Quality-of-Service and Quality-of-Protection Issues in Preplanned Recovery Schemes Using Redundant Trees

Guoliang Xue, *Senior Member, IEEE,* Li Chen, and Krishnaiyan Thulasiraman, *Fellow, IEEE*

***Abstract*—In this paper, we study quality-of-service (QoS) and quality-of-protection (QoP) issues in redundant tree based preplanned recovery schemes for a single-link failure in two-edge connected graphs and for a single-node failure in two-connected graphs. We present schemes (to be called G-MFBG schemes) that generalize the schemes (to be called MFBG schemes) developed by Médard *et al.* to construct a pair of redundant trees, called red and blue trees, which guarantees fast recovery from any single-link/node failure, as long as the failed node is not the root node. Using the G-MFBG schemes, we study QoS issues relating to red/blue trees. We present effective heuristics for computing a pair of redundant trees with low average delay or small total cost. We develop an optimal algorithm for computing a pair of red/blue trees with maximum bandwidth. Furthermore, a pair of red/blue trees guarantees fast recovery from simultaneous multiple failures if it satisfies certain properties. This leads us to define the concept of QoP of a pair of red/blue trees. We present an effective heuristic to construct a pair of red/blue trees with high QoP. The paper concludes with a discussion of computational results that demonstrate the effectiveness of the different algorithms presented.**

***Index Terms*—Protection and restoration, quality-of-protection (QoP), quality-of-service (QoS), redundant trees.**

I. Introduction

PROTECTION and restoration in high-speed networks is an important issue that has been studied extensively [1], [6], [7], [13]–[15], [17], [19], [20], [22], [25], [26], [30]–[33]. It has important applications in synchronous optical network (SONET) and wavelength-division multiplexing (WDM) networks [1], [4], [5], [10], [18]–[20], [29], [35]. In [15], Médard *et al.* proposed a tree-based preplanned recovery scheme (we will call it the MFBG scheme), which is applicable to any protocol, in particular, WDM, SONET, and ATM, which allows the use of tree routings and redundancy for recovery from failures. For two-connected graphs, the MFBG scheme constructs two directed trees rooted at the root node. One of them, the blue tree, is used as the working tree. The other, the red tree, is used for recovery. When a single node (other than the root node) fails, every other node in the graph is still connected to the root node via either the red tree or the blue tree. For two-edge connected graphs, the MFBG scheme constructs a pair of red/blue trees rooted at the root node. When a single link fails, every node in the graph is still connected to the root node via either the red tree or the blue tree.

In this paper, we first investigate several important measures of quality of preplanned recovery schemes using red/blue trees. First, assuming that each link in the network has a known delay, one design goal is to construct a pair of red/blue trees with minimum average delay in the blue (primary) tree. We present an effective heuristic for constructing such a pair of red/blue trees. Next, assuming that each link in the network has a known cost, another design goal is to construct a pair of red/blue trees with minimum total cost. For this problem, we present an effective heuristic. Finally, assuming that each link in the network has a known bandwidth, one design goal is to construct a pair of red/blue trees with maximum possible bottleneck bandwidth. We present an efficient algorithm for constructing such a pair of red/blue trees. We then define the concept of quality-of-protection (QoP) and present an effective heuristic which constructs a pair of red/blue trees with high QoP. Finally, we discuss generalizations of the MFBG scheme. Computational results are presented to demonstrate the effectiveness of our algorithms and heuristics.

The rest of the paper is organized as follows. In Section II, we introduce concepts and definitions relating to fast recovery using redundant trees and define the optimization problems involving quality-of-service (QoS) and QoP. In Section III, we first present the MFBG scheme of [15] and illustrate this with an example. We then present a scheme, called the G-MFBG scheme, which generalizes the MFBG scheme. In Sections IV and V, we develop effective heuristics for computing a pair of red/blue trees that has low average delay or small total cost, respectively. In Section VI, we present an efficient optimal algorithm for computing a pair of red/blue trees with maximum bandwidth. In Section VII, the problem of maximizing the QoP is considered, and an effective heuristic to compute red/blue trees with good QoP is discussed. In Section VIII, we establish that the G-MFBG scheme generalizes the MFBG scheme and demonstrate this with an example. We also discuss certain properties of the G-MFBG scheme. Computational results are presented in Section IX and the paper is concluded in Section X with future research directions.

Manuscript received January 1, 2003; revised May 20, 2003. This paper was supported in part by ARO under Grant DAAD19-00-1-0377 and in part by the NSF ITR under Grant ANI-0312435 and under Grant ANI-0312635.

G. Xue is with the Department of Computer Science and Engineering at Arizona State University, Tempe, AZ 85287-5406 USA (e-mail: xue@asu.edu; URL: http://www.fulton.asu.edu/~xue).

L. Chen is with the Department of Computer Science, The University of Vermont, Burlington, VT 05405 USA (e-mail: lchen@cs.uvm.edu).

K. Thulasiraman is with the School of Computer Science, University of Oklahoma, Norman, OK 73019 USA (e-mail: thulasi@ou.edu; URL: http://www.cs.ou.edu/~thulasi).

Digital Object Identifier 10.1109/JSAC.2003.816597

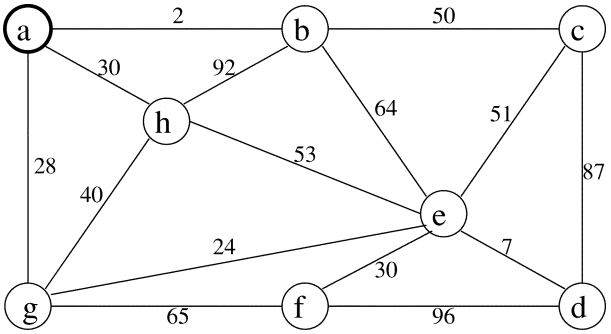

Fig. 1. Graph with edge weights.

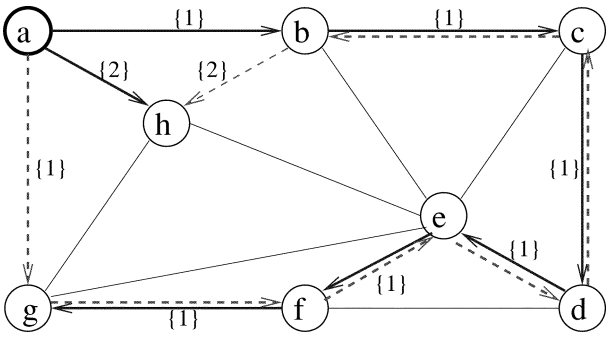

Fig. 2. Blue tree (solid arcs) and red tree (dashed arcs).

II. Fast Recovery Using Redundant Trees

We model a computer network using an undirected graph $G(V, E)$, where V is a set of n *vertices* and E is a set of m *edges*. A vertex represents a computer or a router. An edge represents a communication link. We will use vertex and node interchangeably, as well as edge and link. A graph is *connected* if there is a path connecting any given pair of vertices in the graph. An edge e of G is a *bridge* if there exists a pair of vertices u and v such that every $u - v$ path passes through edge e. A vertex w is an *articulation point* if there exists a pair of vertices u and v ($u \neq w, v \neq w$) such that every $u - v$ path passes through vertex w. A connected graph is *two-edge connected* (or edge-redundant) if it does not contain a bridge. A connected graph is *two connected* (or vertex redundant) if it does not contain an articulation point. We assume that the graph G is either two-edge connected or two connected. We will use $[u, v]$ to denote the *undirected* edge connecting vertices u and v. We will use (u, v) to denote the *directed* edge from vertex u to vertex v. We assume standard graph theoretic notations [24], [28], unless specified otherwise.

In the following, we briefly describe the MFBG scheme for preplanned recovery against single-link (node, respectively) failure in two-edge connected (two connected, respectively) graphs.

Fig. 1 illustrates a sample network with 8 nodes and 15 links. Fig. 2 illustrates two directed trees rooted at the root node a, spanning all the other nodes in the network. The tree with solid edges is the *blue tree* and the tree with dashed edges is the *red tree*. The blue tree is the *working* tree and the red tree is the *backup* tree.

In the redundant tree protocol [7], [13]–[15], a packet from a node u to a node v is transmitted from u to the root node and then from the root node to v. For example, a packet from h to c in Fig. 2 would go from h to a and then from a to c. If there is no link or node failure, each node in the network is connected to the root node in the blue tree (and in the blue tree). As a result, a packet from h to c would go from h to a, and then from a to b to c, all in the blue tree.

When a single-link failure occurs, say at link $[b, c]$, nodes c, d, e, f, g are no longer connected to a via the blue tree. However, they are connected to a via the red tree. The packet from h to c would go from h to a (via the blue tree), and then from a to g to f to e to d to c (via the red tree). When a single-node failure occurs, say at node b, none of the remaining nodes (except node h) is connected to a in the blue tree. They are, however, connected to a in the red tree. The packet from h to c would go from h to a (via the blue tree), and then from a to g to f to e to d to c (via the red tree). The precomputed red/blue trees enable ultra fast recovery from single-link/node failure using automatic protection switching (APS).

Definition 1: Let $G(V, E, s)$ be an undirected graph with edge set E and vertex set V, where $s \in V$ is a distinguished *root*. Let T^B and T^R be a pair of directed trees such that there is a directed path p_v^B in T^B from s to every vertex $v \in V$ and a directed path p_v^R in T^R from s to every vertex $v \in V$.

1) T^B and T^R form a pair of single-node recovery trees if for any chosen vertex $u \neq s \in V$ and any $w \neq u \in V, p_w^B$ and p_w^R do not both contain node u.
2) T^B and T^R form a pair of single-link recovery trees if for any chosen edge $[u, v] \in E$ and any $w \in V, p_w^B$ and p_w^R do not both use edge $[u, v]$.

We will use recovery trees to denote either a pair of single-node recovery trees or a pair of single-link recovery trees when the exact meaning can be derived from the context. We will use p_v^B and p_v^R to denote the two paths discussed above throughout this paper.

Assume that each link $[u, v]$ also has a nonnegative *delay*, denoted by $d(u, v)$. Let T^B and T^R be a pair of recovery trees rooted at node s. Then, for each node $v \in V$, there is a *delay from s to v in T^B* [denoted by $d_{T^B}(s, v)$], and a *delay from s to v in T^R* [denoted by $d_{T^R}(s, v)$], which are the sum of the delays of the edges on the directed path from s to v in T^B and T^R, respectively. We define the *average delay* of T^B as $d^{\text{avg}}(T^B) = (1)/(|V| - 1) \sum_{v \in V} d_{T^B}(s, v)$. Therefore, $2 \cdot d^{\text{avg}}(T^B)$ is the average delay between a pair of nodes sustained in the redundant tree protocol, in the absence of any failure.

The red/blue trees in Fig. 2 form a pair of single-node recovery trees. So do the red/blue trees in Fig. 3. Assume that the edge labels in Fig. 1 represent edge delays. Then, the average delay of the blue tree in Fig. 2 is $(30+65+30*2+7*3+87*4+50*5+2*6)/7 = 112.29$ and the average delay of the red tree in Fig. 2 is $(92+50*2+87*3+7*4+30*5+65*6+28*7)/7 = 173.86$. The average delay of the blue tree in Fig. 3 is 62. Therefore, the pair of recovery trees in Fig. 3 is better than the pair of recovery trees in Fig. 2 in terms of average delays in the working tree. These discussions lead to the following optimization problems.

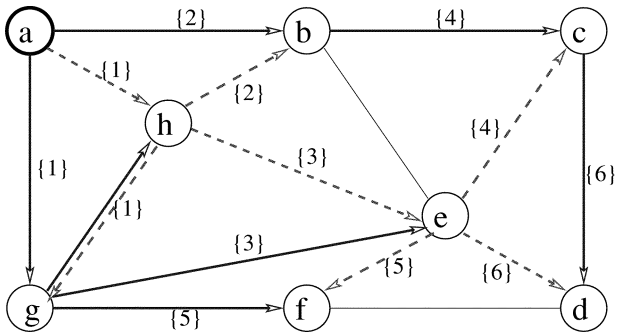

Fig. 3. Recovery trees with lower average delays.

Definition 2: Let $G(V, E, s)$ be a graph where each edge has a positive delay and s is the root node. A pair of single-node recovery trees T^B and T^R is called a pair of min-delay single-node recovery trees if the average delay of T^B is minimum among all single-node recovery trees. The MinDelayV problem asks for a pair of min-delay single-node recovery trees. A pair of single-link recovery trees T^B and T^R is called a pair of min-delay single-link recovery trees if the average delay of T^B is minimum among all single-link recovery trees. The MinDelayE problem asks for a pair of min-delay single-link recovery trees. We use *min-delay recovery trees* to mean either a pair of min-delay single-node recovery trees or a pair of min-delay single-link recovery trees.

It is challenging to compute min-delay recovery trees in general. In Section IV, we will present an effective heuristic algorithm which computes a pair of recovery trees with low average delay in the working tree. The worst-case time complexity of our heuristics is $O(n^2(m + n \log n))$.

Assume that each link $[u, v]$ has a nonnegative *cost*, denoted by $d(u, v)$. Let T^B and T^R be a pair of recovery trees rooted at node s. The total cost of T^B and T^R, denoted by $c(T^B, T^R)$, is the sum of the edge costs over edges that are used by at least one of the two trees. $c(T^B, T^R)$ reflects the network usage by the pair of recovery trees. Therefore, it is desirable to design recovery trees with small cost.

Definition 3: Let $G(V, E, s)$ be a graph where each edge has a nonnegative cost and s is the root node. A pair of single-node recovery trees T^B and T^R is called a pair of min-cost single-node recovery trees if the total cost of T^B and T^R is minimum among all single-node recovery trees. The MinCostV problem asks for a pair of min-cost single-node recovery trees. A pair of single-link recovery trees T^B and T^R is called a pair of min-cost single-link recovery trees if the total cost of T^B and T^R is minimum among all single-link recovery trees. The MinCostE problem asks for a pair of min-cost single-link recovery trees. We will use the term *min-cost recovery trees* to mean either a pair of min-cost single-node recovery trees or a pair of min-cost single-link recovery trees.

In Section V, we will present effective heuristics, which compute a pair of recovery trees with small total cost. The worst-case time complexity of our algorithms is $O(n^2(m+n))$, which is a factor of n higher than that of the MFBG scheme [15].

Assume that each link $[u, v]$ also has a positive *bandwidth*, denoted by $b(u, v)$. Let T^B and T^R be a pair of recovery trees rooted at node s. We define the *bandwidth* of T^B as $b(T^B) = \min_{(u,v)\in T^B} b(u, v)$ and the *bandwidth* of T^R as $b(T^R) = \min_{(u,v)\in T^R} b(u, v)$. We define the *bottleneck bandwidth* of the pair of recovery trees as $b(T^B, T^R) = \min\{b(T^B), b(T^R)\}$.

Definition 4: Let $G(V, E, s)$ be a graph where each edge has a positive bandwidth and s is the root node. A pair of single-node recovery trees T^B and T^R is called a pair of max-bandwidth single-node recovery trees if the bottleneck bandwidth of T^B and T^R is maximum among all single-node recovery trees. The MaxBandV problem asks for a pair of max-bandwidth single-node recovery trees. A pair of single-link recovery trees T^B and T^R is called a pair of max-bandwidth single-link recovery trees if the bottleneck bandwidth of T^B and T^R is maximum among all single-link recovery trees. The MaxBandE problem asks for a pair of max-bandwidth single-link recovery trees. We will use the term *max-bandwidth recovery trees* to mean either a pair of max-bandwidth single-node recovery trees or a pair of max-bandwidth single-link recovery trees.

Unlike the optimization problems with QoS issues related to delay and cost, where we could only present *heuristics* to find *suboptimal* solutions, the max-bandwidth recovery trees problem can be solved efficiently. In Section VI, we will present an efficient algorithm which computes a pair of recovery trees with maximum bottleneck bandwidth. The worst-case time complexity of our algorithms is $O(nm)$, which is the same as that of the MFBG scheme [15].

Although the recovery trees are designed for recovery from a single-link/node failure, they can be used to recover from certain simultaneous multiple failures as well. For example, the recovery trees shown in Fig. 2 can survive simultaneous failures of links $[b, c]$ and $[a, h]$. When such simultaneous failures occur, node b is still connected to the root node a via the blue tree, nodes c, d, e, f, g are all connected to the root node via the red tree, node h is connected to node b via the red tree, which is then connected to the root node via the blue tree. However, the recovery trees shown in Fig. 2 cannot survive simultaneous failures of three or more links (in the blue tree and/or the red tree).

Similarly, the recovery trees shown in Fig. 3 can survive simultaneous failures of links $[g, h], [a, b], [e, h], [b, c], [f, g], [c, d]$. When such simultaneous failures occur, nodes b, c, d, e, f, g, h can be connected to the root node a via the spanning tree consisting of the links $[a, g], [a, h], [g, e], [e, c], [e, d], [e, f], [h, b]$. However, the recovery trees shown in Fig. 3 cannot survive simultaneous failures of seven or more links (in the blue tree and/or the red tree). These discussions lead to the following optimization problem.

Definition 5: Let T^B and T^R be a pair of single-link recovery trees. The quality-of-protection (QoP) of T^B and T^R is defined as the maximum integer k such that there exists an instance of k simultaneous link failures that T^B and T^R can survive. The MaxQoPE problem asks for a pair of single-link recovery trees with maximum QoP.

We note that the pair of single-link recovery trees shown in Fig. 3 has a higher QoP than the pair of single-failure recovery trees shown in Fig. 2.

Next, we define the QoP of a pair of single-node recovery trees. Whereas the removal of any link used by the blue tree (red tree, respectively) disconnects the blue tree (red tree, respectively), removal of certain nodes (for instance, the leaf nodes) will not disconnect the blue tree (red tree, respectively). This situation must be taken care of in the definition of QoP for single-node recovery trees. Toward this end, we first define a *cut* S (with respect to T^B and T^R) as a set of nodes whose removal disconnects the blue tree. A cut S is called *critical* if every proper subset S_1 of S is also a cut, but the removal of S_1 breaks the blue tree into fewer subtrees than the removal of S. We shall also refer to a critical cut as a critical set of node failures.

Definition 6: Let T^B and T^R be a pair of single-node recovery trees. The QoP of T^B and T^R is defined as the maximum integer k such that there exists a critical cut S of size k that T^B and T^R can survive.

III. MFBG Schemes and Generalizations

For ease of discussion, we first give a formal description of the MFBG protection schemes for node recovery and link recovery. We then present schemes that generalize the MFBG schemes. These generalized schemes are used in heuristics and algorithms presented in the following sections.

Algorithm 1: MFBG-V

step_1 Initialize T^B and T^R to contain the root node s only. Assign node s both a blue voltage $v^B(s) > 0$ and a red voltage $v^R(s) = 0$.

step_2 Find a cycle $[s, v_1, \ldots, v_k, s]$ with $k \geq 2$. Let $s \rightarrow v_1 \rightarrow \cdots \rightarrow v_k$ be the blue chain and $s \rightarrow v_k \rightarrow \cdots \rightarrow v_1$ be the red chain. Add the blue chain to T^B and the red chain to T^R. Assign blue voltages at the new nodes in this cycle such that $v^B(s) > v^B(v_1) > \cdots > v^B(v_k) > v^R(s)$.

step_3 if T^B spans all the nodes in G, stop.

step_4 Find a path $[x, v_1, \ldots, v_k, y]$ connecting two distinct nodes x and y on T^B and $k \geq 1$ nodes not on T^B such that $v^B(x) > v(y)$ where $v(y) = v^B(y)$ unless $y = s$, in the latter case $v(y)$ is set to $v^R(s)$.
Let $x \rightarrow v_1 \rightarrow \cdots \rightarrow v_k$ be the blue chain and $y \rightarrow v_k \rightarrow \cdots \rightarrow v_1$ be the red chain. Add the red chain to T^R and the blue chain to T^B.
Let $v^M(x)$ be the maximum of all voltages that are lower than $v^B(x)$. Assign blue voltages at these new nodes on this path such that $v^B(x) > v^B(v_1) > \cdots > v^B(v_k) > v^M(x)$. goto Step_3.

The MFBG scheme for node recovery is listed as Algorithm 1 and the MFBG scheme for link recovery is listed as Algorithm 2. In these schemes, each tree node u is assigned a *blue voltage* $v^B(u)$. In some cases, u is also assigned a *red voltage* $v^R(u)$. The elegant voltage techniques are introduced by [15].

Note that in both MFBG-V and MFBG-E, *a cycle starts out with a node on the tree, passes through $k \geq 2$ nodes not on the tree, and returns to a different node on the tree*. We will assume this rule about cycles and paths (without specifically saying it) in all algorithmic descriptions in the rest of this paper.

Algorithm 2: MFBG-E

step_1 Initialize T^B and T^R to contain the root node s only. Assign node s both a blue voltage $v^B(s) > 0$ and a red voltage $v^R(s) = 0$.

step_2 Find a cycle $[s, v_1, \ldots, v_k, s]$ with $k \geq 2$. Let $s \rightarrow v_1 \rightarrow \cdots \rightarrow v_k$ be the blue chain and $s \rightarrow v_k \rightarrow \cdots \rightarrow v_1$ be the red chain. Add the blue chain to T^B and the red chain to T^R. Assign voltages at the new nodes in this cycle such that $v^B(s) > v^B(v_1) > v^R(v_1) > \cdots > v^B(v_k) > v^R(v_k) > v^R(s)$.

step_3 if T^B spans all the nodes in G, stop.

step_4 Find a path $[x, v_1, \ldots, v_k, y]$ connecting two (not necessarily distinct) nodes x and y on T^B and $k \geq 1$ nodes not on T^B such that $v^B(x) \geq v(y)$ where $v(y) = v^B(y)$ unless $y = s$, in the latter case $v(y)$ is set to $v^R(s)$.
Let $x \rightarrow v_1 \rightarrow \cdots \rightarrow v_k$ be the blue chain and $y \rightarrow v_k \rightarrow \cdots \rightarrow v_1$ be the red chain. Add the red chain to T^R and the blue chain to T^B.
Let $v^M(x)$ be the maximum of all voltages that are lower than $v^B(x)$. Assign blue voltages at these new nodes on this path such that $v^B(x) > v^B(v_1) > v^R(v_1) > \cdots > v^B(v_k) > v^R(v_k) > v^M(x)$. goto Step_3.

The MFBG schemes grow the red tree and blue tree gradually by finding a path or a cycle connecting a node already on the tree to another node (or the same node in the case of a cycle) via nodes not on the tree. Once a path or cycle is found, we grow the red tree and the blue tree with the help of the voltages. As proved in [15], the voltage rule guarantees the correctness of the algorithms.

As an example, we explain how Algorithm 1 constructs the pair of red/blue trees in Fig. 2. Node a is chosen as the root node. In Step_2, the algorithm finds the cycle $[a, b, c, d, e, f, g, a]$. It assigns the voltages and constructs the tree edges with label $\{1\}$ in the figure. In Step_4, the algorithm finds the path $[a, h, b]$ [note that $v^B(a) > v^B(b)$]. Note that $v^M(a)$ equals $v^B(b)$ at this time. It assigns the voltage for h such that $v^B(a) > v^B(h) > v^M(a)$.

The voltage rule of MFBG-V imposes a *complete order* on the $n+1$ voltages $\{v^B(v) \mid v \in V\} \cup \{v^R(s)\}$. Adding an artificial node s^R to the n nodes in G, we can establish a one-to-one correspondence between the $n+1$ nodes and the $n+1$ voltages, with s^R corresponding to $v^R(s)$ and v corresponding to $v^B(v)$ for every nonartificial node v. Therefore, the pair of red/blue trees imposes a *partial order* $\prec$ on the $n+1$ nodes by the following rules.

PV1 : If (u, v) is a directed edge from u to v in T^B, then $v \prec u$.

PV2 : If (u, v) is a directed edge from u to v in T^R with $u \neq s$, then $u \prec v$.

PV3 : If (s, v) is a directed edge from s to v in T^R, then $s^R \prec v$.

It turns out that this partial order is sufficient to guarantee the construction of a pair of single-node failure recovery trees. This leads us to present a modified MFBG scheme for single-node recovery as Algorithm 3.

Algorithm 3: G-MFBG-V

step_1 Initialize T^B and T^R to contain the root node s only. Initialize the partial order on the $n+1$ nodes with the precedence relation $s^R \prec s$.

step_2 Find a cycle $[s, v_1, \ldots, v_k, s]$.
Let $s \to v_1 \to \cdots \to v_k$ be the blue chain and $s \to v_k \to \cdots \to v_1$ be the red chain. Add the red chain to T^R and the blue chain to T^B.
Use the precedence relations $s^R \prec v_k \prec v_{k-1} \prec \cdots \prec v_1 \prec s$ to augment the partial order on the $n+1$ nodes. Note that $u \prec v$ and $v \prec w$ imply $u \prec w$.

step_3 if T^B spans all the nodes in G, stop.

step_4 Find a path $[x, v_1, \ldots, v_k, y]$ connecting two distinct nodes x and y on T^B and $k \geq 1$ nodes not on T^B such that either $y = s$ or $x \neq y$.
Let $x \to v_1 \to \cdots \to v_k$ be the blue chain and $y \to v_k \to \cdots \to v_1$ be the red chain. Add the red chain to T^R and the blue chain to T^B.
If $y = s$, we will use y to denote s^R in the following. Use the precedence relations $y \prec v_k \prec v_{k-1} \prec \cdots \prec v_1 \prec x$ to augment the partial order on the $n+1$ nodes.
goto Step_3.

Similarly, one can show that the voltage rule for MFBG-E defines a complete order on the set of (undirected) edges of G that are used by T^B or T^R. As in the case of node recovery, a carefully designed partial order is sufficient. For this purpose, we define a partial order $\prec$ on the (undirected) edges of the graph G using the red/blue trees T^R and T^B in the following way.

PE1 : Let $[u, v]$ and $[v, w]$ be two-undirected edges of G such that for some node w, p_w^B passes edge $[u, v]$ before passing edge $[v, w]$. Then, $[v, w] \prec [u, v]$.

PE2 : Let $[u, v]$ and $[v, w]$ be two-undirected edges of G such that for some node w, p_w^R passes edge $[u, v]$ before passing edge $[v, w]$. Then, $[u, v] \prec [v, w]$.

PE3 : Let $[u, w]$ and $[v, w]$ be two-undirected edges of G such that p_w^B passes edge $[u, w]$ and that p_w^R passes edge $[v, w]$. Then, $[v, w] \prec [u, w]$.

Algorithm 4: G-MFBG-E

step_1 Initialize T^B and T^R to contain the root node s only. Initialize the partial order on the m edges to be the empty set.

step_2 Find a cycle $[s, v_1, \ldots, v_k, s]$.
Let $s \to v_1 \to \cdots \to v_k$ be the blue chain and $s \to v_k \to \cdots \to v_1$ be the red chain. Add the red chain to T^R and the blue chain to T^B.
Use the precedence relations $[s, v_k] \prec [v_k, v_{k-1}] \prec \cdots \prec [v_2, v_1] \prec [v_1, s]$ to augment the partial order on the m edges. Note that $e_1 \prec e_2$ and $e_2 \prec e_3$ imply $e_1 \prec e_3$.

step_3 if T^B spans all the nodes in G, stop.

step_4 Find a path $[x, v_1, \ldots, v_k, y]$ connecting two (not necessarily distinct) nodes x and y on T^B and $k \geq 1$ nodes not on T^B such that either one of x and y is the root node s or $e^B(x) \neq e^R(y)$, where $e^B(u)$ ($e^R(u)$, respectively) denotes the undirected edge of G which is adjacent with u and is on the $s - u$ path in T^B (T^R, respectively).
Let $x \to v_1 \to \cdots \to v_k$ be the blue chain and $y \to v_k \to \cdots \to v_1$ be the red chain. Add the red chain to T^R and the blue chain to T^B.
Use the precedence relations $[y, v_k] \prec [v_k, v_{k-1}] \prec \cdots \prec [v_2, v_1] \prec [v_1, x]$ to augment the partial order on the m edges.
goto Step_3.

Our modified MFBG scheme for link recovery based on the above partial order is presented as Algorithm 4. Formal proofs establishing that Algorithms 3 and 4 generalize Algorithms 1 and 2, respectively, and other illustrations relating to this generalization will be given in Section VIII.

IV. Reducing the Average Delay

In this section, we present an effective heuristic for constructing a pair of single-failure recovery trees with low average delay. Algorithm 5 constructs a pair of such single-node recovery trees. Recall that $d_{T^B}(s, x)$ is the $s - x$ delay in T^B. During each iteration of Algorithm 5, we find a path (a cycle in the first iteration) such that the maximum $s - v$ delay in T^B (among the newly added nodes v) is as small as possible after the addition of the blue chains and the red chains.

Algorithm 5: MinDelayV

step_1 Initialize T^B and T^R to contain the root node s only. Initialize the partial order as in Algorithm 3.

step_2 Find a cycle $(s, v_1, v_2, \ldots, v_k, s)$ such that the delay of this cycle minus the delay of its last edge (v_k, s) is minimum among all such cycles.
Let $s \to v_1 \to \cdots \to v_k$ be the blue chain and $s \to v_k \to \cdots \to v_1$ be the red chain. Add the red chain to T^R and the blue chain to T^B.
Augment the partial order as in Algorithm 3.

step_3 If T^B spans all the nodes in G, stop.

step_4 Find a path $(x, v_1, \ldots, v_k, y)$ connecting two distinct nodes x and y on T^B and $k \geq 1$ nodes not on T^B such that either $y = s$ or $x \neq y$ and that $d_{T^B}(s, x)$ plus the $x - v_k$ delay in the found path is minimum among all such paths.
Let $x \to v_1 \to \cdots \to v_k$ be the blue chain and $y \to v_k \to \cdots \to v_1$ be the red chain. Add the red chain to T^R and the blue chain to T^B.
Augment the partial order as in Algorithm 3.
goto Step_3.

Theorem 1: The heuristic MinDelayV terminates with a pair of single-node recovery trees if the graph is two connected. The worst-case running time of the heuristic is $O(n^2(m + n \log n))$.

Proof: The correctness and finite termination of the heuristic follows from Theorem 7 to be proved in Section VIII.

Since the set of cycles or paths is nonempty, we can always find one which minimizes our objective function. Therefore, the heuristic terminates after a finite number of steps if the graph is two connected.

Now, we explain an $O(n^2(m + n\log n))$ time implementation of the heuristic. There are $O(n)$ choices for v_k in a cycle of form $[s, v_1, v_2, \ldots, v_k, s]$. For each chosen v_k, we can hide edge $[v_k, s]$ and compute a minimum delay path from v_k to s in $O(m + n\log n)$ time. Therefore, the worst-case time complexity of Step_2 is $O(n(m + n\log n))$.

Each time a path is computed in Step_4, MinDelayV adds at least one node to both T^B and T^R. Therefore, Step_4 is executed at most $O(n)$ times. We will show that the worst-case time complexity of each execution of Step_4 is $O(n(m + n\log n))$. There are $O(n)$ possible choices of node y. Assume that y is chosen. Let $v_{11}, v_{12}, \ldots, v_{1l}$ be all the neighboring nodes of y that are not already on T^B. Let v_0 be a new (artificial) node which is connected to $v_{11}, v_{12}, \ldots, v_{1l}$ by zero delay edges $[v_0, v_{11}], [v_0, v_{12}], \ldots, [v_0, v_{1l}]$. Hide node y. Hide all the nodes u in T^B such that $u \prec y$. Then, we obtain a new graph G_y from the original graph G. Find a constrained shortest path (if none exists, the cost is assumed to be infinity) from s to v_0 in G_y such that the first portion (closer to s) of the path consists of only edges in T^B that were not hidden in the above hiding process and the other portion (closer to v_0) does not contain any nodes in T^B. Replacing node v_0 on this path by y produces an $x - y$ path, which we denote by π_y. Remove node v_0 from the graph and unhide all the hidden nodes or edges. π_y can be computed in $O(m + n\log n)$ time using a modification of Dijkstra's algorithm. We loop y over all of the nodes on the tree T^B and choose the best π_y (path delay minus delay of the last edge is minimized). Therefore, Step_4 can be implemented in $O(n(m + n\log n))$ time. This shows that the worst-case time complexity of the heuristic is $O(n^2(m + n\log n))$. □

Let us illustrate MinDelayV with the sample network shown in Fig. 1, assuming that the edge labels are the edge delays. Also assume that node a is the root node.

In Step_2 of MinDelayV, we find the cycle $[a, g, h, a]$. We label all the edges on this cycle with $\{1\}$, indicating that they are added to the red/blue trees during the first iteration. The blue chain is $a \to g \to h$ and the red chain is $a \to h \to g$.

The first time Step_4 is executed, we find the path $[a, b, h]$. The blue chain is $a \to b$ and the red chain is $h \to b$. The edges on this path are labeled with $\{2\}$ because they are selected during the second iteration. The second time Step_4 is executed, we find the path $[g, e, h]$. The blue chain is $g \to e$ and the red chain is $h \to e$. The edges on this path are labeled with $\{3\}$ because they are selected during the third iteration. The third time Step_4 is executed, we find the path $[b, c, e]$. The blue chain is $b \to c$ and the red chain is $e \to c$. The edges on this path are labeled with $\{4\}$ because they are selected during the forth iteration. The fourth time Step_4 is executed, we find the path $[g, f, e]$. The blue chain is $g \to f$ and the red chain is $e \to f$. The edges on this path are labeled with $\{5\}$ because they are selected during the fifth iteration. The fifth time Step_4 is executed, we find the path $[c, d, e]$. The blue chain is $c \to d$ and the red chain is $e \to d$. The edges on this path are labeled with $\{6\}$ because they are selected during the sixth iteration.

At this time, we have found a pair of single-node recovery trees illustrated in Fig. 3. Note that the average delay of the working (blue) tree blue is 62, which is smaller than the average delay of T^B in Fig. 2 (which is 112.29) and the average delay of T^R in Fig. 2 (which is 173.86).

Our heuristic for constructing a pair of single-link recovery trees with low average delay in the working tree is presented in Algorithm 6.

Algorithm 6: MinDelayE

step_1 Initialize T^B and T^R to contain the root node s only. Initialize the partial order as in Algorithm 4.

step_2 Find a cycle $[s, v_1, v_2, \ldots, v_k, s]$ such that the delay of this cycle minus the delay of its last edge (v_k, s) is minimum among all such cycles. Let $s \to v_1 \to \cdots \to v_k$ be the blue chain and $s \to v_k \to \cdots \to v_1$ be the red chain. Add the red chain to T^R and the blue chain to T^B. Augment the partial order as in Algorithm 4.

step_3 If T^B spans all the nodes in G, stop.

step_4 Find a path $[x, v_1, \ldots, v_k, y]$ connecting two (not necessarily distinct) nodes x and y on T^B such that either one of x and y is the root node s or $e^B(x) \not\prec e^R(y)$ and that $d_{T^B}(s, x)$ plus the $x - v_k$ delay in the found path is minimum among all such paths.
Let $x \to v_1 \to \cdots \to v_k$ be the blue chain and $y \to v_k \to \cdots \to v_1$ be the red chain. Add the red chain to T^R and the blue chain to T^B. Augment the partial order as in Algorithm 4.
goto Step_3.

Theorem 2: The heuristic MinDelayE terminates with a pair of single-link recovery trees if the graph is two-edge connected. The worst-case running time of the heuristic is $O(n^2(m + n\log n))$.

Proof: The correctness and finite termination follows from Theorem 9. The time complexity can be analyzed similarly as in the Proof of Theorem 1. □

A couple of closely related heuristics for delay reduction in red/blue trees are presented in [31]. However, the ordering rules in [31] are too restrictive. The ordering rules in MinDelayV and MinDelayE are more general and, therefore, can lead to better performance. Computational results for MinDelayV and MinDelayE will be presented in Section IX.

V. Reducing Total Cost

If hardware usage is a major concern, one would like to construct a pair of single-failure recovery trees with minimum total cost. However, finding a pair of red/blue trees with minimum total cost is difficult [15]. In this section, we will present a pair of effective heuristics for constructing a pair of single-failure recovery trees with small total cost.

Recall that initially only the root node s is on the trees. By selecting a cycle $[x, v_1, \ldots, v_k, x]$, we add k nodes to the trees at a cost of $c(x, v_1) + c(v_1, v_2) + \cdots + c(v_k, x)$. By selecting a path $[x, v_1, \ldots, v_k, y]$, we add k nodes to the trees at a cost of $c(x, v_1) + c(v_1, v_2) + \cdots + c(v_k, y)$. Therefore, we define the

scaled cost of cycle $[x, v_1, \ldots, v_k, x]$ as $(c(x, v_1) + c(v_1, v_2) + \cdots + c(v_k, x))/k$ and the scaled cost of path $[x, v_1, \ldots, v_k, y]$ as $(c(x, v_1) + c(v_1, v_2) + \cdots + c(v_k, y))/k$. Algorithm 7 (Algorithm 8, respectively) presented below finds a path (path or cycle, respectively) with low scaled cost at each step of the construction of red/blue trees.

Note that finding a path or cycle with minimum scaled cost is not easy. For both heuristics, we use depth first search (DFS) to find a set of feasible cycles and paths and choose the one with the lowest scaled cost.

Similar to the case of delay reduction heuristics, we can prove the following theorems. The time complexity is $O(n^2(m+n))$ instead of $O(n^2(m + n\log n))$ because we are using DFS instead of shortest paths.

Theorem 3: The heuristic MinCostV terminates with a pair of single-node recovery trees if the graph is two connected. The worst-case running time of the heuristic is $O(n^2(m+n))$. □

Theorem 4: The heuristic MinCostE terminates with a pair of single-link recovery trees if the graph is two-edge connected. The worst-case running time of the heuristic is $O(n^2(m+n))$.□

Algorithm 7: MinCostV

step_1 Initialize T^B and T^R to contain the root node s only. Initialize the partial order as in Algorithm 3.

step_2 Find a cycle $[s, v_1, \ldots, v_k, s]$ in the graph with low scaled cost.
Let $s \to v_1 \to \cdots \to v_k$ be the blue chain and $s \to v_k \to \cdots \to v_1$ be the red chain. Add the red chain to T^R and the blue chain to T^B.
Augment the partial order as in Algorithm 3.

step_3 if T^B spans all the nodes in G, stop.

step_4 Find a path $[x, v_1, \ldots, v_k, y]$ with low scaled cost connecting two distinct nodes x and y on T^B and $k \geq 1$ nodes not on T^B such that either $y = s$ or $x \neq y$.
Let $x \to v_1 \to \cdots \to v_k$ be the blue chain and $y \to v_k \to \cdots \to v_1$ be the red chain. Add the red chain to T^R and the blue chain to T^B.
Augment the partial order as in Algorithm 3.
goto Step_3.

Algorithm 8: MinCostE

step_1 Initialize T^B and T^R to contain the root node s only. Initialize the partial order as in Algorithm 4.

step_2 Find a cycle $[s, v_1, \ldots, v_k, s]$ with low scaled cost.
Let $s \to v_1 \to \cdots \to v_k$ be the blue chain and $s \to v_k \to \cdots \to v_1$ be the red chain. Add the red chain to T^R and the blue chain to T^B.
Augment the partial order as in Algorithm 4.

step_3 if T^B spans all the nodes in G, stop.

step_4 Find a path $[x, v_1, \ldots, v_k, y]$ with low scaled cost connecting two (not necessarily distinct) nodes x and y on T^B and $k \geq 1$ nodes not on T^B such that either one of x and y is the root node s or $e^B(x) \neq e^R(y)$.
Let $x \to v_1 \to \cdots \to v_k$ be the blue chain and $y \to v_k \to \cdots \to v_1$ be the red chain. Add the red chain to T^R and the blue chain to T^B.
Augment the partial order as in Algorithm 4.
goto Step_3.

VI. Maximizing Bottleneck Bandwidth

In this section, we present an efficient algorithm for constructing a pair of red/blue trees with maximum bottleneck bandwidth. For any positive C, define $G(C)$ to be the subgraph of G with only those edges whose bandwidth is C or larger. Our algorithm first uses bisection on the different bandwidth values to find the largest bandwidth value B such that $G(B)$ is two connected (two-edge connected). It then applies the corresponding MFBG scheme to $G(B)$ to construct a pair of single-node (single-link) recovery trees. The algorithm is formally stated in Algorithm 9.

Algorithm 9: MaxBand

step_1 Use bisection on the $O(m)$ bandwidth values to find the largest B such that $G(B)$ is 2-connected (2-edge connected).

step_2 Apply G-MFBG-V (G-MFBG-E, respectively) on $G(B)$ to construct a pair of single node (link) recovery trees.

Theorem 5: The time complexity of Algorithm 9 is $O(mn)$. If G is two connected, Algorithm 9 constructs a pair of single-node recovery trees with maximum bottleneck bandwidth. If G is two-edge connected, Algorithm 9 constructs a pair of single-link recovery trees with maximum bottleneck bandwidth.

Proof: The G-MFBG-V scheme can construct a pair of single-node recovery trees if and only the graph is two connected. The G-MFBG-E scheme can construct a pair of single-link recovery trees if and only the graph is two-edge connected. This proves the correctness of the algorithm.

Using Tarjan's DFS technique [23], we can test whether a given graph is two connected (two-edge connected) in linear time. Using the selection technique, the median can be found in linear time [2]. Since the maximum number of bisections is $O(\log n)$, the time complexity of Algorithm 9 is $O(mn)$. □

Note that MaxBand is an *optimal algorithm*, which is different from the *heuristic* presented in [30].

VII. Enhancing Quality of Protection

As we have seen from the heuristics in the previous section, the pair of single-failure recovery trees can be constructed by choosing an initial cycle and then adding paths and cycles iteratively until all nodes are covered. The QoP of the pair of single-link recovery trees equals the total number of cycles and and paths used in this construction process. Therefore, it is desirable to construct a pair of single-link recovery trees using more paths and cycles.

Since our goal is to enhance the QoP, we can choose the root node to our advantage. Such an algorithm is presented next.

Algorithm 10: QoPE

step_1 Initialize T^B and T^R to contain the root node s only. Initialize the partial order on the m edges to be the empty set.

Step_2 Find a cycle $[s, c_1, c_2, \ldots, c_k, s]$ with minimum hop count and $k \geq 2$.
Let $s \rightarrow v_1 \rightarrow \cdots \rightarrow v_k$ be the blue chain and $s \rightarrow v_k \rightarrow \cdots \rightarrow v_1$ be the red chain. Add the red chain to T^R and the blue chain to T^B.
Use the precedence relations $[s, v_k] \prec [v_k, v_{k-1}] \prec \cdots \prec [v_2, v_1] \prec [v_1, s]$ to augment the partial order on the m edges. Note that $e_1 \prec e_2$ and $e_2 \prec e_3$ imply $e_1 \prec e_3$.

Step_3 If T^B spans all the nodes in G, stop.

Step_4 Find a minimum hop path $[x, v_1, \ldots, v_k, y]$ connecting two (not necessarily distinct) nodes x and y on T^B and $k \geq 1$ nodes not on T^B such that either one of x and y is the root node s or $e^B(x) \neq e^R(y)$, where $e^B(u)$ ($e^R(u)$, respectively) denotes the undirected edge of G which is adjacent with u and is on the s-u path in T^B (T^R, respectively).
Let $x \rightarrow v_1 \rightarrow \cdots \rightarrow v_k$ be the blue chain and $y \rightarrow v_k \rightarrow \ldots \rightarrow v_1$ be the red chain. Add the red chain to T^R and the blue chain to T^B.
Use the precedence relations $[y, v_k] \prec [v_k, v_{k-1}] \prec \cdots \prec [v_2, v_1] \prec [v_1, x]$ to augment the partial order on the m edges.
goto Step_3.

Theorem 6: Algorithm 10 constructs a pair of single-link recovery trees provided that G is two connected. The time complexity of Algorithm 10 is $O(n^2 m)$.

Proof: The correctness of the algorithm follows from the correctness of the G-MFBG-E scheme.

Step_2 takes $O(nm)$ time, because finding a minimum hop cycle *including a particular node* requires $O(n+m)$ time using breadth first search (BFS). Step_4 takes $O(nm)$ time, because finding a minimum hop path or cycle starting *from a given node* requires $O(n+m)$ time using BFS. Step_4 is executed $O(n)$ times. Therefore, the time complexity of Algorithm 10 is $O(n^2 m)$. □

Selecting one link from each path (or cycle) defines a set of simultaneous link failures that the red/blue trees can protect. However, selecting one node from each path (or cycle) will not define a set of simultaneous node failures that the trees can protect unless these nodes have degree 2 in the blue tree. This presents difficulties if we wish to use our G-MFBG algorithm for designing red/blue trees with a high value for quality of node failure protection.

Suppose we design the red/blue trees using DFS [32] and call two vertices *unrelated* if neither is a descendant of the other in the DFS tree. With this definition we can see that the red/blue trees can protect simultaneous failures of mutually unrelated vertices. This means that to design red/blue trees with a high value for the quality of vertex failure protection, we need an algorithm for designing the blue tree with the largest number of mutually unrelated vertices. Research along the above lines is in progress.

VIII. Generalizations of the MFBG Scheme

In this section, we show that G-MFBG-V (G-MFBG-E, respectively) is indeed a generalization of MFBG-V (MFBG-E, respectively). We will prove that for any given two-connected graph (two-edge connected graph, respectively), every pair of red/blue trees that can be constructed by MFBG-V (MFBG-E, respectively) can also be constructed by G-MFBG-V (MFBG-E, respectively). We then present an example showing the existence of a pair of red/blue trees which can be constructed by G-MFBG, but not by MFBG. We will present another example showing the existence of a pair of redundant trees which cannot be constructed by G-MFBG. Recall that p_v^B (p_v^R, respectively) denotes the unique $s-v$ path in T^B (T^R, respectively) and that $e^B(v)$ ($e^R(v)$, respectively) denotes the undirected edge of G which is adjacent with u and is on p_v^B (p_v^R, respectively).

Theorem 7: Let G be a two-connected graph and s a distinguished root node. Then, Algorithm 3 correctly constructs a pair of single-node recovery trees T^B and T^R.

Proof: We will use T_1^B and T_1^R to denote the blue tree and the red tree obtained after Step_2 of the algorithm and use T_{j+1}^B and T_{j+1}^R to denote the blue tree and red tree obtained after adding the blue chain and the red chain corresponding to a path found in Step_4 of the algorithm to T_j^B and T_j^R, respectively.

First, we will prove that T_j^B and T_j^R impose a partial order on the $n+1$ nodes in $V' = V \cup \{s^R\}$ via PV1, PV2, and PV3 (see Section III for definition) for every j. In other words, for any two distinct nodes $u, v \in V', u \prec v$ and $v \prec u$ cannot be both true.

At the end of Step_2, all precedence relations can be characterized by $s^R \prec v_k \prec v_{k-1} \prec \cdots \prec v_1 \prec s$. Therefore, at the end of Step_2, T_1^B and T_1^R impose a partial order on V'.

Assume that T_j^B and T_j^R impose a partial order on V'. Let $[x, v_1, \ldots, v_k, y]$ be the path found in Step_4 of the algorithm. It follows from the algorithm that $x \neq y$.

Let u and v be two nodes in T_j^B such that $u \not\prec v$ before this execution of Step_4 but $u \prec v$ after this execution of Step_4. This would imply $u \prec y$ and $x \prec v$ before this execution of Step_4. This in turn would imply that $v \not\prec u$ before this execution of Step_4, for otherwise we would have $x \prec v \prec u \prec y$, which contradicts the assumption that $x \neq y$. Therefore, after this execution of Step_4, $u \prec v$ and $v \prec u$ cannot be both true.

Let u be a node in T_j^B and v a node in T_{j+1}^B but not T_j^B. $u \prec v$ after this execution of Step_4 implies that $u = y$ or $u \prec y$ before this execution of Step_4. $v \prec u$ after this execution of Step_4 implies that $u = x$ or $x \prec u$ before this execution of Step_4. Therefore, $u \prec v$ and $v \prec u$ after this execution of Step_4 implies that $x \prec y$ before this execution of Step_4, which contradicts our assumption that $x \neq y$.

Let u and v be two nodes in T_{j+1}^B but not T_j^B. Without loss of generality, we may assume that $u = v_i$ and $v = v_{i'}$ with $1 \leq i < i' \leq k$. It follows from our definition of the precedence relations that $u \not\prec v$ and $v \not\prec u$ before this execution of Step_4 and that $u \not\prec v$ and $v \prec u$ after this execution of Step_4. Therefore, T_{j+1}^B and T_{j+1}^R also impose a partial order on V'.

Second, we will prove that the algorithm can find a cycle or path as long as T^B does not span all the nodes in V. The existence of the cycle found in Step_2 follows from the fact that the graph is two connected [15]. For the same reason, in Step_4, we can find a path $[u, v_1, \ldots, v_k, v]$ connecting two distinct nodes u and v already on the trees and $k \geq 1$ nodes not on the trees. If $u \not\prec v$, Step_4 is successful with the path

$[x = u, v_1, \ldots, v_k, y = v]$. If $u \prec v$, then we must have $v \not\prec u$. In this case, Step_4 is successful with the path $[x = v, v_k, \ldots, v_1, y = u]$. Therefore, Algorithm 3 can successfully construct a pair of red/blue trees T^R and T^B spanning all the nodes in V.

Finally, we will prove that for any node $u \in V$ other than the root node s, p_u^B and p_u^R are node-disjoint. Let x be any internal node on p_u^B. We must have $u \prec x \prec s$. Let y be any internal node on p_u^R. We must have $s^R \prec y \prec u$. Therefore, we have $y \prec x$, which implies that $x \neq y$. This completes the proof of the theorem. □

Theorem 8: Let G be a two-connected graph and s a distinguished root node. Then, every pair of red/blue trees constructed by Algorithm 1 can be constructed by Algorithm 3.

Proof: Note that at the end of Step_2 and each execution of Step_4 of Algorithm 1, the set of voltages $\{v^B(u) | u \in T_j^B\} \cup \{v^R(s)\}$ are distinct and imply a partial order on the $n+1$ nodes in V' via PV1, PV2, and PV3. [If (u, v) is a directed edge from u to v in T^B, then $v^B(v) < v^B(u)$; If (u, v) is a directed edge from u to v in T^R with $u \neq s$, then $v^R(u) < v^R(v)$; If (s, v) is a directed edge from s to v in T^R, then $v^R(s) < v^B(v)$]. Therefore, for each feasible execution of Algorithm 1, there is a corresponding feasible execution of Algorithm 3 which updates T^B and T^R in exactly the same way. Therefore, every pair of red/blue trees constructed by Algorithm 1 can also be constructed by Algorithm 3. □

Theorem 9: Let G be a two-edge connected graph and s a distinguished root node. Then, Algorithm 4 correctly constructs a pair of single-link recovery trees T^B and T^R.

Proof: We will use T_1^B and T_1^R to denote the blue tree and the red tree obtained after Step_2 of the algorithm and use T_{j+1}^B and T_{j+1}^R to denote the blue tree and red tree obtained after adding the blue chain and the red chain corresponding to a path found in Step_4 of the algorithm to T_j^B and T_j^R, respectively.

First, we will prove that T_j^B and T_j^R impose a partial order on the m edges of G via PE1, PE2, and PE3 for every j. In other words, for any two distinct edges $e_1, e_2 \in E, e_1 \prec e_2$, and $e_2 \prec e_1$ cannot be both true.

At the end of Step_2, all precedence relations can be characterized by $[s, v_k] \prec [v_k, v_{k-1}] \prec \cdots \prec [v_1, s]$. Therefore, at the end of Step_2, T_1^B and T_1^R impose a partial order on E.

Assume that T_j^B and T_j^R impose a partial order on E. Let $[x, v_1, \ldots, v_k, y]$ be the path found in Step_4 of the algorithm. It follows from the algorithm that $e^B(x) \not\prec e^R(y)$.

Let e_1 and e_2 be two edges of G that are used by T_j^B or T_j^R such that $e_1 \prec e_2$ and $e_2 \not\prec e_1$ before this execution of Step_4 but $e_1 \prec e_2$ and $e_2 \prec e_1$ after this execution of Step_4. Due to the way the precedence relations are added during Step_4, $e_2 \prec e_1$ becomes true due to this execution of Step_4 if and only if $e_2 \prec e^R(y)$ and $e^B(x) \prec e_1$ before this execution of Step_4. However, that implies $e^B(x) \prec e^R(y)$ before this execution of Step_4, contradicting to our assumption that $e^B(x) \not\prec e^R(y)$. Therefore, $e_1 \prec e_2$ and $e_2 \prec e_1$ cannot be both true after this execution of Step_4.

Let e_{old} be an edge of G that is used by T_j^B or T_j^R and e_{new} be an edge of G that is used by T_{j+1}^B or T_{j+1}^R but not T_j^B or T_j^R. Before this execution of Step_4, none of $e_{\text{old}} \prec e_{\text{new}}$ or $e_{\text{new}} \prec e_{\text{old}}$ is true. If $e_{\text{old}} \prec e_{\text{new}}$ is true after this execution of Step_4, we would have either $e_{\text{old}} = e^R(y)$ or $e_{\text{old}} \prec e^R(y)$. If $e_{\text{new}} \prec e_{\text{old}}$ is true after this execution of Step_4, we would have either $e_{\text{old}} = e^B(x)$ or $e^B(x) \prec e_{\text{old}}$. Therefore, if $e_{\text{old}} \prec e_{\text{new}}$ and $e_{\text{new}} \prec e_{\text{old}}$ are both true after this execution of Step_4, we would have $e^B(x) \prec e^R(y)$, another contradiction. Therefore, $e_{\text{new}} \prec e_{\text{old}}$ and $e_{\text{old}} \prec e_{\text{new}}$ cannot be both true after this execution of Step_4.

Let e_1 and e_2 be two edges of G that are used by T_{j+1}^B or T_{j+1}^R but not T_j^B or T_j^R. Without loss of generality, assume that e_1 is closer to x than e_2. It follows from the algorithm that $e_2 \prec e_1$ but $e_1 \not\prec e_2$. Therefore, $e_1 \prec e_2$ and $e_2 \prec e_1$ cannot be both true after this execution of Step_4. Therefore, T_{j+1}^B and T_{j+1}^R also impose a partial order on E.

Second, we will prove that the algorithm can find a cycle or path as long as T^B does not span all the nodes in V. The existence of the cycle found in Step_2 follows from the fact that the graph is two connected [15]. For the same reason, in Step_4, we can find a path $[u, v_1, \ldots, v_k, v]$ connecting two (not necessarily distinct) nodes u and v already on the trees and $k \geq 1$ nodes not on the trees. If $u = s$ or $v = s$, Step_4 is trivially successful. Now assume that both u and v are different from s. Since T_j^B and T_j^R impose a partial order on $E, e^B(u) \prec e^R(v)$ and $e^B(v) \prec e^R(u)$ cannot be both true, for otherwise we would have $e^B(u) \prec e^R(v) \prec e^B(v) \prec e^R(u) \prec e^B(u)$, contradicting the assumption that T_j^B and T_j^R impose a partial order on E. Therefore, Step_4 is always successful.

Finally, we will prove that for any node $u \in V$ other than the root node s, p_u^B, and p_u^R are link disjoint. Let e_1 be any edge on p_u^R. We must have $e_1 \prec e^B(u)$. Let e_2 be any edge on p_u^B. We must have $e^R(u) \prec e_2$. Therefore, we have $e_1 \prec e_2$, which implies that $e_1 \neq e_2$. This completes the proof of the theorem. □

Theorem 10: Let G be a two-edge connected graph and s a distinguished root node. Then, every pair of red/blue trees constructed by Algorithm 2 can be constructed by Algorithm 4.

Proof: In Step_2 of both Algorithm 2 and Algorithm 4, we are seeking a cycle. Clearly, Algorithm 4 can choose the cycle that Algorithm 2 is using. At the end of Step_2, T^B, and T^R imposes a partial order on E.

Step_4 of Algorithm 2 selects a path $[x, v_1, \ldots, v_k, y]$ such that x and y are already on the trees, $v_1, \ldots, v_k$ are not on the trees yet, and that $v^B(x) > v^B(y)$ unless $y = s$. In both cases, it was proved in [15] that p_x^B and p_y^R are link-disjoint. This implies that $e^R(y) \prec e^B(x)$, which, in turn, implies that $e^B(x) \not\prec e^R(y)$. This means that Step_4 of Algorithm 4 can use the same path. Therefore, we have proved that every pair of red/blue trees constructed by Algorithm 2 can be constructed by Algorithm 4. □

Fig. 4 shows a pair of single-failure (link or node) recovery trees R (the red tree) and B (the blue tree). We will show that both Algorithm 3 and Algorithm 4 can construct this pair of trees while Algorithm 1 and 2 cannot construct this pair of trees.

First of all, we show that both G-MFBG-V and G-MFBG-E can construct this pair of red/blue trees. Starting from root node a, we can find the cycle $[a, b, c, d, a]$ and assign the corresponding red tree edges and blue tree edges labeled $\{1\}$. For G-MFBG-V, the partial order is characterized by $a^R \prec d \prec c \prec b \prec a$. For G-MFBG-E, the partial order is characterized by $[a, d] \prec [d, c] \prec [c, b] \prec [b, a]$.

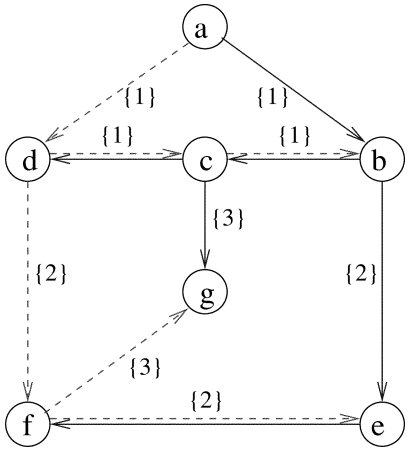

Fig. 4. Pair of red/blue trees.

Next, we find the path $[b, e, f, d]$. For G-MFBG-V, we note that $b \not\prec d$. For G-MFBG-E, we note that $[b, a] \not\prec [a, d]$. Therefore, the path is feasible for both G-MFBG-V and G-MFBG-E. Hence, we can assign the tree edges labeled $\{2\}$ as shown in Fig. 4. For G-MFBG-V, the augmented partial order is characterized by $a^R \prec d \prec c \prec b \prec a$ and $d \prec f \prec e \prec b$. For G-MFBG-E, the augmented partial order is characterized by $[a, d] \prec [d, c] \prec [c, b] \prec [b, a]$ and $[a, d] \prec [d, f] \prec [f, e] \prec [e, b] \prec [b, a]$.

Finally, we find a path $[c, g, f]$. For G-MFBG-V, c and f are unrelated in the partial order. Therefore, the path is feasible. For G-MFBG-E, $[c, b]$ and $[f, d]$ are unrelated in the partial order. Therefore, the path is feasible. Hence, we can assign the tree edges labeled $\{3\}$ as shown in Fig. 4. This completes the tree construction.

Next, we show that this pair of trees cannot be constructed by either Algorithm 1 or Algorithm 2.

If it were constructed using the MFBG scheme, the first cycle must be $[a, b, c, d, a]$ (this is the only cycle that is formed by a path from a in the blue tree followed by an edge that is only used by the red tree). Since there is only one blue edge (c, g) into node g and only one red edge (f, g) into node g, $[c, g, f]$ must be a path found in Step_4 of the algorithms. For a similar reason, $[b, e, f, d]$ must be a path found in Step_4 of the algorithms. Since the end node f on $[c, g, f]$ is an internal node on $[b, e, f, d]$, the algorithms finds $[b, e, f, d]$ before $[c, g, f]$.

When the cycle $[a, b, c, d, a]$ is found, both MFBG-V and MFBG-E add the edges $(a, b), (b, c), (c, d)$ to the blue tree and the edges $(a, d), (d, c), (c, b)$ to the red tree. The MFBG-V scheme (MFBG-E scheme, respectively) assigns the voltages to nodes a, b, c, d such that $v^B(a) > v^B(b) > v^B(c) > v^B(d) > v^R(a) = 0$ [$v^B(a) > v^B(b) > v^R(b) > v^B(c) > v^R(c) > v^B(d) > v^R(d) > v^R(a) = 0$, respectively].

When the path $[b, e, f, d]$ is found (note that $v^B(b) > v^B(d)$), both MFBG-V and MFBG-E add the edges $(b, e), (e, f)$ to the blue tree and the edges $(d, f), (f, e)$ to the red tree. For MFBG-V, $v^M(b) = v^B(c)$. For MFBG-E, $v^M(b) = v^R(b)$. Therefore, the MFBG-V scheme (MFBG-E scheme, respectively) assigns the voltages to nodes e, f such that $v^B(b) > v^B(e) > v^B(f) > v^B(c)$ [$v^B(b) > v^B(e) > v^R(e) > v^B(f) > v^R(f) > v^R(b)$, respectively].

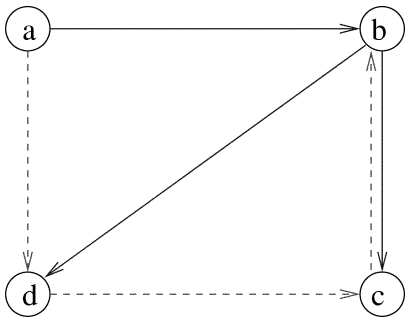

Fig. 5. Difficult case.

When the path $[c, g, f]$ is found (note that $v^B(f) > v^B(c)$), both MFBG-V and MFBG-E add the arc (f, g) to the blue tree and the arc (c, g) to the red tree and assign voltage(s) at g accordingly.

However, the resulting pair of red/blue trees so constructed is different from the pair of red/blue trees shown in Fig. 4. At the last step of the tree construction, c and f are unrelated (for G-MFBG-V), $[c, b]$ and $[d, f]$ are unrelated (for G-MFBG-E). Therefore, the generalized MFBG schemes can add (c, g) to the blue tree and (f, g) to the red tree. However, we have $v^B(c) < v^B(f)$. Therefore, the original MFBG schemes can only add (f, g) to the blue tree and (c, g) to the red tree. This example, together with Theorems 3 and 4, shows that G-MFBG-V (G-MFBG-E, respectively) is indeed more general than MFBG-V (MFBG-E, respectively).

We want to point out that there are red/blues that cannot be generated by our generalized schemes. Such an example is illustrated in Fig. 5. Note that there are only two cycles $[a, b, c, d, a]$ and $[a, b, d, a]$ passing through root node a. If we pick the first cycle, there should be an additional blue edge (c, d) in the blue tree [the edge (b, d) should be removed from the blue tree in this case]. If we pick the second cycle, there should be an additional red edge (d, b) in the red tree [the edge (c, b) should be removed from the red tree in this case]. This shows that our generalized schemes are not the most general.

IX. Computational Results

In order to evaluate how useful our proposed algorithms and heuristics are, we implemented all of them and tested them out on randomly generated input data. We use RedBlueV and RedBlueE to denote the algorithms in [15] for node recovery and link recovery, respectively, where the paths and cycles are computed using DFS, without any of the QoS or QoP considerations.

A C++ class library, LEDA is used in all of our implementations. The experiments were carried out on a Pentium III desktop with 512 M memory and running RedHat Linux 7.0. We have used 50, 100, and 200 as the number of nodes in the networks. For each value of n, we have used $3n$ and $n \log n$ as the number of links. Therefore, there are six node-link combinations. For each given size (given by the node-link

TABLE I
MinDelayV WITH DELAY IN $[1, 10]$

network	RedBlueV		MinDelayV		decrease
$n \times m$	avg	sec	avg	sec	
050x0150	33.23	0.00	9.89	0.04	70%
050x0282	25.40	0.00	5.85	0.07	77%
100x0300	56.92	0.01	11.93	0.24	79%
100x0664	39.81	0.01	5.94	0.52	85%
200x0600	93.15	0.08	13.43	1.63	86%
200x1529	66.18	0.03	6.14	4.01	91%

TABLE II
MinDelayE WITH DELAY IN $[1, 10]$

network	RedBlueE		MinDelayE		decrease
$n \times m$	avg	sec	avg	sec	
050x0150	32.20	0.00	9.76	0.07	70%
050x0282	19.82	0.00	5.61	0.21	72%
100x0300	52.83	0.01	11.68	0.48	78%
100x0664	29.25	0.01	5.90	1.96	80%
200x0600	89.83	0.09	13.62	3.58	85%
200x1529	46.58	0.04	6.18	18.49	87%

TABLE III
MinDelayV USING HOP COUNT AS DELAY

network	RedBlueV		MinDelayV		decrease
$n \times m$	avg	sec	avg	sec	
050x0150	5.98	0.00	2.55	0.04	57%
050x0282	4.60	0.00	1.87	0.07	59%
100x0300	10.44	0.01	3.05	0.25	71%
100x0664	7.25	0.01	2.07	0.51	71%
200x0600	17.17	0.08	3.53	1.80	79%
200x1529	12.12	0.03	2.28	3.99	81%

TABLE IV
MinDelayE USING HOP COUNT AS DELAY

network	RedBlueE		MinDelayE		decrease
$n \times m$	avg	sec	avg	sec	
050x0150	5.85	0.00	2.58	0.08	56%
050x0282	3.60	0.00	1.87	0.21	48%
100x0300	9.66	0.01	3.04	0.53	69%
100x0664	5.35	0.01	2.11	2.02	61%
200x0600	16.40	0.08	3.53	3.96	78%
200x1529	8.53	0.04	2.34	19.77	73%

TABLE V
MinCostV WITH COST IN $[1, 10]$

network	RedBlueV		MinCostV		decrease
$n \times m$	avg	sec	avg	sec	
050x0150	400.99	0.00	254.64	0.02	37%
050x0282	409.96	0.00	205.35	0.03	50%
100x0300	793.39	0.01	499.77	0.12	37%
100x0664	822.35	0.01	383.60	0.22	53%
200x0600	1592.74	0.09	1003.18	0.90	37%
200x1529	1633.09	0.04	718.59	1.93	56%

TABLE VI
MinCostE WITH COST IN $[1, 10]$

network	RedBlueE		MinCostE		decrease
$n \times m$	avg	sec	avg	sec	
050x0150	403.05	0.00	253.83	0.02	37%
050x0282	409.12	0.00	202.70	0.03	50%
100x0300	788.22	0.01	497.81	0.12	37%
100x0664	821.10	0.01	386.00	0.22	53%
200x0600	1595.09	0.08	1007.62	0.90	37%
200x1529	1637.52	0.03	711.17	1.94	57%

TABLE VII
MaxBandV WITH BANDWIDTH IN $[1, 10]$

network	RedBlueV		MaxBandV		increase
$n \times m$	avg	sec	avg	sec	
050x0150	1.00	0.00	2.83	0.00	183%
050x0282	1.00	0.00	5.51	0.00	451%
100x0300	1.00	0.01	2.16	0.01	116%
100x0664	1.00	0.01	5.75	0.01	475%
200x0600	1.00	0.08	1.67	0.07	067%
200x1529	1.00	0.03	5.74	0.06	474%

combination), we randomly generate 100 two-connected graphs (for node recovery) or two-edge connected graphs (for link recovery). The bandwidth, cost, and delay of the links are random integers uniformly distributed in the range $[1, 10]$. *Each entry in the tables reported here is the average over* 100 *runs.*

Tables I and II present the results of delay reduction for node recovery trees and link recovery trees, respectively, with the link delay uniformly distributed in the range $[1, 10]$. From the tables, we observe that QoS heuristics require longer running time, but construct trees with lower average delays. The delay reductions range from 70% to 91%. We observe that QoS heuristics are more effective for $m = n \log n$ than for $m = 3n$. This is expected because when there are more edges in the graph, there are more choices at each step of the QoS heuristics.

In many cases, hop count is used as a measurement for end-to-end delays. Therefore, we also tested the delay reduction heuristics when the delay of every link is one. These results are reported in Tables III and IV. We observe that the delay reductions here are slightly lower, but are consistent, ranging from 48% to 81%.

Results for the cost reduction heuristics are reported in Tables V and VI for node recovery and link recovery, respectively. We observe that the running time of these heuristics are pretty short. Delay reduction are consistent, ranging from 37% to 57%. Again, we notice that the results for $m = n \log n$ are better than that for $m = 3n$.

In Tables VII and VIII, we present computational results for MaxBand for node recovery and link recovery, respectively. We observe from the tables that MaxBandV and MaxBandE construct red/blue trees with significantly larger bandwidths while using very short time. Again, the performance for $m = n \log n$ is better than that for $m = 3n$.

TABLE VIII
MaxBandE WITH BANDWIDTH IN $[1, 10]$

network	RedBlueE		MaxBandE		increase
$n \times m$	avg	sec	avg	sec	
050x0150	1.00	0.00	2.72	0.00	172%
050x0282	1.00	0.00	5.53	0.00	453%
100x0300	1.00	0.01	2.02	0.01	102%
100x0664	1.00	0.01	5.74	0.01	474%
200x0600	1.00	0.08	1.70	0.06	070%
200x1529	1.00	0.03	5.70	0.06	470%

TABLE IX
EnhancingQoPE

network	RedBlueE		MaxQoPE		increase
$n \times m$	QoP	sec	QoP	sec	
050x0150	23.49	0.00	47.38	0.00	102%
050x0282	25.75	0.00	48.00	0.00	86%
100x0300	45.51	0.01	96.44	0.01	112%
100x0664	50.92	0.01	97.99	0.01	92%
200x0600	91.46	0.08	194.18	0.06	112%
200x1529	99.73	0.04	197.90	0.06	98%

Finally, Table IX reports the results of our heuristic for enhancing QoP in link recovery schemes. The running time of the QoP heuristic is similar to that without the QoP consideration. The increase in QoP is between 86% and 112%, which is significant. Here, we observe that the effective of the QoP heuristic performs better for $m = 3n$ than for $m = n \log n$. A possible explanation to this scenario is that it is easier for the DFS used in our implementation of the MFBG schemes to find small cycles and paths when there are more the edges in the graph.

X. CONCLUSION AND FUTURE RESEARCH

The focus of this paper is on the investigation of several issues relating to QoS and QoP in redundant tree based preplanned recovery schemes for any single-link failure in two-edge connected graphs and any single-node failure in two-connected graphs. The work by Médard *et al.* [15] on the MFBG schemes and the work by Lumetta *et al.* [12] on robustness of recovery schemes provided the inspiration for the work reported in this paper. We have added and extended the contributions in [15] in different ways. We have investigated the construction of red/blue trees using QoS parameters such as delay, cost, and bandwidth. We have introduced the concept of QoP and showed how to incorporate this parameter in the construction of red/blue trees. Finally, we have generalized the scheme in [15]. In the following, we summarize these contributions and present some directions of future research.

The MFBG algorithms to construct the red/blue trees are very elegant and generalize and simplify an earlier work [7]. Whereas the scheme in [7] is based on st-numbering [24], the MFBG scheme employs a method closely related to what is called the *ear decomposition* of two-connected and graphs and two-edge connected graphs [28]. Basically, in ear decomposition cycles and paths are added one at a time. These cycles and paths are called ears of the decomposition. In this paper, we have further generalized the MFBG scheme. The generalized algorithm G-MFBG uses a new node ordering rule, in contrast to the voltage based rule used by MFBG. We have also pointed out that there are red/blues that cannot be generated by this generalized scheme. In fact, we have given an example of a pair of red/blue trees (constructed by adding two ears at time) that cannot be generated by G-MFBG. However, it can be shown that the new rule is the most general one if one were to use an algorithm which adds cycles and paths one at a time.

The red/blue trees are also called independent trees. To recover from two simultaneous failures three independent trees are required. In other words, the graph under consideration must be three connected. It is possible to construct three independent trees for three-connected graphs. But the algorithm in [3] is quite complex. Developing a simpler scheme similar to ours to construct three independent trees for three-connected graphs is an interesting direction of research. Another related work in this context is in [9].

The concept of QoP introduced in this paper captures the ability of single-link recovery schemes to provide protection against simultaneous multiple failures. The more the number of ears used in constructing the red/blue trees the better the QoP. Another interesting direction of research is to develop similar ideas to enhance the QoP of trees developed for recovery from any pair of link failures.

In [16], the authors propose a generalized loop-back recovery schemes for optical mesh networks whose topologies are two-edge connected or two connected. This work is in the context of recovery using link restoration. The generalized loop-back recovery method constructs two directed spanning graphs such that one of the spanning subgraphs, R, contains the reverse of the directed edges of the other spanning subgraph, B. These spanning subgraphs are constructed using cycles and paths (ears) one at a time. This approach is, thus, closely related to that employed by MFBG and G-MFBG algorithms. As in the case of redundant trees we can define the QoP of the generalized loop-back recovery graphs. The ideas developed in this paper can, therefore, be useful in designing methods for constructing generalized loop-back recovery graphs with high QoP.

The use of red/blue trees permits path rerouting. If a node is disconnected from the primary route because of a failure, then a backup route can be identified using these trees. This recovery can be achieved in a distributed manner using a broadcast scheme as in [16]

The concept of QoP introduced in this paper is closely related to the DFS based construction of red/blue trees of [32] and the concept of robustness introduced in [12]. In [32], Xue and Gottapu present two linear time algorithms for constructing red/blue trees in two-edge connected graphs or two-connected graphs, based on pre-order traversal of the DFS tree and post-order traversal of the DFS tree. The pre-order traversal algorithm constructs a pair of red/blue trees with more ears, therefore can

provide more localized protection. The post-order traversal algorithm constructs a pair of red/blue trees with fewer ears, therefore, has a smaller total cost. In [12], Lumetta *et al.* define robustness of a single-failure recovery scheme as the percentage of the simultaneous multiple failures that this recovery scheme can protect. The smaller size the ears (lengths of cycles or paths), the greater the number of ears and, hence, the better the QoP. It will be interesting to develop a heuristic (similar to Algorithm 10) which incorporates both QoP and robustness as parameters in the construction of red/blue trees.

In the development of our heuristics for red/blue tree construction, we have assumed that the three metrics, namely, average delay, total cost and bandwidth are independent ones. In a practical scenario, this may not be the case. A single heuristic which takes care of these parameters in an integrated manner is desirable. In such an integrated heuristic, we can incorporate appropriate tuning parameters whose weights will depend upon the needs of the application. This is a typical approach often followed in developing heuristics for problems involving multiple parameters (for instance, the QoS routing problem). Work along these lines is now in progress. The three heuristics we have developed provide the basis for developing an integrated heuristic for the red/blue tree construction problem. Another direction of future research is to design provably good approximation schemes to the QoS and QoP problems related to red/blue trees. Results of a similar nature reported in the context of QoS routing algorithms (for instance, see [8] and [11]) may be profitably used in this work.

Our simulations have been performed on graphs selected from the LEDA library. More realistic graphs specifically designed to capture characteristics of telecommunication networks have been reported in [21], [27], and [34]. Our future work will study the performance of our heuristics by evaluating them on such more realistic networks.

ACKNOWLEDGMENT

The authors would like to thank the reviewers for their careful comments and suggestions which have improved the quality of our presentation.

REFERENCES

[1] V. Anand and C. Qiao, "Dynamic establishment of protection paths in WDM networks," in *IEEE ICCCN'2000*, 2000, pp. 198–204.

[2] M. Blum, R. Floyd, V. Pratt, R. Rivest, and R. Tarjan, "Time bounds for selection," *J. Comput. Syst. Sci.*, vol. 7, pp. 448–461, 1972.

[3] J. Cheriyan and S. N. Maheshwari, "Finding nonseperating induced cycles and independent spanning trees in 3-connected graphs," *J. Algorithms*, vol. 9, pp. 507–537, 1988.

[4] I. Chlamtac, A. Ganz, and G. Karmi, "Lightpath communications: An approach to high bandwidth optical WAN's," *IEEE Trans. Commun.*, vol. 40, pp. 1171–1182, July 1992.

[5] P. E. Green Jr., *Fiber Optic Networks*. Englewood Cliffs, NJ: Prentice-Hall, 1993.

[6] D. K. Hsing, B. C. Cheng, G. Goncu, and L. Kant, "A restoration methodology based on pre-planned source routing in ATM networks," in *Proc. IEEE Int. Conf. Communications (ICC)*, June 1997, pp. 277–182.

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

[8] A. Juttner, B. Szviatovski, I. Mecs, and Z. Rajko, "Lagrange relaxation based method for the QoS routing problem," in *Proc. IEEE INFOCOM'01*, Apr. , pp. 859–868.

[9] S. Khuller and B. Schieber, "On independent trees," *Inform. Process. Lett.*, vol. 42, pp. 321–323, 1992.

[10] K. C. Lee and V. O. K. Li, "A wavelength rerouting algorithm in wide-area all-optical networks," *J. Lightwave Technol.*, vol. 14, pp. 1218–1229, June 1996.

[11] D. H. Lorenz and D. Raz, "A simple efficient approximation scheme for the restricted shortest path problem," *Oper. Res. Lett.*, vol. 28, pp. 213–219, 2001.

[12] S. S. Lumetta, M. Médard, and Y. Tseng, "Capacity versus robustness: A tradeoff for link restoration in mesh networks," *J. Lightwave Technol.*, vol. 18, pp. 1765–1775, Dec. 2000.

[13] M. Médard, S. G. Finn, and R. A. Barry, "A novel approach to automatic protection switching using trees," in *Proc. IEEE Int. Conf. Communications (ICC)*, 1997, pp. 272–276.

[14] ——, "WDM loop-back recovery in mesh networks," in *Proc. OFC'98*, 1998, pp. 298–299.

[15] M. Médard, S. G. Finn, R. A. Barry, and R. G. Gallager, "Redundant trees for preplanned recovery in arbitrary vertex-redundant or edge-redundant graphs," *IEEE/ACM Trans. Networking*, vol. 7, pp. 641–652, Oct. 1999.

[16] M. Médard, R. A. Barry, S. G. Finn, W. He, and S. S. Lumetta, "Generalized loop-back recovery in optical mesh networks," *IEEE Trans. Networking*, vol. 10, pp. 153–164, Feb. 2002.

[17] D. Medhi and D. Tipper, "Multi-layered network survivability—Models, analysis, architecture, framework and implementation and overview," in *Proc. DARPA Information Survability Conf. Exposition*, Hilton Head, SC, Jan. 25–27, 2000, pp. 172–186.

[18] B. Mukherjee, *Optical Communication Networks*. New York: McGraw-Hill, 1997.

[19] S. Ramamurthy and B. Mukherjee, "Survivable WDM mesh networks. Part I—Protection," in *Proc. IEEE INFOCOM*, Mar. 1999, pp. 744–751.

[20] ——, "Survivable WDM mesh networks. II. Restoration," in *Proc. IEEE Int. Conf. Communications (ICC)*, June 1999, pp. 2023–2030.

[21] G. Siganos, M. Faloutsos, P. Faloutsos, and C. Faloutsos, "Power-laws and the AS-level internet topology," *IEEE/ACM Trans. Networking*, to be published.

[22] G. D. Signorelli, M. Gryseels, and P. M. Demeester, "SDH over WDM: Interworking and planning aspects," in *Proc. SPIE*, vol. 3408, 1998, pp. 235–246.

[23] R. E. Tarjan, "Depth first search and linear graph algorithms," *SIAM J. Comput.*, vol. 1, pp. 146–160, 1972.

[24] K. Thulasiraman and M. N. S. Swamy, *Graphs: Theory and Algorithms*. New York: Wiley, 1992.

[25] H. Wang, E. Modiano, and M. Médard, "Partial path protection for WDM networks: End-to-end recovery using local failure information," in *Proc. IEEE ISCC'2002*, July, pp. 719–725.

[26] N. Wauters, C. Ocakoglu, K. Struyve, and F. P. Falcao, "Survivability in a new pan-European carriers' carrier network based on WDM and SDH technology: Current implementation and future requirements," *IEEE Commun. Mag.*, vol. 37, no. 8, pp. 63–69, 1999.

[27] B. M. Waxman, "Routing of multipoint connections," *IEEE J. Select. Areas Commun.*, vol. 6, pp. 1617–1622, Dec. 1988.

[28] D. B. West, *Introduction to Graph Theory*. Englewood Cliffs, NJ: Prentice-Hall, 1996, 2001.

[29] T. H. Wu and W. I. Way, "A novel passive protected SONET bidirectional self-healing ring architecture," *J. Lightwave Technol.*, vol. 10, pp. 1314–1322, Sept. 1992.

[30] G. Xue, L. Chen, and K. Thulasiraman, "QoS issues in redundant trees for protection in vertex-redundant or edge-redundant networks," in *Proc. IEEE Int. Conf. Communications (ICC)*, 2002, pp. 2766–2770.

[31] ——, "Delay reduction in redundant trees for pre-planned protection against singlelink/node failure in 2-connected graphs," in *Proc. IEEE GLOBECOM*, 2002, pp. 2691–2695.

[32] G. Xue and R. Gottapu, "Survivable network design using path-based spanners," *Comput. Commun.*, 2002, submitted for publication.

[33] B. Zhou and H. T. Mouftah, "Survivable alternate routing for WDM networks," in *Proc. Optical Fiber Communication Conf. (OFC)*, Mar. 2002, pp. 543–544.

[34] E. W. Zegura, K. Calvert, and M. J. Donahoo, "A quantitative comparison of graph-based models for internet topology," *IEEE/ACM Trans. Networking*, vol. 5, pp. 770–783, Dec. 1997.

[35] T. F. Znati, T. Alrabiah, and R. Melhem, "Low-cost, delay-bounded point-to-multipoint communication to support multicasting over WDM networks," *Comput. Networks*, vol. 38, pp. 423–445, 2002.

Guoliang Xue (M'98–SM'99) received the B.S. degree in mathematics and the M.S. degree in operations research from Qufu Teachers University, Qufu, China, and the Ph.D. degree in computer science from the University of Minnesota, Minneapolis, in 1991.

He worked for two years at the Army High Performance Computing Research Center, Minneapolis, MN, as a Postdoctoral Fellow before joining the University of Vermont, Burlington, as an Assistant Professor of computer science, where he was promoted to Associate Professor in 1999. Since 2001, he has been an Associate Professor of computer science and engineering at Arizona State University, Tempe. His research interests include design and analysis of algorithms, optimization, and computer networks. He has published over 100 technical papers in these areas, including papers in the *SIAM Journal on Computing, SIAM Journal on Discrete Mathematics, and SIAM Journal on Optimization*.

Dr. Xue received the second prize from the Chinese National Education Committee for his publications in Operations Research. As a graduate student, he received the Doctoral Dissertation Fellowship from the Graduate School of the University of Minnesota. In 1994, he received the Research Initiation Award from the National Science Foundation (NSF). He is the coeditor of eight journal special issues and conference proceedings. He is the Technical Program Co-Chair of the IEEE IPCCC 2003 and the IEEE HPSR 2004 and is serving as a Technical Program Committee Member of many conferences. He has served as a panelist to NSF and a reviewer to National Science and Engineering Research Council of Canada (NSERC). He is profiled in the Marquis *Who's Who in America*. He has published papers in the IEEE TRANSACTIONS ON COMPUTERS and the IEEE TRANSACTIONS ON COMMUNICATIONS.

Li Chen received the B.S. degree in computer science from Najing University, Najing, China and the M.S. degree in computer science from the University of Vermont, Burlington, in 2002. She is currently working toward the Ph.D. degree in computer science from Texas A&M University, College Station.

From 2002 to 2003, she was an Instructor in the Department of Computer Science, University of Vermont.

Krishnaiyan Thulasiraman (M'72–S'84–F'90) received the Ph.D. degree in electrical engineering from the Indian Institute of Technology (IIT), Madras, India, in 1968.

He holds the Hitachi Chair and is Professor with the School of Computer Science, University of Oklahoma, Norman, where he has been since 1994. Prior to joining the University of Oklahoma, he was a Professor (1981–1994) and Chair (1993–1994) of the Electrical and Computer Engineering Department, Concordia University, Montreal, Canada. He was on the Faculty in the Electrical Engineering and Computer Science Departments, IIT, from 1965 to 1981. He has published more than 100 papers in archival journals. He coauthored *Graphs, Networks, and Algorithms* (New York: Wiley, 1981) and *Graphs: Theory and Algorithms* (New York: Wiley, 1992). He has received more than $1 000 000 in direct costs of research grants from Nortel Networks and national and state research agencies in Canada. He has held visiting positions at the Tokyo Institute of Technology, University of Karlsruhe, University of Illinois at Urbana–Champaign, and Chuo University, Tokyo. His research interests have been in graph theory, combinatorial optimization, and algorithms and applications in a variety of areas in computer science and electrical engineering: electrical networks, VLSI physical design, systems level testing, communication protocol testing, parallel/distributed computing, telecommunication network planning, interconnection networks, etc.

Dr. Thulasiraman has received several awards and honors including Elected Member of the European Academy of Sciences (2002), the IEEE CAS Society Golden Jubilee Medal (1999), Senior Research Fellowship of the Japan Society for Promotion of Science (1988), and Guest Professorship of the German National Science Foundation (1990). He has been Vice President (Administration) of the IEEE CAS Society (1998, 1999), Technical Program Chair of ISCAS (1993, 1999), Co-Guest Editor of a special issue on "Computational Graph Theory: Algorithms and Applications" for the IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS (March 1988), Associate Editor of the IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS (1989–1991, 1999–2001), and Founding Regional Editor of the *Journal of Circuits, Systems, and Computers*. Recently, he founded the Technical Committee on graph theory and computing of the IEEE CAS Society.