimination of any vertex (edge) in the graph leaves each destination vertex connected to the source by at least one of the directed trees. This approach has been termed the multi-

tree approach [29]. The algorithm we present uses a different approach from [29] and is a generalization of the one presented in [42], [24]. We overview the issue of redundant trees in Section II. Section III presents our algorithm, first for the vertex failure case (Section III-A), then for the edge failure case (Section III-B). Section IV compares our algorithm to the approach of [29] and shows that our algorithm gives a superset of the previously available solutions and that our algorithm yields better performance for certain criteria and example networks. We give some conclusions and directions for further research in Section V.

II. Redundant Trees

We may now introduce our model. Let us consider that we have one source vertex and many destination vertices. We model a physical network as an undirected graph where each duplex communication link corresponds uniquely to an undirected edge and each network node corresponds uniquely to a vertex. To each undirected edge $[a,b]$ we associate two arcs, i.e., directed edges, (a,b), and (b,a). A directed route is an ordered sequence of vertices such that two vertices which are adjacent in the ordered sequence are endpoints on some edge of the graph. If the arc (a,b) appears in the route, then we say that arc (a,b) is in the route and, by abuse of notation, that edge $[a,b]$ is in the route. We select among the vertices of the graph a source and a set of destinations, different from the source. We create trees whose common root is the source and which include the set of destinations. For a particular source and set of destinations, we define a tree to be a set of arcs such that there is a directed route from the root to every vertex in the tree including only arcs in the tree and such that there are no cycles, i.e., routes which include a vertex more than once. By abuse of notation, we say that edge $[a,b]$ is in a tree when arc (a,b) or arc (b,a) is in the tree. Also, we say that vertex a is in a tree when an arc with a as an endpoint is in the tree. Finally, a directed route $(a_1,a_2,\cdots,a_{n-1},a_n)$ is in a tree if the set of arcs $\{(a_1,a_2),(a_2,a_3),\cdots,(a_{n-1},a_n)\}$ is a subset of the tree.

In this section, we present four different types of approaches to tree-based preplanned protection and evaluate their performance in achieving the following goal: **to design two directed trees in such a fashion that the elimination of any vertex (edge) in the graph (other than the source) leaves each destination vertex connected to the source by at least one of the directed trees for any source, and destination vertices in any vertex (edge)-redundant graph**. The two trees are termed the primary and secondary trees, or Red and Blue trees.

A. Menger's Theorem

The fact that, for any vertex (edge)-redundant graph, there exists a pair of vertex (edge)-disjoint paths between any two vertices is a consequence of Menger's theorem [57], [43]. Fig. 4 shows edge-disjoint and vertex-disjoint paths. Such approaches are presented in [59] for edge-disjoint paths and in [56], [72] for vertex-disjoint paths, usually associated with some sort of shortest path selection. Applications of these techniques to networks are presented in [4], [51]. However, none of these schemes guarantees trees in the case of a single source and multiple destinations, i.e., the union of primary paths may not be a tree and the union of backup paths may also not yield a tree. Fig. 5(a) shows a pair of vertex-disjoint paths between the source and vertex a and a pair of vertex-disjoint paths between the source and vertex b. The two primary paths together form a directed tree, but the two secondary paths do not form a directed tree. Therefore, finding pairs of vertex or edge-disjoint paths between the source and each destination vertex will not achieve our goal. The next sections describe methods which yield trees for the primary and backup paths.

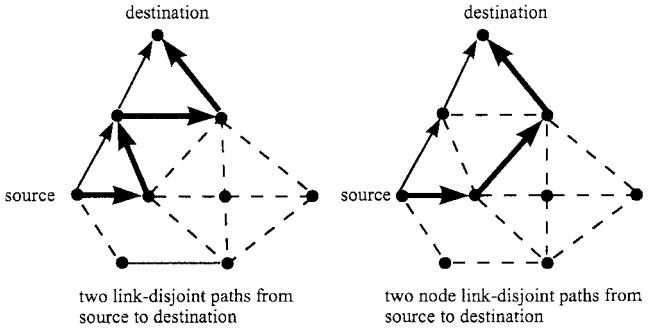


Fig. 4. Pair of link-disjoint paths and pair of vertex-disjoint paths.

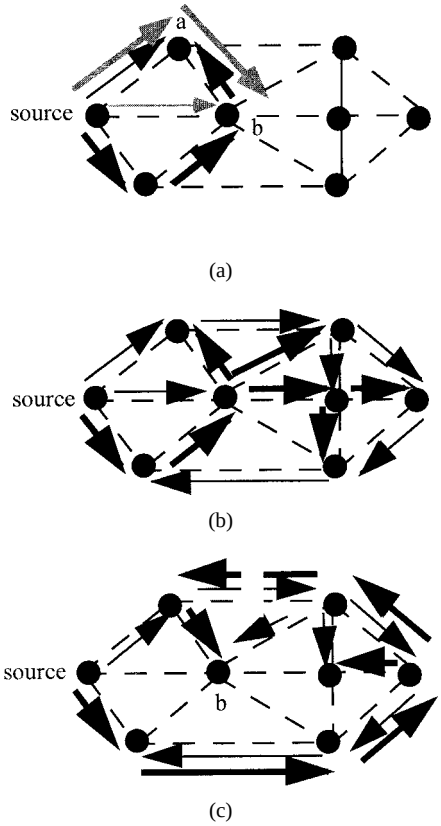


Fig. 5. Three different approaches to building redundant trees. The dashed lines show the edges, the thin arrow lines show the primary tree, and the thick arrow lines show the secondary tree. In the top figure, the grey lines indicate one pair of edge-disjoint paths and the full lines indicate another pair of edge-disjoint paths. An interrupted line corresponds to a failed edge.

B. Edge-Disjoint Trees

If two trees share no edges in common, then the failure of an edge cannot affect both the primary and the secondary

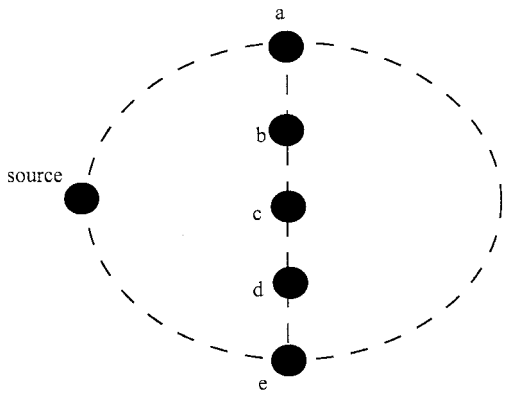


Fig. 6. An example of a network which cannot have edge-disjoint trees.

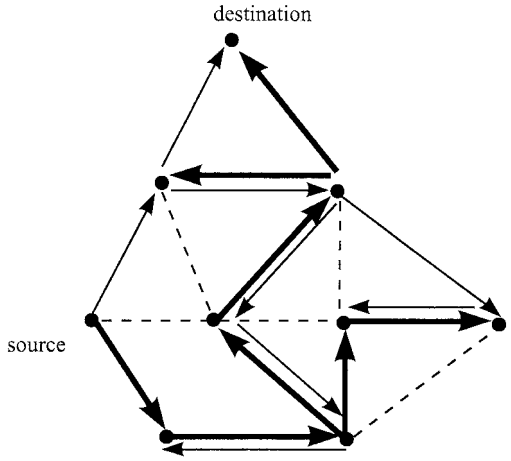


Fig. 7. Application of our algorithm for an edge-redundant graph.

tree. In [80], the case of two edge-disjoint spanning trees on undirected graphs is considered. In this case, one spanning tree remains unaffected even if the other tree ceases to be connected. Fig. 5(b) shows two edge-disjoint spanning trees. The topological constraint in [80] is, however, four-connectedness, which is greater than the edge-redundancy requirement, i.e., two-connectedness. A simple example of an edge-redundant graph which does not allow edge-disjoint spanning trees is given in Fig. 6. If $[c, d]$ is in the primary tree, then c must be reached in the secondary tree through $[b, c]$, and b is reached in the secondary tree through $[a, b]$. Thus, b cannot be reached in the primary tree. Moreover, the edge-disjoint approach does not produce vertex-disjoint trees (for instance trees which share only the source and the destination vertices). Therefore, the edge-disjoint approach does not comply with our goal.

C. Arc-Disjoint Trees

Rather than require two trees to share no edges, one may require that they not traverse links in the same direction, i.e., to be arc-disjoint [19], [62], [52]. Fig. 5(c) shows two arc-disjoint spanning trees and a failed edge. The failure of the edge indicated on the figure entails the failure of both arcs associated with that edge and thus vertex b becomes disconnected from the source.

D. Multi-Trees

The multi-tree approach presented in [29] satisfies our goal for the case of edge-redundant graphs. Fig. 7 shows an example of multi-trees built using our algorithm. In Section IV, we discuss this approach in more detail and show that the set of multi-trees achievable by the approach of [29] is a proper subset of the set of multi-trees achievable by our approach. The approach in [29] is not designed to satisfy the vertex failure case.

III. Redundant Trees for Vertex and Edge-Redundant Graphs

In the following sections, we present a new algorithm for achieving the goal stated at the beginning of Section II. Let us consider that we have a vertex (edge)-redundant undirected graph $\mathcal{G}(\mathcal{N}, \mathcal{E})$, where $\mathcal{N}$ is a set of nodes and $\mathcal{E}$ is a set of edges. We wish to show that, for any source vertex $s \in \mathcal{N}$, we may create two directed trees, which we shall name B and R, for Blue and Red, such that, after eliminating any vertex other than s (in the vertex-redundant case) or any edge (in the edge-redundant case), s remains connected to all vertices of $\mathcal{N}$ through B and/or through R deprived of the eliminated vertex or edge, respectively. In Section III-A, we describe our algorithm, for any vertex-redundant undirected graph. In Section III-B, we extend our results of Section III-A to edge-redundant graphs.

A. Multi-Trees for Vertex-Redundant Graphs

We now restrict ourselves to vertex failures. Note that a vertex failure entails the failure of at least two edges. We start by choosing an undirected cycle containing s. If this cycle does not include all vertices in the graph, we then choose an undirected path that starts on some vertex in the cycle, passes through some set of vertices not on the cycle, and ends on another vertex on the cycle. If the cycle and path above do not include all vertices of the graph, we again construct another path, starting on some vertex already included, passing through one or more vertices not included, and then ending on another already included vertex. The algorithm continues to add new vertices in this way until all vertices are included.

By applying Menger's theorem, we can establish that, in a vertex-redundant graph, a cycle must exist containing s. It can also be seen that, for any such cycle, a path can be added as above, and subsequent such paths can be added, in arbitrary ways, until all vertices are included. It is less simple to choose the B and R trees, as illustrated in Fig. 5(c).

The algorithm below specifies a particular ordering on vertices without finding what specific numerical values to associate with the vertices (since these values are only of use in seeing the analogy and not in establishing relative ordering of the vertices). Based on this ordering, it incrementally builds the trees. We start with a vertex-redundant graph $\mathcal{G} = (\mathcal{N}, \mathcal{E})$ and a source vertex s. The algorithm chooses a cycle, and then subsequent paths, and also orders the vertices as they get included in the cycle or one of the paths. We associate

two values with $s, v^R = 0$ and an arbitrary value V, so that $v^B(s) = V > 0$. One way to think of this is that there is a battery at vertex s and the ordering of the vertices corresponds to the voltages at different vertices when there are resistors on each edge. In the following algorithm, v can take either value $v^B(s)$ or $v^R(s)$ for s. The set of vertices which have already been assigned a value at stage j of the algorithm is denoted by $\mathcal{N}_j$. At each stage j of the algorithm, we keep track of arcs which can form one of the possible pairs of Red and Blue trees. $\mathcal{A}_j^B$ is the partial construction of the Blue tree at stage j. $\mathcal{A}_j^R$ is the partial construction of the Red tree at stage j.

Algorithm 1:

1) Set $j = 1$.
2) Choose any cycle $(s, c_1, \cdots, c_k, s)$ in the graph with $k \geq 2$. Let $\mathcal{N}_1$ be the set of vertices $\{s, c_1, \cdots, c_k\}$ and order these vertices by $V > v(c_1) > \cdots > v(c_k) > 0$.
3) $\mathcal{A}_1^B = \{(s, c_1), (c_1, c_2), \cdots, (c_{k-1}, c_k)\}$
   $\mathcal{A}_1^R = \{(s, c_k), (c_k, c_{k-1}), \cdots, (c_2, c_1)\}$.
4) If $\mathcal{N}_j = \mathcal{N}$, then terminate.
5) $j := j + 1$.
6) Choose a path $P_j = (x_{j,0}, x_{j,1}, \cdots, x_{j,L_j}), L_j \geq 2$, in the graph such that $x_{j,0} \in \mathcal{N}_{j-1}$, $x_{j,L_j} \in \mathcal{N}_{j-1}$, with $v(x_{j,0}) > v(x_{j,L_j})$. If $x_{j,0} = s$, then $v(x_{j,0}) = V$. If $x_{j,L_j} = s$ then $v(x_{j,L_j}) = 0$. The other vertices, $x_{j,i}, 1 \leq i < L_j$ are chosen outside of $\mathcal{N}_{j-1}$.
7) $\mathcal{N}_j = \mathcal{N}_{j-1} \cup \{x_{j,1}, \cdots, x_{j,L_j-1}\}$.
8) Order the vertices in P_j by $v(x_{j,0}) > v(x_{j,1}) > \cdots > v(x_{j,L_j-1}) > v_{\max}$ where
$$v_{\max} = \max_{y \in \mathcal{N}_{j-1}} (v(y) : v(y) < v(x_{j,0})).$$
9) $\mathcal{A}_j^B = \mathcal{A}_{j-1}^B \cup \{(x_{j,0}, x_{j,1}), (x_{j,1}, x_{j,2}), \cdots, (x_{j,L_j-2}, x_{j,L_j-1})\}$
   $\mathcal{A}_j^R = \mathcal{A}_{j-1}^R \cup \{(x_{j,L_j}, x_{j,L_j-1}), (x_{j,L_j-1}, x_{j,L_j-2}), \cdots, (x_{j,2}, x_{j,1})\}$.
10) Go to step 4.

This particular way of ordering vertices has the convenient feature that the new vertices at each new path P_j are entered all together into the previously ordered set. We also see that the dominant computational task is simply finding the cycle and the subsequent paths. Choosing the cycle and paths in different ways can yield different types of trees, depending on one's requirements.

Let us show that the algorithm terminates.

Lemma 1: The algorithm terminates if the graph is vertex redundant.

Proof: Since additional vertices are added to N_j each time step 7 is executed, the algorithm will terminate unless some nodes have not been assigned a value and step 6 is unable to find a new path P_{j+1}. Suppose that there is a vertex which has not been assigned a value. Equivalently, we suppose that $\mathcal{N} \backslash \mathcal{N}_j$ is not empty. Because of node-redundancy, there must be at least two edges from $\mathcal{N}$ to $\mathcal{N} \backslash \mathcal{N}_j$. If there exist two distinct edges $[z, x]$ and $[y, x]$ sharing an endpoint $x \in \mathcal{N} \backslash \mathcal{N}_j$ while $z, y \in \mathcal{N}_j$, then $P_j = (z, x, y)$, where we assume w.l.o.g. that $v(z) > v(y)$. If there do not exist two such edges $[z, x]$ and $[y, x]$, then because of node redundancy there must be at least two distinct nodes $x, w \in \mathcal{N} \backslash \mathcal{N}_j$ and two distinct nodes $z, y \in \mathcal{N}_j$ such that $[x, z]$ and $[w, y]$ are edges. If there is a path P from x to w using only vertices in $\mathcal{N} \backslash \mathcal{N}_j$, then assuming w.l.o.g. that $v(z) > v(y)$, we can create P_{j+1} by traversing vertices in the following order: z, x, the vertices of path P in order, w and finally y. If there is no such path P, then there is a path P' which has some vertex in $\mathcal{N}_j$. Let y' be the last vertex in P' to be in $\mathcal{N}_j$. If $y' \neq y$, then let us assume w.l.o.g. that $v(y') > v(y)$. We can create path P_{j+1} by traversing vertices in the following order: y', the vertices in P' between y' and w, w and finally y. If $y' = y$, then by Menger's theorems and our assumptions there must exist another path P'' from x to w which has some vertex in $\mathcal{N}_j$ and which does not include y. Let y'' be the last vertex in $\mathcal{N}_j$ in path P''. Let us assume w.l.o.g. that $v(y'') > v(y)$. We may create P_{j+1} by traversing vertices in the following order: y'', the vertices in P'' between y'' and w, w and finally y. Thus, in all cases, we can create a path P_{j+1} if $\mathcal{N} \backslash \mathcal{N}_j \neq \emptyset$. Q.E.D.

Let us consider two spanning trees, B and R of $\mathcal{N}$, each rooted at s, such that for any $x \in \mathcal{N} \backslash \{s\}$, for any $y \in \mathcal{N} \backslash \{s, x\}$, there exists a path from s to y in $B \backslash \{x\}$ or in $R \backslash \{x\}$. At least one pair of such B and R trees exists. Consider the case where $B = \mathcal{A}_{\tilde{j}}^B$ and $R = \mathcal{A}_{\tilde{j}}^R$, where $\tilde{j}$ is the j at which the algorithm terminates in step 4. Let us show that there is a directed route in B from s to any $x \in \mathcal{N}$. From our construction, it is immediate that there is a directed route in B from s to any $x \in \mathcal{A}_1^B$, since the latter is simply a directed ring. If there is a directed route in B from s to any vertex in $\mathcal{N}_j$, then let us show that there is a directed route in B from s to any $x \in \mathcal{N}_{j+1}$. If $x \in \mathcal{N}_j$, then we are done. If $x \in \mathcal{N}_{j+1} \backslash \mathcal{N}_j$, then $x \in P_{j+1}$. There is a directed route in $\mathcal{A}_j^B$ from s to $x_{j+1,0}$ using only nodes in $\mathcal{N}_j$ and a directed route in $\mathcal{A}_{j+1}^B$ from $x_{j+1,0}$ to x using nodes on P_{j+1}. Combining the two routes yields a route in $\mathcal{A}_{j+1}^B$ from s to x. Therefore, by induction, there is a directed route in B from s to any $x \in \mathcal{N}$. In a similar way, we can prove that there is a directed route in R from s to any $x \in \mathcal{N}$. The only significant change is that $\mathcal{A}_1^B$ must be replaced by $\mathcal{A}_1^R$ and that $x_{j,0}$ must be replaced by $x_{j+1,L_{j+1}}$. We can see by our induction argument on j that the directed route in B_j from s to any $x \in \mathcal{N}_j$ consists of monotonically decreasing values (excluding s). Similarly, the directed route in R_j from s to any $x \in \mathcal{N}_j$ consists of monotonically increasing values (excluding s). Since these properties hold for each j, they also hold for the final trees B and R. Therefore, there can be no cycles in B, for a cycle would imply that the values of the vertices traversed would decrease and then increase. Similarly, there are no cycles in R. The fact that B and R have no cycles and that there is a directed route from s to any vertex in $\mathcal{N}$ for each tree implies that B and R are trees rooted at s.

It now remains for us to show that eliminating a vertex in $\mathcal{N}$ leaves each remaining vertex connected to s through B and/or R. Let $x \neq s$ be an arbitrary vertex that is removed from the graph, and let $y \neq s$ be some other vertex. We wish to show that y can still be reached from s in either the B tree or the R

tree. Since the vertices are ordered, we either have $v(y) > v(x)$ or $v(y) < v(x)$. For the first case, y can still be reached from s on B, since the path from s to y is monotonically decreasing in vertex values, and thus cannot contain vertex x. In the second case, y can be reached from s on R because of the monotonic increasing property there. Our discussion leads to stating the following proposition.

Proposition 1: There exist at least one B tree and one R tree such that B and R are spanning trees rooted at s and such that for any $x \in \mathcal{N}\backslash\{s\}$, for any $y \in \mathcal{N}\backslash\{s, x\}$, there exists a path from s to y in $B\backslash\{x\}$ or in $R\backslash\{x\}$.

B. Multi-Trees for Edge-Redundant Graphs

The above algorithm for vertex failure protection does not work in this case, because it is not always possible at a stage $j \geq 2$ to find paths as above. In particular, it is sometimes necessary to find a cycle that leaves the set of vertices $\mathcal{N}_{j-1}$ on one vertex and returns to the same vertex, i.e., with $x_{j,0} = x_{j,L_j}$. This can be handled by letting each vertex x have two "voltages," $v^B(x)$ and $v^R(x)$ associated with it. We arbitrarily select $v^B(s) = V$ and $v^R(s) = 0$. The ordering in step 2 of the algorithm is then replaced with

$$V > v^B(c_1) > v^R(c_1) > v^B(c_2) > v^R(c_2) > \cdots > v^B(c_k) > v^R(c_k) > 0.$$

The ordering in step 8 of the algorithm is replaced in a similar way. The complete algorithm is then Algorithm 2, as follows.

Algorithm 2:

1) Set $j = 1$.
2) Choose any cycle $(s, c_1, \cdots, c_k, s)$ in the graph with $k \geq 2$. Let $\mathcal{N}_1$ be the set of vertices $\{s, c_1, \cdots, c_k\}$ and order these vertices by $v > v^B(c_1) > v^R(c_1) > v^B(c_2) > v^R(c_2) > \cdots > v^B(c_k) > v^R(c_k) > 0$.
3) $\mathcal{A}_1^B = \{(s, c_1), (c_1, c_2), \cdots, (c_{k-1}, c_k)\}$
   $\mathcal{A}_1^R = \{(s, c_k), (c_k, c_{k-1}), \cdots, (c_2, c_1)\}$.
4) If $\mathcal{N}_j = \mathcal{N}$, then terminate.
5) $j := j + 1$.
6) Choose a path or cycle $P_j = (x_{j,0}, x_{j,1}, \cdots, x_{j,L_j})$ in the graph. For a path, $L_j \geq 2$, with $x_{j,0} \in \mathcal{N}_{j-1}, x_{j,L_j} \in \mathcal{N}_{j-1}$, and $v^B(x_{j,0}) > v^B(x_{j,L_j})$. For a cycle, $L_j \geq 3$, with $x_{j,0} = x_{j,L_j} \in \mathcal{N}_{j-1}$. The other vertices, $x_{j,i}, 1 \leq i < L_j$ are chosen outside of $\mathcal{N}_{j-1}$.
7) $\mathcal{N}_j = \mathcal{N}_{j-1} \cup \{x_{j,1}, \cdots, x_{j,L_j-1}\}$.
8) Order the new vertices by $v^B(x_{j,0}) > v^B(x_{j,1}) > v^R(x_{j,1}) > v^B(x_{j,2}) > v^R(x_{j,2}) > \cdots > v^B(x_{j,L_j-1}) > v^R(x_{j,L_j-1}) > v_{\max}$ where $v_{\max} = \max_{\{\mathcal{N}_{j-1} | v^B(y) < v^B(x_{j,0}), v^R(y) < v^B(x_{j,0})\}} \{v^B(y), v^R(y)\}$.
9) $\mathcal{A}_j^B = \mathcal{A}_{j-1}^B \cup \{(x_{j,0}, x_{j,1}), (x_{j,1}, x_{j,2}), \cdots, (x_{j,L_j-2}, x_{j,L_j-1})\}$
   $\mathcal{A}_j^R = \mathcal{A}_{j-1}^R \cup \{(x_{j,L_j}, x_{j,L_j-1}), (x_{j,L_j-1}, x_{j,L_j-2}), \cdots, (x_{j,2}, x_{j,1})\}$.
10) If $\mathcal{N}_j = \mathcal{N}$, then terminate.
11) Go to step 4.

We first show that the algorithm terminates.

Lemma 2: The algorithm for the edge-redundant case terminates.

Proof: We shall proceed by contradiction. The algorithm would fail to terminate correctly iff, at step 6, no new path or cycle could be found but a vertex in $\mathcal{N}$ was never included in $\mathcal{N}_j$. We therefore assume that such a vertex exists. Because of the connectedness of our graph, there is an edge, e, from $\mathcal{N}_{j-1}$ to $\mathcal{N}\backslash\mathcal{N}_{j-1}$ which connects some x in $\mathcal{N}\backslash\mathcal{N}_{j-1}$ to some vertex y in $\mathcal{N}_{j-1}$. Because of the edge-redundancy of our graph, there exists a path between s and x which does not traverse e. Let f be the last edge from which this path exits $\mathcal{N}_j$. f exits $\mathcal{N}_j$ at vertex z. Note that z and y may be the same. Then, there exists a path from y to z, or z to y, passing through x, which would be selected at step 6 in the algorithm. Therefore, we have a contradiction. Q.E.D.

As in the previous section, we seek to find two spanning trees, B and R, each rooted at s, such that for any $x \in \mathcal{N}\backslash\{s\}$, for any $y \in \mathcal{N}\backslash\{s, x\}$, there exists a path from s to y in $B\backslash\{x\}$ or in $R\backslash\{x\}$. At least one pair of such B and R trees exists. Let $B = \mathcal{A}_j^B$ and $R = \mathcal{A}_j^R$. If we replace $v(x)$ by $v^B(x)$, the same arguments as for Algorithm 1 show that there exists a directed route in B from s to any $x \in \mathcal{N}$. Similarly, if we replace $v(x)$ by $v^R(x)$, the arguments given for Algorithm 1 may be used to show that there exists a directed route in R from s to any $x \in \mathcal{N}$. Moreover, there can be no cycles in B, for a cycle would imply that the "voltages," $v^B(x)$, of the vertices traversed would decrease and then increase. Similarly, there are no cycles in R. Therefore, as for the vertex-redundant case, we may show that B and R are spanning trees of $\mathcal{N}$ rooted at s.

It now remains for us to show that eliminating an edge e in $\mathcal{E}$ leaves each remaining vertex connected to s through B or R. If the edge e is on neither or only one of the trees, then the result is trivial. Hence, the interesting case occurs when e is in both trees. Suppose some edge $e = [x_{j,i}, x_{j,i+1}]$ is in both trees, for some j and some $0 < i < L_j - 1$. Note that e cannot have as an endpoint the first or last node of P_j. If e is removed, and if e is on the directed route in B from s to some y, then we must have $v^B(x_{j,i+1}) > v^B(y) > v^R(y)$. We also have $v^R(x_{j,i}) > v^B(x_{j,i+1})$, so $v^R(x_{j,i}) > v^R(y)$. This, however, means that e cannot be on the directed route from s to y in R. Thus, the modified algorithm works for the edge-redundant case and we have proved the proposition below.

Proposition 2: There exist two spanning trees, B and R, rooted at s such that, for any $[x_1, x_2] \in \mathcal{E}$, for any $x \in \mathcal{N}\backslash\{s\}$, there exists a path from s to x in $B\backslash\{[x_1, x_2]\}$ or in $R\backslash\{[x_1, x_2]\}$.

Note that whenever we select a cycle in step 6, we create a situation in which a vertex failure at the vertex starting that cycle will cause a failure in both trees.

Fig. 7 shows an example of the application of our algorithm. Although the B and R trees share edges, we see that eliminating any edge does not disconnect any vertex from the source. We may compare this situation with that in Fig. 5(c). Figs. 8 and 9 show two different examples of B and R trees which can be built from the same numbering on the vertices of a graph.

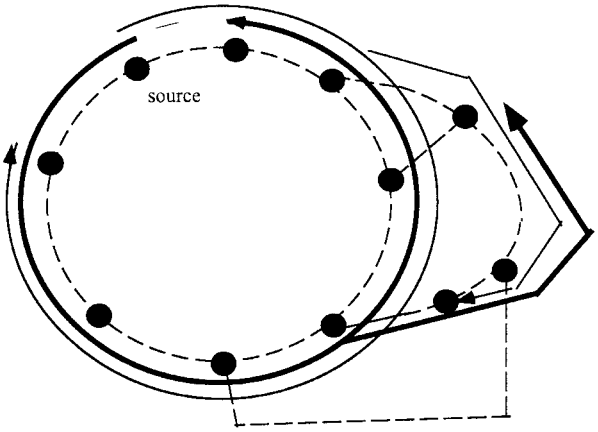


Fig. 8. One example of B and R trees.

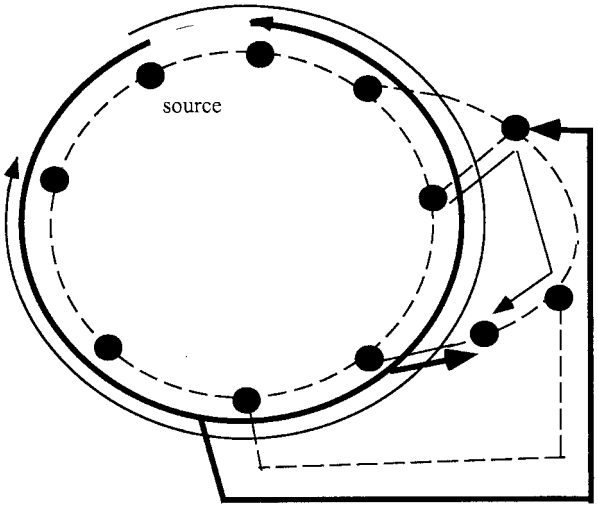


Fig. 9. Another example of B and R trees using the same node numbering.

Note that the running time of our algorithm is $O(n^3)$, where n is the number of vertices.

We may briefly comment on the possibilities of using our algorithm in a distributed manner, although our algorithm has been described in a manner which makes it centralized and preplanned. Since the union of two trees in which the root of one of the trees is part of the other tree is a tree, we can apply our algorithm independently to two networks which share a single vertex (a situation akin to single-homing rings). The union of the two Blue trees is a valid Blue tree and the union of the Red trees is a valid Red tree for edge redundancy if we take the single shared vertex to be the root of one of the Blue and Red tree pairs.

IV. Comparison of Our Scheme with the Previous Multi-Tree Approach

In this section, we show that our algorithm for the vertex-redundant case (Algorithm 1) is more general than the multi-tree approach presented in [29]. In Section IV-A, we first give an overview of the algorithm presented in [29]. Next, in Section IV-B, we show that there are multi-trees which can be built by our scheme but which cannot be built by the previous approach. The examples we present are motivated by different performance metrics. We show that our algorithms offer better performance, for our examples and those metrics, than [29]. Finally, in Section IV-C, we show that any multi-trees generated by the scheme in [29] can be generated by our algorithm. Note that we consider our algorithm for vertex-redundant graphs, even though [29] considers edge failures. There are two reasons why we may consider Algorithm 1 rather than Algorithm 2. First, the algorithm in [29] applies to vertex-redundant sub-networks only. Second, since eliminating a node entails the elimination of at least two edges, showing that Algorithm 1 is more general than the algorithm in [29] entails that our solutions to edge failure are a superset than those offered by [29].

A. Overview of the Previous Algorithm

In order to compare our scheme to the one presented in [29], it is necessary to reiterate most of the algorithm in [29]. The scheme in [29] determines an $s-t$ numbering ([20], [39]) denoted by g. Let us select two nodes, s and t in $\mathcal{G}$, such that s and t are end points of some edge in $\mathcal{E}$. An $s-t$ numbering on the graph is defined to be such that $g(s) = 1$, $g(t) = n$ and every vertex $x \in \mathcal{N}\backslash\{s,t\}$ has two adjacent vertices, z and y, which satisfy $g(z) < g(x) < g(y)$. Note that an $s-t$ numbering is akin to our value assignment.

The scheme in [29] divides an edge-redundant network into several vertex-redundant blocks. The algorithm then works on each of those blocks. Let us consider an example of one such block. The algorithm creates an $s-t$ numbering and then builds trees which traverse the numbering in increasing or decreasing order (except for two special vertices, s and t). The scheme of [29] first selects the first edge which will be used by the primary tree. The root is denoted by s and $[s,t]$ is the edge chosen as the first edge of the primary tree.

In order to construct an $s-t$ numbering, the scheme in [29] first constructs a depth-first-search (DFS) numbering and its associated tree [13] rooted at t and such that $N(t) = 1$ and $N(s) = 2$, where $N(v)$ is the numbering given by the DFS to vertex v. Note that the DFS numbering is used solely as a tool to construct the $s-t$ numbering and is not the $s-t$ numbering itself.

Let us first establish some definitions. Lowpoint paths are defined in the following manner: $(v_1, \cdots, v_m, v_{m+1})$ is the lowpoint path from v_1 if $(v_1, \cdots, v_m)$ is a directed tree route in the DFS tree, (v_m, v_{m+1}) is a graph arc which is not in the DFS tree (such an arc is called a frond) and $N(v_{m+1})$ is the smallest number for which there exists such a path from v_1. The algorithm to build $s-t$ numberings given in [29] is given below. The algorithm operates on a stack which initially holds only s and t, with s on top. Note that vertices are "new" if they have not yet been placed on the stack.

Algorithm 3:

Set $i := 1$

while the stack is not empty do begin

1) Remove the top vertex v from the stack
2) $g(v) := i$; $i := i + 1$

3) For all tree arcs (v,w) to a new vertex w, let $(w = w_1,\cdots,w_m,w_{m+1})$ be the lowpoint path from w; push $w_m,\cdots,w_1$ onto the stack (w_m first)
4) For all paths $(u_0,u_1,\cdots,u_m,v)$ from some old vertex u_0 to v such that $u_1,\cdots,u_m$ are new, (u_i,u_{i+1}), $m>i\geq 0$, is a tree edge and (u_m,v) is a frond, push $u_1,\cdots,u_m$ onto the stack (u_1 first)

end.

At this point, we may make a remark about step 4 which will be useful in Section IV-C. We know that u_0 must still be on the stack, otherwise u_0 would have been popped already. For the sake of contradiction, let us suppose that u_0 was popped before v. When u_0 was popped, v was new or old. If v was new when u_0 was popped, then there existed some path $(w,v_1,\cdots,v_m,v,u_m,\cdots,u_1,u_0)$ from some old w to u_0 such that $v_1,\cdots,v_m,v,u_m,\cdots,u_1$ were new. Thus, $u_m,\cdots,u_1$ could not be new when v was popped. If v was old when u_0 was popped, then the path $(v,u_m,\cdots,u_1,u_0)$ is from an old vertex to u_0 with only new vertices in between. Thus, u_0 must still be on the stack when v is popped. Therefore, $g(v)<g(u_1)<g(u_2)<\cdots<g(u_m)<g(u_0)$.

B. Comparison of Tree-Building Schemes

In this section, we present examples of networks for which, under certain criteria, the algorithm we present yields better outcomes than Algorithm 3.

1) Propagation delays:[1] Consider a network with a root node s (which sources the two trees) and F sets of three vertices:

$u(0,1),u(0,2),u(0,3)$
$u(0,1),u(0,2),u(0,3)$
.
.
.
$u(F-1,1),u(F-1,2),u(F-1,3)$.

For each $i=0,\cdots,F-1$, the vertices
$(s,u(i,1),u(i,2),u(i3),s)$
form a cycle. We will call these cycles "petals." The node $u(i,2)$ is referred as the "tip" of the petal.

The tips of the petals are connected to form a cycle
$(u(0,2),u(1,2),u(2,2),\cdots,u(F-1,2),u(0,2))$.

We will call this cycle the "rim." Algorithm 3 will first designate a node as t. Without loss of generality we can assume petal i has t, and node $t=u(i,1)$. Note that the numbering will have $g(s)=1$ and $g(t)=3F+1$. All other nodes have numbers between $g(s)$ and $g(t)$. Now note that one of the trees, say T, is formed by having each node find a parent that has a larger number. In other words, T is actually a tree rooted at t, which is connected to s by the edge (s,t). This tree T does not have any branches to s, except through (s,t). This means that there is a path along the tree that goes at least halfway around the rim. Thus, the maximum distance from the root s to any node along the tree is at least $F/2$.

On the other hand, Algorithm 1 can find trees that have maximum number of hops from the root at most two (e.g., each petal forms a cycle in the algorithm). Fig. 10 illustrates a network with four petals and the two trees constructed by Algorithm 1. Thus, for this example, Algorithm 1 can insure a short propagation delay.

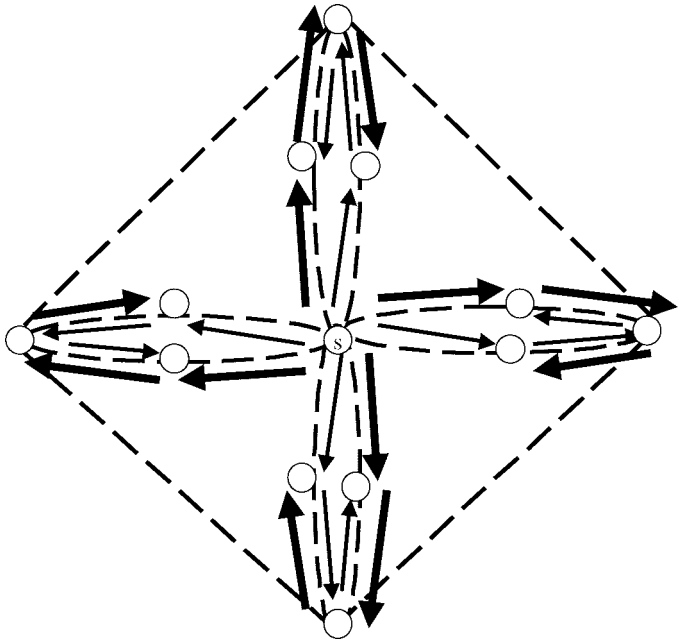


Fig. 10. An example of trees which can be built with Algorithm 1 but not Algorithm 3.

2) Channel availability in WDM networks: In this example, we consider an example of a network where two redundant trees cannot be found by Algorithm 3 because of lack of availability of certain channels. Such channels might be, for instance, wavelengths in a WDM system. The network for our discussion is illustrated in Fig. 11. The primary tree is to be carried by wavelength λ_1, while the secondary is to be carried by wavelength λ_2. Let us suppose that, owing to the presence of other traffic in the network, λ_2 is unavailable on $[s,t]$ and λ_1 is unavailable on $[t,v_1]$. Note that, because there are three edges incident on s, we cannot split s to form two distinct vertex-redundant blocks. Note that Algorithm 3 will not be able to operate on the network of Fig. 11 given our constraints. Indeed, we see that the primary broadcast tree on λ_1 must have (s,t) included in it, because t cannot be reached in any other way by λ_1. Moreover, t cannot access any node other than s using λ_1. Therefore, any broadcast tree using λ_1 must have two arcs emanating from s and such a tree cannot be constructed by Algorithm 3.

3) Replication at Nodes: For our final example, we consider a constraint on the replication or splitting that can occur at a node. We place a constraint, per tree, that each node can replicate incoming signals at most twice for its output. For instance, a WDM system may carry each tree on a separate wavelength and may have at most a two-way splitter per wavelength. Fig. 12 illustrates the network we consider for our example and the application of Algorithm 1 to the example network. Using Algorithm 3, a three-way split would have to occur at t. Indeed, access to the vertices on the three "petals" emanating from t can only be achieved through t

[1]This example is due to one of the anonymous reviewers, whose contribution the authors gratefully acknowledge.

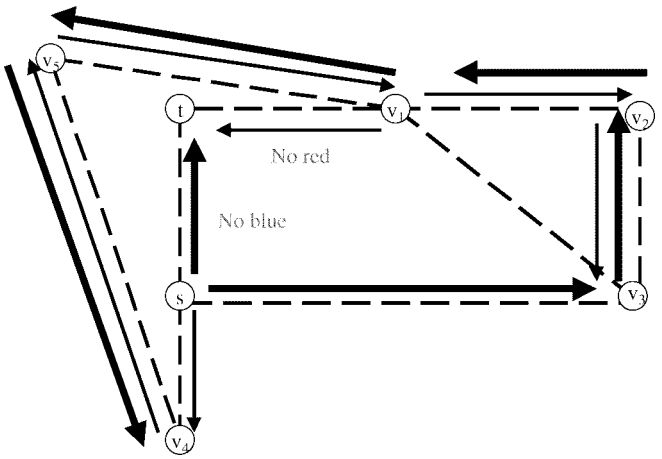


Fig. 11. A second example of trees which can be built with Algorithm 1 but not Algorithm 3.

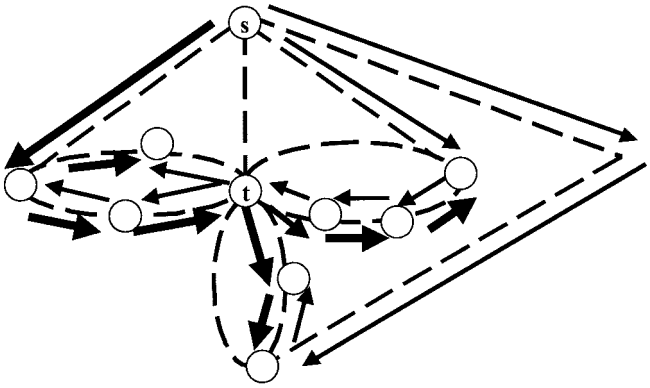


Fig. 12. A third example of trees which can be built with Algorithm 1 but not Algorithm 3.

or through s directly. If we have a constraint that a vertex can have a maximum of two outgoing arcs per tree, then that constraint cannot be met using Algorithm 3 but can be met by Algorithm 1.

C. Construction of Previously Known Multi-Trees with Our Scheme

Let us now show that every multi-tree pair constructed by the scheme in [29] can be built by our scheme. In order to show this, it is sufficient to show that any ordering achieved by the scheme in [29] can be achieved by our scheme. Let us consider a block. Each block is vertex redundant, although we are interested in edge failures. There exist a vertex z other than s which is adjacent to t, because we have vertex redundancy. Moreover, z is reachable from s by the DFS tree. Consider the cycle constructed by adding $[z,t]$ and $[t,s]$ to the edges on the path from s to z in the DFS tree. The first time that step $3_{\text{Algorithm 3}}$ is run, 1 is the lowpoint of s. The first run of step $3_{\text{Algorithm 3}}$ corresponds to picking a cycle, albeit a cycle constrained by the DFS structure. Every subsequent execution of $3_{\text{Algorithm 3}}$ or $4_{\text{Algorithm 3}}$ corresponds to the selection of set of vertices as in step $6_{\text{Algorithm 1}}$ and the ordering of the vertices as done in step $8_{\text{Algorithm 1}}$, as explained below.

- **Execution of** $3_{\text{Algorithm 3}}$: the end-point w_{m+1} of the lowpoint path must not have been popped off the stack yet, for otherwise the tree arc (w_{m+1}, w_m) would have been considered in an earlier execution of $3_{\text{Algorithm 3}}$. Hence, $g(v) < g(w_{m+1})$. The new vertices are pushed onto the stack. These vertices can be chosen by step $6_{\text{Algorithm 1}}$. The placing of vertices on the stack satisfies proper ordering achieved by step $8_{\text{Algorithm 1}}$.
- **Execution of** $4_{\text{Algorithm 3}}$: one P_j end-point (v) was just popped off the stack and the other (u_0) is still on the stack, as explained in our remark in Section IV-A. The vertices u_0 and v are already ordered relatively to each other, with $g(u_0) > g(v)$. The vertices which are pushed onto the stack form a path. The order in which they are pushed ensures that the $s-t$ numbers, or voltages, are in the proper order along the new path. The order in which the vertices are pushed onto the stack is equivalent to the order established by step $8_{\text{Algorithm 1}}$.

Note that the algorithm selects paths rather than cycles (except for the choice of s and t, which corresponds to the choice of our first cycle), whereas our algorithm may select cycles at step $6_{\text{Algorithm 1}}$. Such paths always exist for Algorithm 3 because the blocks are vertex redundant.

V. CONCLUSION

We have established an algorithm which, on any vertex (edge)-redundant network, can build two directed multi-trees which together make connections immune to failure of a single vertex (edge). Our algorithm expands the set of known solutions and, for certain criteria and example networks, yield better performance than the previously known method. Our algorithm can be modified to take into account cost functions and it can be implemented in a distributed fashion for edge redundancy under certain conditions. Any cost function can be applied to select our cycles and paths. Thus, the cost functions, such as delay minimization, which are often associated with Steiner trees for multicasting, can be taken into account when we select our cycles and paths.

There are several possibilities for further investigation. Four such areas of interesting future work are the areas of multiple failures, spare capacity planning, cost minimization, and link rerouting. One approach to several failures is to constuct two trees, if possible, to guard against one failure, then construct two more trees on the network remaining after a failure. If failures are sufficiently rare, where no two failures are expected to occur simultaneously, then such an approach may be acceptable, although it may require rerouting traffic after a failure from the first secondary-tree built to the second primary-tree built. If failures occur not because of benign conditions, such as hardware-component fatigue, but, e.g., because of incorrect messages which may disrupt several vertices, then the approach of having only of two multi-trees at any time may not be sufficient. The issue of three-trees for edge failure in a three-connected network has been addressed in [81]. The issue of whether a similar extension may be achieved using our algorithms is of interest.

The issues of spare capacity planning and cost minimization are related, since they concern the application of our algorithm in a fashion which is cost effective. The fact that we provide more solutions than the previously known multi-tree algorithm for edge failures entails that there are certain networks and associated cost metrics for which our algorithm performs better, as illustrated in Section IV-B. Spare capacity is an important issue because having two live streams or having capacity reserved for recovery may be costly. Moreover, under certain conditions, there may not be enough capacity to accommodate certain multi-trees. Therefore, the link-by-link capacity, which we have not explicitly considered here, may need to be incorporated in the criteria for tree selections. Note that the problem of finding minimum cost trees for a certain source and a set of destinations is the Steiner tree problem, which is NP-hard. Good surveys of the problem and heuristics can be found in [63], [70], [68], [74] and applications to networks can be found in [6], [48], [10]. Because the Steiner tree problem is NP-hard, the issue of cost minimization and capacity utilization may be difficult.

Finally, we may comment on extending our algorithm to loopback so as to achieve link or node rerouting rather than path rerouting. The placing of traffic onto the backup path in BLSR is called loopback, because in a ring, the traffic backhauls through the vertices it already traversed. The multi-tree approach we have presented is mostly a path rerouting approach. A drawback to path rerouting is that it requires each path to have a dedicated backup path. The maximum spare capacity may be of the order of the sum of the capacities of all the routes in the network. Link rerouting does not reroute paths but rather the traffic carried by a certain link or vertex. The spare capacity need never be greater than the maximum capacity of any link or vertex. Since link (or vertex) rerouting is not based on routes, the tree-based approach we have given is not appropriate. We have extended our approach to allow for link rerouting for vertex (edge) failures in arbitrary vertex (edge)-redundant networks ([23]).

REFERENCES

[1] M. H. Ammar, S. Y. Cheung, and C. M. Scoglio, "Routing multipoint connections using virtual paths in an ATM network," in *Proc. INFOCOM'93*, pp. 1c.4.1–8.

[2] M. Azuma, Y. Fujii, Y. Sato, T. Chujo, and K. Murakami, "Network restoration algorithm for multimedia communication services and its performance characteristics," *IEICE Trans. Commun.*, vol. E78-B, no. 7, pp. 987–994, July 1995.

[3] G. B. Brewster and M. S. Borella, "Multicast routing algorithms for the WDM shufflenet local optical network," in *Proc. ICC '97*, vol. 1, pp. 111–115.

[4] R. Bhandari, "Optimal diverse routing in telecommunication fiber networks," in *Proc. INFOCOM'94*, vol. 3, pp. 11c.3.1–11.c.3.11.

[5] J. Bicknell, C. E. Chow, and S. Syed, "Performance analysis of fast distributed network restoration algorithms," in *Proc. GLOBECOM '93*, vol. 3, pp. 1596–1600.

[6] K. Bharath-Kumar and J. M. Jaffe, "Routing to multiple destinations in computer networks," *IEEE Trans. Commun.*, vol. COM-31, pp. 343–351, Mar. 1983.

[7] A. Bellary and K. Mizushima, "Intelligent transport network survivability: Study of distributed and centralized control techniques using DCS and ADM's," in *Proc. GLOBECOM'90*, pp. 1264–1268.

[8] A. Banerjea, C. J. Parris, and D. Ferrari, "Recovering guaranteed performance service connections from single and multiple faults," in *Proc. GLOBECOM'94*, vol. 1, pp. 162–166.

[9] M. Barezzani, E. Pedrinelli, and M. Gerla, "Protection planning in transmission networks," in *Proc. SUPERCOMM/ICC'92*, pp. 316.4.1–316.4.5.

[10] F. Bauer and A. Varma, "ARIES: A rearrangeable edge-based online Steiner algorithm," *IEEE J. Select. Areas Commun.*, vol. 15, pp. 382–397, Apr. 1997.

[11] C. E. Chow, J. Bicknell, S. McCaughey, and S. Syed, "A fast distributed network restoration algorithm," in *Proc. 12th Int. Phoenix Conf. Computers and Communcations*, Mar. 1993, pp. 261–267.

[12] R. S. K. Chng, C. P. Botham, D. Johnson, G. N. Brown, M. C. Sinclair, M. J. O'Mahony, and I. Hawker, "A multi-layer restoration strategy for reconfigurable networks," in *Proc. GLOBECOM '94*, vol. 3, pp. 1872–1878.

[13] T. H. Corman, C. E. Leiserson, and R. L. Rivest, *Introduction to Algorithms*. Cambridge, MA: MIT Press; New York: McGraw-Hill, 1990.

[14] J.-M. Chang and N. F. Maxemchuk, "Reliable broadcast protocols," *ACM Trans. Comput. Syst.*, vol. 2, no. 3, pp. 251–273, Aug. 1984.

[15] M. Doar and I. Leslie, "How bad is naïve multicast routing?," in *Proc. INFOCOM'93*, pp. 1c.2.1–1c.2.8.

[16] R. Doverspike, "A multi-layered model for survivability in intra-LATA transport networks," in *Proc. GLOBECOM'91*, pp. 2025–2031.

[17] R. Doverspike and B. Wilson, "Comparison of capacity efficiency of DCS network restoration routing techniques," *J. Network Syst. Manage.*, vol. 2, 1994.

[18] D. Edinger, P. Duthie, and G. R. Prabhakara, "A new answer to fiber protection," *Telephony*, Apr. 9, 1990, pp. 53–55.

[19] J. Edmonds, "Edge-disjoint branchings," in *Combinatorial Algorithms, Courant Institute Computer Science Symposium 9*. New York: Algorithmics Press, Jan. 24–25, 1972.

[20] S. Even and R. E. Tarjan, "Computing an st-numbering," *Theoretical Comput. Sci.*, vol. 2, pp. 339–344, 1976.

[21] T. Frisanco, "Optimal spare capacity design for various protection switching methods in ATM networks," in *Proc. ICC 97*, vol. 1, pp. 293–298.

[22] H. Fujii and N. Yoshikai, "Double search self-healing algorithm and its characteristics," *Electron. Commun. Jpn.*, pt. 1, vol. 77, no. 3, 1994.

[23] M. Médard, S. G. Finn, and R. A. Barry, "WDM loop-back recovery in mesh networks" in *Proc. OFC'98*, pp. 298–299.

[24] ———, "A novel approach to automatic protection switching using trees," in *Proc. ICC'97*, pp. 272–276.

[25] A. Gersht, S. Kheradpir, and A. Shulman, "Dynamic bandwidth-allocation and path-restoration in SONET self-healing networks," *IEEE Trans. Rel.*, vol. 45, pp. 321–331, June 1996.

[26] C. J. Green, "Protocols for a self-healing network," in *Proc. 1995 Military Communications (MILCOM)*, pt. 1, pp. 252–256.

[27] W. D. Grover, "The SelfHealing™ Network," *Proc. GLOBECOM '87*, vol 2, pp. 1090–1095.

[28] D. K. Hsing, B.-C. Cheng, G. Goncu, and L. Kant, "A restoration methodology based on pre-planned source routing in ATM networks," *Proc. ICC '97*, vol. 1, pp. 277–182.

[29] A. Itai and M. Rodeh, "The multi-tree approach to reliability in distributed networks," *Inform. Computation*, vol. 79, pp. 43–59, 1988.

[30] D. Johnson, G. N. Johnson, S. L. Beggs, C. Botahm, I. Hawker, R. S. K. Chng, M. C. Sinclair, and M. J. O'Mahony, "Distributed restoration strategies in telecommunications networks," in *Proc. IEEE ICC*, 1994, vol. 1, pp. 483–488.

[31] J. P. Jue and B. Mukherjee, "The advantages of partitioning multicast transmissions in a single-hop optical WDM network," in *Proc. ICC '97*, vol. 1, pp. 427–143.

[32] H. Kobrinski and M. Azuma, "Distributed control algorithms for dynamic restoration in DCS mesh networks: Performance evaluation," in *Proc. GLOBECOM'93*, vol. 3, pp. 1584–1588.

[33] S.-B. Kim, "An optimal VP-based multicast routing in ATM networks," in *Proc. INFOCOM'96*, vol. 2, pp. 10c.4.1–10c.4.8.

[34] R. Kawamura, H. Hadama, and I. Tokizawa, "Implementation of self-healing function in ATM networks," *J. Network Syst. Manage.*, vol. 3, no. 3, pp. 243–264, 1995.

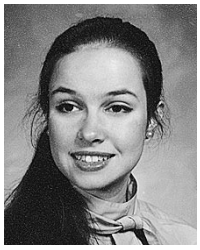
[35] J. Kadiririe and G. Knight, "Comparison of dynamic multicast routing algorithms for wide-area packet-switched (asynchronous transfer mode) networks," in *Proc. INFOCOM'95*, pp. 2c.1.–8.

[36] R. Kawamura, K. Sato, and I. Tokizawa, "High-speed self-healing techniqies utilizing virtual paths," presented at the 5th Int. Network Planning Symp., Kobe, Japan, May 1992.

[37] Y. Kajiyama, N. Tokura, and K. Kikuchi, "ATM self-healing ring," in *Proc. GLOBECOM 92*, pp. 639–643.

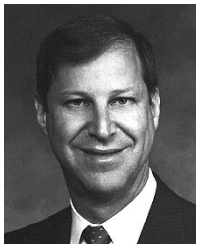
[38] M. T. Lucas, B. J. Dempsey, and A. C. Weaver, "MESH: Distributed error recovery for multimedia streams in wide-area multicast networks," in *Proc. ICC '97*, vol. 2, pp. 1127–1131.

[39] A. Lempel, S. Even, and I. Cederbaum, "An algorithm for planarity testing of graphs," in *Theory of Graphs, In. Symp.*, July 1966, pp. 215–232.
[40] N. D. Lin, A. Zolfaghari, and B. Lusignan, "ATM virtual path self-healing based on a new path restoration protocol," in *Proc. GLOBECOM '94*, vol. 2, pp. 794–798.
[41] P. Mateti and N. Deo, "On algorithms for enumerating all circuits of a graph," *SIAM J. Comput.*, vol. 5, no. 1, Mar. 1976.
[42] M. Médard, S. G. Finn, and R. A. Barry, "Automatic protection switching for multicasting in optical mesh networks," in *Proc. OFC'97*, pp. 314–315.
[43] K. Menger, "Zur allgemeinen kurventheorie," *Fundamental Math.*, vol. 10, pp. 96–115, 1927.
[44] K. Murakami and H. S. Kim, "Virtual path routing for survivable ATM networks," *IEEE/ACM Trans. Networking*, vol. 4, Feb. 1998.
[45] MPLS Web Site: www.ietf.org/html.charters/mpls-charter.html
[46] R. Nakamura, H. Ono, and K. Nishikawara, "Reliable switching services," in *Proc. GLOBECOM'94*, vol. 3, pp. 1596–1600.
[47] D. J. Pai and H. L. Owen, "An algorithm for bandwidth management with survivability constraints in ATM networks," in *Proc. ICC '97*, vol. 1, pp. 261–266.
[48] S. Ramanathan, "Multicast tree generation in networks with asymmetric links," *IEEE/ACM Trans. Networking*, vol. 4, pp. 558–568, Aug. 1996.
[49] J. Shi and J. Fonseka, "Interconnection of self-healing rings," in *Proc. ICC'96*, pp. 1563–1567.
[50] J. B. Slevinsky, W. D. Grover, and M. H. MacGregor, "An algorithm for survivable network design employing multiple self-healing rings," in *Proc. GLOBECOM'93*, vol. 3, pp. 1568–1573.
[51] S. Z. Shaikh, "Span-disjoint paths for physical diversity in networks," in *Proc. IEEE Symp. Computers and Communications*, 1995, pp. 127–133.
[52] Y. Shiloah, "Edge-disjoint branching in directed multigraphs," *Inform. Processing Lett.*, vol. 8, no. 1, Jan. 1979.
[53] H. Sakauchi, Y. Okanoue, H. Okazaki, and S. Hasegawa, "Distributed self-healing control in SONET," *J. Network Syst. Manage.*, vol. 1, no. 2, 1993.
[54] M. W. Slominski and H. Okazaki, "Guided restoration of ATM Cross-Connect networks," in *Proc. ICC 94*, vol. 1, pp. 466–470.
[55] J. Sosnosky, "Service application for SONET DCS distributed restoration," *IEEE J. Select. Areas Commun.*, vol. 12, pp. 59–68, Jan. 1994.
[56] J. W. Suurballe and R. E. Tarjan, "A quick method for finding shortest pairs of disjoint paths," *Networks*, vol. 14, pp. 325–336, 1984.
[57] M. Stoer, *Design of Survivable Networks*. New York: Springer-Verlag, 1992.
[58] C.-C. Shyur, S.-H, Tsao, and Y.-M. Wu, "Survivable network planning methods and tools in Taiwan," *IEEE Commun. Mag.*, vol. 33, Sept. 1995.
[59] J. W. Suurballe, "Disjoint paths in a network," *Networks*, pp. 125–145, 1974.
[60] R. E. Tarjan, "A good algorithm for edge-disjoint branching," *Inform. Processing Lett.*, vol. 3, no. 2, Nov. 1974.
[61] H. Towster, R. Stephenson, S. Morgan, M. Keller, R. Mayer, and R. Shalayda, "Self-healing ring networks: Gateway to public information networking," *IEEE Commun. Mag.*, vol. 28, pp. 54–60, June 1990.
[62] M. Tomizawa, Y. Yamabayashi, N. Kawase, and Y. Kobayashi, "Self-healing algorithm for logical mesh connection on ring networks," *Electron. Lett.*, vol. 30, no. 19, pp. 1615–1616, Sept. 15, 1994.
[63] S. Voß, "Steiner's problem in graphs: Heuristic methods," *Discrete Appl. Mathem.*, vol. 40, pp. 45–72, 1992.
[64] R. Venkateswaran, C. S. Raghavendra, X. Chen, and V. P. Kumar, "Hierarchical multicast routing in ATM networks," in *Proc. ICC'96*, pp. 1690–1694.
[65] J. Veerasamy, S. Vemkatesan, and J. C. Shah, "Effect of traffic splitting on link and path restoration planning," in *Proc. GLOBECOM'94*, vol. 3, pp. 1867–1871.
[66] B. Van Caenegem, N. Wauters, and P. Demeester, "Spare capacity assignment for different restoration strategies in mesh survivable networks," in *Proc. ICC 97*, vol. 1, pp. 288–292.
[67] O. J. Wasem, "Optimal topologies for survivable fiber optic networks using SONET self-healing rings," in *Proc. GLOBECOM'91*, vol. 3, pp. 57.5.1–57.5.7.
[68] B. M. Waxman, "Routing of multipoint connections," *IEEE J. Select. Areas Commun.*, vol. 6, pp. 1617–1622, Dec. 1989.
[69] T.-H. Wu, R. H. Caldwell, and M. Boyden, "A multi-period design model for survivable network architecture selection for SDH/SONET interoffice networks," *IEEE Trans. Rel.*, vol. 40, pp. 417–432, Oct. 1991.
[70] P. Winter, "Steiner tree problem in networks: A survey," *Networks*, vol. 17, pp. 129–167, 1987.
[71] T.-H. Wu, D. J. Kolar, and R. H. Cardwell, "High-speed self-healing ring architectures for future interoffice networks," in *Proc. GLOBECOM'89*, vol. 2, pp. 23.1.1–23.1.7.
[72] J. S. Whalen and J. Kenney, "Finding maximal link disjoint paths in a multigraph," in *Proc. GLOBECOM '90*, pp. 403.6.1–403.6.5.
[73] C.-S. Wu, S.-W. Lee, and Y.-T. Hou, "Backup VP preplanning strategies for survivable multicast ATM networks," in *Proc. ICC '97*, vol. 1, pp. 267–271.
[74] P. Winter and J. MacGregor Smith, "Path-distance heuristics for the Steiner problem in undirected networks," *Algorithmica*, vol 7, no. 2–3, pp. 309–327, 1992.
[75] T.-H. Wu, *Fiber Network Service Survivability*. Norwood, MA: Artech House, 1992.
[76] ———, "A passive protected self-healing mesh network architecture and applications," *IEEE/ACM Trans. Networking*, vol. 2, pp. 40–52, Feb. 1994.
[77] T.-H. Wu and W. I. Way, "A novel passive protected SONET bidirectional self-healing ring architecture," *J. Lightwave Technol.*, vol. 10, Sept. 1992.
[78] Y. Xie, M. J. Lee, and T. N. Saadawi, "Multicasting over ATM using connection server," in *Proc. ICC '97*, vol. 3, pp. 1376–1380.
[79] C. H. Yang and S. Hasegawa, "FITNESS: Failure immunization technology for network service survivability," in *Proc. GLOBECOM'88*, vol. 3, pp. 47.3.1–47.3.6.
[80] B. Yener, Y. Offek, and M. Yung, "Design and performance of convergence routing on multiple spanning trees," in *Proc. GLOBECOM'94*, vol. 1, pp. 169–175.
[81] A. Zehavi and A. Itai, "Three tree-paths," *J. Graph Theory*, vol. 13, no. 2, pp. 175–188, 1989.



Muriel Médard (S'91–M'95) received the B.S. degree in 1989, the M.S. degree in 1991, and the Sc.D. degree in 1995, all from the Massachusetts Institute of Technology (MIT), Cambridge.

She is currently an Assistant Professor in the Electrical and Computer Engineering Department and the Coordinated Science Laboratory, University of Illinois Urbana-Champaign. From 1995 to 1998, she was a Staff Member at MIT Lincoln Laboratory in the Optical Communications and the Advanced Networking Groups. Her interests are in high-speed networking, optical network security, access networks and time-varying wireless channels.



Steven G. Finn (S'70–M'75) received the B.S. degree in 1969, the M.S. degree in 1969, and the Sc.D. degree in 1975, all in electrical engineering, from the Massachusetts Institute of Technology (MIT), Cambridge.

From 1975 to 1980, he was with Codex Corporation, where he held various positions, including Director of Network Processor Research and Development, involving advanced networking products for high-speed data communications networks, and was also a member of the ANSI and CCITT committees, involved in public data networking standards development. In 1980, he founded the Bytex Corporation, where he was CEO through 1987 and Chairman until 1990, when he returned to MIT as a Vinton Hayes Fellow and Visiting Scientist in the Laboratory for Information and Decision Sciences. Currently, he is a Principal Research Scientist and Lecturer in the Electrical Engineering and Computer Science Department at MIT and a Senior Staff Member at the MIT Lincoln Laboratory. He is also a Consultant for Matrix Partners, a venture capital firm. His current research interests are in the areas of optical networks, high-speed data network transport, network architecture, and network management.

Richard A. Barry received the Ph.D. degree in 1993, the M.S. degree in 1989, and the B.S. degree in 1989, all from the Department of Electrical Engineering, Massachusetts Institute of Technology (MIT), Cambridge.

He is Co-Founder and Architecture Director of Sycamore Networks, Inc., Chelmsford, MA, where he is responsible for network and product architecture. From 1994 to 1997, he was with the Advanced Networks Group (formerly the Optical Communications Technology Group) at MIT's Lincoln Laboratory, involving architecture and performance analysis of WDM and OTDM (soliton) optical networks. From 1993 to 1994, he was a Professor of Electrical Engineering and Computer Science at George Washington University, Washington, DC.

Robert G. Gallager (S'58–M'61–F'68–LF'96) has been a Professor of Electrical Engineering at Massachusetts Institute of Technology (MIT) since 1960 and was Co-Director of the MIT Laboratory for Information and Decision Systems from 1986 to 1998. He is the author of the textbooks *Information Theory and Reliable Communication* (New York: Wiley, 1968), *Data Networks* (Englewood Cliffs, NJ: Prentice-Hall, Ed. 2, 1992), and *Discrete Stochastic Processes* (Norwell, MA: Kluwer, 1995). He has published widely and holds a number of patents in the fields of information theory, wireless networks, high-speed data networks, and communication technology.

Dr. Gallager was President of the IEEE Information Theory Society in 1971. He was the Chairman of the Advisory committee to the National Science Foundation Division on Networking and Communication Research and Infrastructure from 1989 to 1992. He is a Member of the National Academy of Engineering, the National Academy of Sciences, and the American Academy of Arts and Sciences. His honors include the IEEE Baker Prize Paper Award (1966), Guggenheim Fellow (1978), IEEE IT Shannon Award (1983), IEEE Centennial Medal (1984), the IEEE Bennet Prize Paper Award (1993), and two Golden Jubilee Paper Awards from the IEEE IT Society in 1998. He was the recipient of the IEEE's highest honor in 1990, the Medal of Honor, awarded for fundamental contributions to communications coding techniques. In 1999, he received the Harvey Prize in Science and Technology from the Technion, Haifa, Israel.