Comparison of Recovery Schemes to Maximize Restorable Throughput in Multicast Networks

S. Suraprasert[a,1], J. Fakcharoenphol[a,1]

[a] *Department of Computer Engineering, Kasetsart University, Bangkok, 10900, Thailand*

Abstract

Multicast networks have many applications especially in real-time content delivery systems. For high-quality services, users do not expect to witness any interruption; thus, network link failure has to be handled gracefully. In unicast networks there are many approaches for dealing with link failures using backup paths. Recently, Cohen and Nakibly categorized these methods, provided linear programming formulations for optimizing network throughput under the assumption that the paths are splitable, and compared them experimentally. In this work, we take their approach and apply to the multicast failure recovery problem. We propose backup bandwidth allocation algorithms based on linear programs to maximize the throughput, and perform an experimental study on the performance of recovery schemes. We study many recovery schemes in multicast networks and propose a new recovery scheme that performs better than all other recovery scheme except the one that recomputed the whole multicast tree from scratch for each link failure.

Keywords: multicast routing, restoration path

1. Introduction

The latest applications in computer networking such as VoIP, video conference, and entertainment require bandwidth and delay time guarantees for the connections between the source node and destination nodes.

In normal operations, there have been many proposed mechanisms for allocating bandwidth to support real-time applications. The term Quality of Service (QoS) is defined as the quality measure for each connection in real-time networks. The QoS is affected by various factors such as dropped packets, delay, jitter, out-of-order, and error . There are many approaches for guaranteeing QoS, e.g., traffic shaping [1], scheduling algorithm [2], congestion avoidance [3].

These techniques are used in normal settings. However, as the number of links grows rather quickly, link failure is another fact of life. For example, network equipments may fail, they may face electrical shortage problems, or even high-speed fiber links can be cut by trucks. These physical failures take a long time to recover, while applications cannot wait.

Therefore, we have to create the mechanism to support real-time applications that tolerates failures. With this mechanism, the bandwidth for the application should be guaranteed not only in normal operations but also in failure situations.

In this paper, we consider backup path framework, the most common approach for QoS when network links are unreliable. Under this framework, primary paths and backup paths are chosen. Primary paths are used to deliver contents in normal situations, while backup paths are used when failure occurs. As in previous work in unicast (e.g., [4, 5]), we assume that primary paths are given and focus mainly on how to choose backup paths.

In unicast networks, there are many proposed mechanisms, usually called recovery schemes, for allocating backup bandwidth (e.g., [5, 6]). In these mechanisms, when an edge fails, parts of the flow from the source are rerouted to the destinations. The assumption for the recovery scheme is that the bandwidth can be

Email addresses: suwaras@gmail.com (S. Suraprasert), jittat@gmail.com (J. Fakcharoenphol)

guaranteed when only single link or node fails. An excellent work of Cohen and Nakibly [4] categorized these methods and assuming that the primary paths are predetermined, presented linear programs for finding the best backup bandwidth allocation maximizing throughput under many recovery schemes.

On the contrary, in multicast networks, since there are many destinations for a flow originating from same source, the rerouting occurs not only on a path but parts of the original multicast routing trees. While destinations that are not affected by the failure use the original multicast routing tree, the destinations that are directly affected by the failure must receive the flows from the backup paths for that failure.

Our work focuses on the recovery scheme for link failure in multicast networks. The network consists of several multicast groups. Each multicast group has a server considered as a source node and its clients considered as destination nodes. The bandwidth in the network is divided into two categories: service bandwidth and restoration bandwidth. The service bandwidth can be shared between the destinations in the same multicast group. While the restoration bandwidth can be shared between the destinations in the same multicast group and can also be shared between *different* multicast groups in *different* failure situations.

Earlier works (e.g., [7, 8, 9, 10]) in multicast failure handling introduced various recovery or backup schemes together with heuristics for bandwidth allocation. Lau, Jha, Banerjee [11] pioneered the use of optimization in multicast backup bandwidth allocation. They presented an on-line Integer Linear Programming (ILP) algorithm for constructing backup paths under Unrestricted Recovery scheme (UR, to be described later), and compared it with the schemes proposed by [7], [12]. Since solving ILP is computationally expensive, Lau *et al.* also designed an on-line approximation algorithm for the problem and compared that with their ILP-based algorithm and the algorithm of [7].

In this paper, as in the first part of [4], we consider the splitable version of the problem where the backup paths can be splitable, i.e., the flow for each path can be fractional. We use linear programs (LP) to construct backup bandwidth allocation under many recovery schemes, including that of [11], that maximize the throughput.

We note that the unsplitable versions are hard to solve optimally. Cohen and Nakibly [4] showed that for the unicast problems the unsplitable versions for many recovery schemes are NP-hard to approximate better than $|E|^{1/2-\epsilon}$ where E is the set of links in the networks. Since the multicast problems are supersets of the unicast ones, their hardness proofs hold for our multicast cases as well. While our work only considers the simpler splitable case, in practice, it might be possible to convert the optimal fractional restoration paths to good solutions to the unsplitable case using randomized rounding techniques (e.g., as in [13]).

As in [4], our method is offline pre-planning to establish the restoration paths. Since the method is offline, the information of multicast groups must be collected beforehand. Also, again, as in Cohen and Nakibly [4], the primary tree for each multicast group has to be established before solving the LPs to find the restoration paths. We discuss how to obtain the primary trees briefly in Section Appendix A.

1.1. Our contributions

Following the work of Cohen and Nakibly [4] in unicast networks, we study many recovery schemes in multicast networks.

The situation in multicast networks is quite different from one in unicast. First of all, in unicast, each group communicates through a predetermined primary path, while in multicast, the primary communication pattern for each group forms a tree. This complicates the formulation of linear programs because the set of affected nodes for a given failure is harder to characterize.

Our first contribution is to introduce a notion of *suspended parts* to unify the linear programming formulation for all the recovery schemes. Using these linear programs, we are able to find the optimal backup bandwidth allocation for many recovery schemes used previously.

For our second contribution, we propose a new recovery scheme for multicast. Under various situations, our experiments show that, among all previously proposed scheme, one recovery scheme, called Unrestricted Recovery scheme (UR), which based on the work by [4], slightly out-performs other recovery schemes. However, finding UR scheme is computationally intensive, and the scheme itself uses as many backup trees as the number of edges in the primary service trees. We then propose a new scheme called **Path Restricted recovery scheme**. The experiment shows that our new scheme reduces the number of pre-computed backup

trees roughly by half and can be computed 8 times faster, while providing almost the same throughput as the UR scheme. Moreover, from our experiments, the computation time of PR is smaller than the computation times of most of other previously proposed schemes.

1.2. Paper organization

The rest of this paper is organized as follows. The related work is discussed in the section 2 and the problem is defined in section 3. Various recovery schemes and our new one are also described in the problem definition section. The linear programming is described in section 4. The experimental setup and result for comparing between recovery schemes are presented in section 5.The discussion in section 6. And the conclusion in section 7.

2. Related Work

Many MPLS networks need high QoS assurance to support the real-time application, such as Voice over IP and video on demand. The QoS provides the availability and reliability by the use of bandwidth reservation for normal operation and failure situations.

We start by reviewing the works in unicast networking and then the works in multicast networking.

Unicast networking. In the unicast networks, many techniques have been developed (see, [4, 5, 6, 14, 15]) to guarantee the bandwidth connectivity when failures occur. For normal operation, paths usually referred as primary paths, together with backup paths in case of failures. Note that the structures of backup paths depend on the recovery schemes.

Many recovery schemes are proposed, and there are mainly two criteria for evaluating them. The first metric is referred to as Spare Capacity Allocation (SCA), where the goal is to satisfy all requests while minimizing the total bandwidth reserved for backup paths.

Under this metric, many techniques are used to minimize the backup bandwidths such as heuristics by Liu, Tipper, and Siripongwutikorn [6], ILP by Shyur, Lu, and Wen [16] and LP by Kennington and Lewis [17].

The main drawback, as argued in the work of Cohen and Nakibly [4], is that when using SCA metric, the capacity of links are assumed to be unlimited or very large. This assumption is not true after the network has already been designed. To be more practical, Cohen and Nakibly [4] propose to use the throughput as the objective. They assume that the bandwidth of each link is limited and measure the total throughput after fractions of the bandwidth are reserved for the recovery scheme.

Under throughput metric, Bhatia, Kodialam, Lakshman, and Sengupta [5] propose an RLR-based recovery scheme. Together with a linear program, they present a Fully Polynomial Time Approximation Scheme(FPTAS) for path restoration against link and node failures.

Cohen and Nakibly [4] have systematically classified unicast recovery schemes (see Section 3), and show that, for many settings, the bandwidth reservation that maximizes throughput can be computed optimally using Linear Programming.

There are other related works that consider other aspects of the problem. Alicherry and Bhatia [14] propose a k-facility local recovery scheme. This heuristic algorithm finds the minimize restoration bandwidth that will be reserved on each edge. While the approach by Lai, Zheng, and Tsai [15] is to establish the bypass tunnel for each pair of source and sink. This work focus mainly on satisfying the QoS constraints when rerouting is needed.

Multicast networking. The study of restoration bandwidth allocation in multicast networks follows those of unicast settings. All previous works concern the SCA metric.

In [8], a backup tree, called a dual-tree, is constructed to handle link failures. A heuristic for computing restoration bandwidth for Restricted Local Recovery scheme that protects node failure is introduced in Kodialam and Lakshman [7]. Aggarwal *et al.* [9] propose simple heuristic algorithm to establish a set of paths from source to every destinations for restoration when link or node failure.

Lau, Jha, Banerjee [11] pioneered the use of optimization in multicast backup bandwidth allocation. They presented an on-line Integer Linear Programming (ILP) algorithm for constructing backup paths

under Unrestricted Recovery scheme(UR),where everything in effected tree(s) is rerouted when each edge failure occurs, and compare it with the schemes proposed by Kodialam and Lakshman [7], and Singhal, Sahasrabuddhe,and Mukherjee [12]. Since solving ILP is computationally expensive, Lau *et al.* also design an on-line approximation algorithm for the problem and compare that with their ILP-based algorithm and the algorithm of Kodialam and Lakshman [7].

Finally, Li, Wang, and Doverspike [10] introduces a RLR-based recovery scheme that protects node failure, and compare their new scheme with the schemes presented in [7] and [9].

3. Problem Definition

We present restorable flow problem in multicast networks. The problem considers the situation on a network when one edge fails and the traffic sent on that edge has be rerouted to maintain the required communication. To be able to route extra flows, some bandwidth on the edge has to be reserved. The goal of this problem is to allocate the bandwidth for a set of multicast trees to get as much profit as possible provided that there is enough bandwidth reserved for rerouting during a failure for one edge to their terminal nodes(or clients).

3.1. Recovery schemes

The problem has many variations depending on how traffic rerouting is performed. Following the work of [4] on unicast, the recovery schemes considered in this paper include:

- **Global Recovery (GR)** — the restoration flows start at root node to the downstream terminal nodes of the failed edge,
- **Local Recovery (LR)** — the restoration flows start at immediate upstream node of the failed edge to the downstream terminal nodes,
- **Restrict Local Recovery (RLR)** — like RLR scheme in unicast, restoration flows start at immediate upstream node to the immediate downstream node,
- **Unrestricted Recovery (UR)** — flows in the entire primary tree are canceled; the restoration flows starts at root and goes to every terminal nodes, i.e., the multicast tree is entirely rebuilt.

In this paper, we also introduce another recovery scheme called **Path Restricted Recovery (PR)**.

In the Unrestricted Recovery scheme (UR), the suspended parts are all edges in the service trees that contain that failed edge. The number of recovery trees for this scheme may be as high as the number of edges in all service trees. This recovery trees must be provisioned and the large number of trees mean the large amount of memory needed to keep the recovery information.

Inspired by UR, we propose a new recovery scheme called Path Restricted Recovery (PR) described as follows.

- **Path Restricted Recovery (PR)** — the service trees are partitioned into a set of maximal disjoint paths shared by the same group of sources, called *failed paths*. Clearly the number of failed paths is less than the number of edges in all service trees. Each failed path P forms a unit of failure to be considered, i.e., failures on any edge in P is handled by the same backup trees. To find the backup trees, all flows on the service trees that contain P are canceled; the restoration flows start at the roots and go to every terminal node, with the restriction that the flows cannot use any edges in P. So, the recovery trees for this failed path are entirely rebuilt. Note that the number of recovery trees is at most the number of failed paths. (See formal definition in Subsection 3.2.1.)

In Section 5.4, we shall see that PR scheme offers many advantages over the original UR and GR, while still providing better throughput than other schemes.

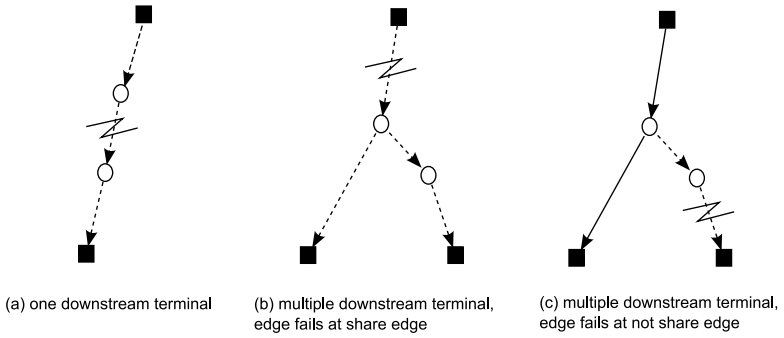

Figure 1: Suspended Parts in Global Recovery

3.2. Formal statements

We are ready to state the problem formally. In the restorable flow problem, we are given a directed network $G = (V, E)$, for each edge $e \in E$, available bandwidth C_e, and a set of k multicast groups. Each multicast group i, for $1 \leq i \leq k$, is specified by a quadruplets $\langle s_i, t_i, D_i, w_i \rangle$, where s_i is the source node, $t_i = \{t_{i1}, t_{i2}, t_{i3}, ..., t_{ij}\}$ is a set of terminal nodes, D_i is the full demand of the group, and w_i is the profit per unit of bandwidth we can satisfy for this group. For each group, we are also given a primary multicast tree T_i that shall be used to route the packets from the source to all terminal nodes.

We want to allocate bandwidth for each multicast group so that when an edge fails, the network has enough bandwidth to reroute the packets according to various recovery schemes. Formally, we want to find, for each multicast group i, the fraction x_i to route the amount of $x_i \cdot D_i$ along the primary tree that maximizes the total profit

$$\sum_{i=1}^{k} w_i \cdot x_i,$$

while having enough unused bandwidth for restoration routing.

For schemes like GR or LR, Cohen and Nakibly [4] view the rerouting as occurring from the source to the destination since they consider the unicast case. However, when dealing with multicast, terminal nodes can be internal nodes of the multicast trees; thus, the rerouting might not occur only from the roots to the leaves of the tree, but to some internal nodes, which are terminal nodes.

When edge failure is detected, the original traffic on the primary trees may be suspended. To unify all recovery schemes, we introduce the notion of *suspended parts*. For the multicast tree T_i, when edge $\tilde{e} \in T_i$ fails, we call the set of edges in T_i that will not carry the original flows as *the suspended parts of T_i from $\tilde{e}$*.

The suspended parts when an edge fails depend on the recovery scheme. For example, for UR and PR, the suspended part is the whole primary tree. For RLR, the suspended part consists of only $\tilde{e}$.

For GR or LR, the suspended part is a bit more complicated.

We first consider GR. Given a failed edge $\tilde{e}$ on primary tree T_i, let u denote the immediate upstream terminal node of $\tilde{e}$ and let B denote the set of immediate downstream terminal nodes of $\tilde{e}$. Let B' denote the set all immediate downstream terminal node from $u_{\tilde{e}}$. Note that $B \subseteq B'$, and $B' - B$ are terminal nodes that is not affected by the failing of $\tilde{e}$. For $v \in B'$, denote the path P_v to be the path in T_i from u to v. The suspended part if we use GR is

$$\bigcup_{v \in B} P_v - \bigcup_{v \in B' - B} P_v.$$

Note that these are the edges carrying flows exclusively for terminal nodes immediate downstream of $\tilde{e}$. (See Figure 1.)

For LR, the suspended part consists of $\tilde{e}$ and edges downstream of $\tilde{e}$ to all immediate downstream terminal nodes B. Figure 2 illustrate this case.

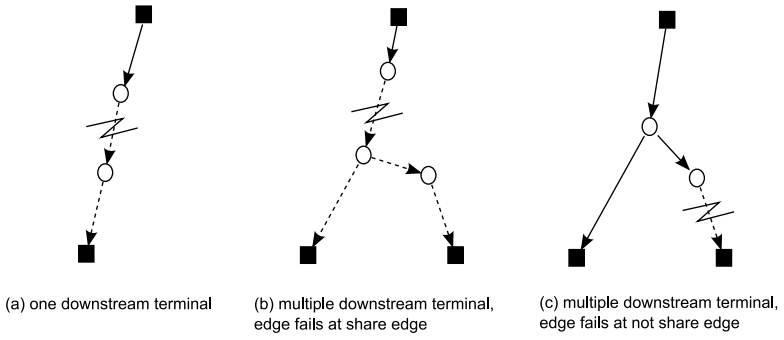

Figure 2: Suspend Parts in Local Recovery

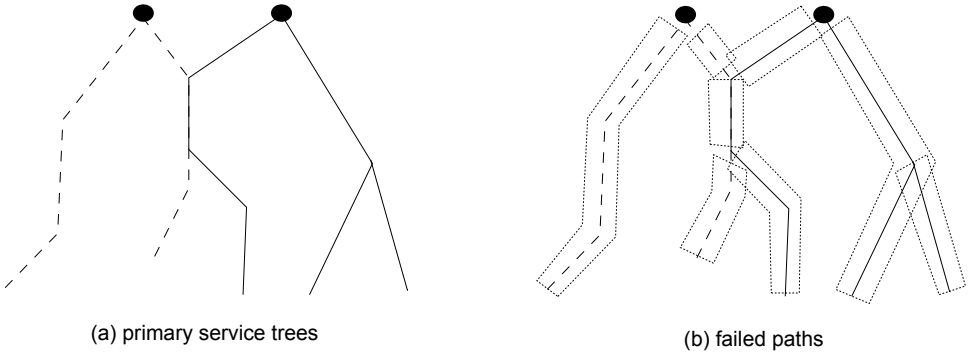

Figure 3: Primary service trees and failed paths

3.2.1. The proposed scheme: Path-Restricted recovery scheme

We are ready to describe how the Path-Restricted recovery scheme (PR) works. The scheme improves the memory requirement over that over that of the UR scheme. While the UR scheme requires one backup tree for each edge in the primary trees, the PR scheme groups edges into a set of maximal edge disjoint paths, called *failed paths*; therefore, when many primary trees share paths, the PR scheme requires a smaller number of trees than the UR scheme.

Figure 3 shows example of failed paths.

To accomplish this design, we need a procedure that builds a list of failed paths from the primary service trees. For each edge e, let the set T_e denote the set of primary trees that use e. We say that edge e and e' belong to the same multicast set iff $T_e = T_{e'}$. Edges in the same multicast set form a forest of rooted trees. Each tree is partitioned into a set of maximal edge disjoint paths in our list in a natural ways. Our procedure repeatedly picks a leaf node v and takes the longest path towards the root not containing any edges used previously. It repeats until all leaves are considered. Each maximal edge disjoint path becomes a failed path in our list.

After that, we establish a recovery tree of the primary service trees that affected by the failure of all edges for each failed path. So, the number of the recovery trees is bound by the number of failed paths instead of the number of edges as in the UR scheme.

The procedure that finds the recovery trees is quite similar to the one in the UR scheme except on the edges not allowed to be in the recovery tree. In the UR scheme, only the failed edge will not be used, but in PR scheme, all edges in the failed path will not be used. The suspended part is still the whole primary service trees that contain each failed path.

	same groups	different groups
same failed edge	shared	not shared
different failed edges	shared	shared

Table 1: Bandwidth sharing

$$
\begin{array}{rll}
\max & \sum_i w_i \cdot x_i & \\
\text{subject to} & & \\
\text{(P-CAP)} & \sum_i m_i(e) \cdot D_i \leq C_e & \forall e \in E \\
\text{(P-FLOW)} & m_i(e) = \begin{cases} x_i & e \in T_i \\ 0 & \text{otherwise,} \end{cases} & \\
 & 0 \leq x_i \leq 1 & \forall i \in \{1, 2, .., k\} \\
 & 0 \leq m_i(e) \leq 1 & \forall e \in E, \forall i \in \{1, 2, .., k\}
\end{array}
$$

Figure 4: Linear program LP-NoRes.

4. Linear programs

In this section, we describe linear programs that, given a set of primary multicast trees, find the most profitable bandwidth allocation for each recovery schemes. We shall compare the results to the optimal routing on the same primary multicast trees with no bandwidth reserved for restoration.

Note that we only consider the case where one edge fails at a time. If that happens, the flow in that edge must be rerouted along the restoration path.

Table 1 gives a short summary on when the reserved bandwidth can be shared. Here we give a brief explanation. Because we assume that at most one edge fails at a time, the bandwidth on a given edge reserved for two edge failures will not be used simultaneously; thus, they can be shared.

Next, we consider the case when a single edge fails. Consider the bandwidth reserved for different terminal nodes. Although the restoration paths might not be the same, the bandwidths are used to deliver the same content; thus, we assume that they can be shared. Clearly, the reserved bandwidth for different multicast groups cannot be shared.

The next subsection presents the linear program for baseline ideal case. Then, five linear programs for methods of restoration considered in this paper are presented.

4.1. Routing in primary multicast trees with no restoration: ideal case

In this section, we present a linear program that allocates the amount of flow to be sent for each multicast group (in terms of fractions of D_i for group i) to maximize the profit. Note that this linear program *does not* reserve any bandwidth for restoration. The goal for this linear program is only to provide a baseline ideal case for comparison.

We define one additional variable.

- $m_i(e)$, for $1 \leq i \leq k$ and for any edge $e \in E$, the fraction of D_i routed over edge e

The linear program, called LP-NoRes, is shown in Figure 4.

Constraints (P-CAP) ensure that the amount of flow in each edge is not greater than its capacity, while constraints (P-FLOW) account for the bandwidth used by each multicast group on each edge. Note that we do not need flow conservation here, because the primary multicast trees already capture that. Linear program LP-NoRes determines the maximum profit given the primary multicast trees.

(B-FLOWBASE)

$$B_i^{\tilde{e}}(e) = \begin{cases} 0 & e \in SP_i^R(\tilde{e}) \\ m_i(e) & \text{otherwise} \end{cases} \qquad \forall \tilde{e} \in E, \forall i \in \{1, 2, ..., k\}$$

(FAILEDGE)

$$\acute{b}_i^{\tilde{e}}(\tilde{e}) = 0 \qquad \forall \tilde{e} \in E, \forall i \in \{1, 2, ..., k\}$$

(CAP)

$$\sum_i \left(\acute{b}_i^{\tilde{e}}(e) + B_i^{\tilde{e}}(e) \right) \cdot D_i \leq C_e \qquad \forall e \in E, \forall \tilde{e} \in E$$

(DEMANDSAT)

$$\sum_{e=(u,t_{ij}) \in E} \left(b_{ij}^{\tilde{e}}(e) + B_i^{\tilde{e}}(e) \right) = x_i \qquad \begin{array}{l} \forall \tilde{e} \in E, \forall i \in \{1, 2, ..., k\}, \\ \forall j \in \{j | t_{ij} \in t_i\} \end{array}$$

(UBOUND)

$$\acute{b}_i^{\tilde{e}}(e) \geq b_{ij}^{\tilde{e}}(e) \qquad \begin{array}{l} \forall e \in E, \forall \tilde{e} \in E, \\ \forall i \in \{1, 2, ..., k\}, \forall j \in \{j | t_{ij} \in t_i\} \end{array}$$

$$0 \leq b_i^{\tilde{e}}(e) \leq 1,\ 0 \leq B_i^{\tilde{e}}(e) \leq 1 \qquad \forall e \in E, \forall \tilde{e} \in E, \forall i \in \{1, 2, ..., k\}$$

Figure 5: Linear program LP-COMMON.

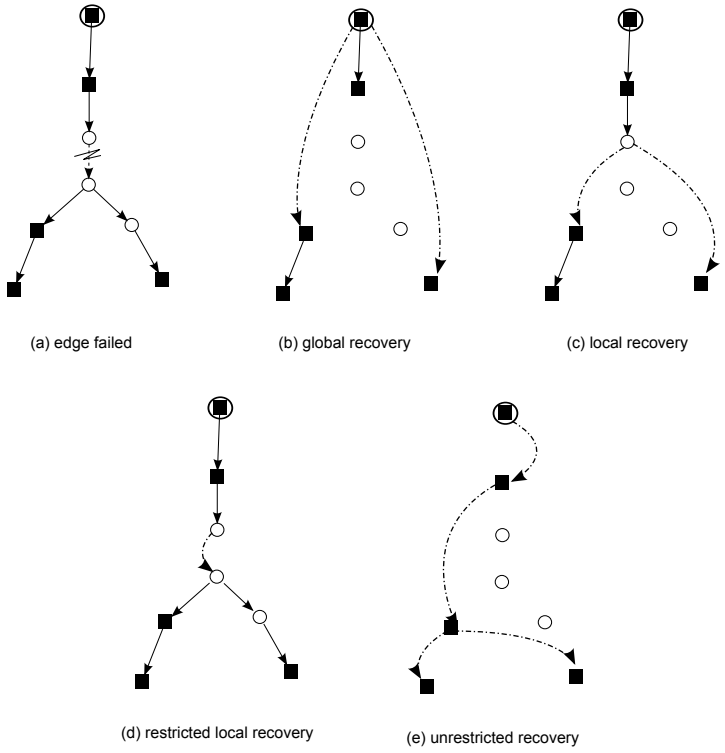

Figure 6: Recovery Schemes in Multicast Networks

(GR-CON)

$$\sum_{e=(u,v)\in E} b_{ij}^{\tilde{e}}(e) = \sum_{e=(v,w)\in E} b_{ij}^{\tilde{e}}(e) \qquad \begin{array}{l}\forall v \in V, \forall \tilde{e} \in E, v \neq s_i, t_{ij}, \\ \forall i \in \{1,2,...,k\}, \forall j \in \{j | t_{ij} \in t_i\}\end{array}$$

Figure 7: Linear program LP-GR.

(LR-CON)

$$\sum_{e=(u,v)\in E} b_{ij}^{\tilde{e}}(e) = \sum_{e=(v,w)\in E} b_{ij}^{\tilde{e}}(e) \qquad \begin{array}{l}\forall v \in V, \forall \tilde{e} = (a,b) \in E, v \neq a, t_{ij}, \\ \forall i \in \{1,2,...,k\}, \forall j \in \{j | t_{ij} \in t_i\}\end{array}$$

Figure 8: Linear program LP-LR.

4.2. Global Recovery (GR).

Main idea of this scheme is to send the restoration flow from the root to terminal nodes affected by the failed edge. When one edge in a multicast tree fails, in GR, the immediate downstream terminal nodes are protected by restoration flows from the source node of multicast group. The flows can be routed on reserved bandwidth and on the suspended parts (as defined in Section 3). The GR is illustrated in figure 6-b.

We are ready to describe the constraints added to LP-NoRes. The linear program, called LP-Common and LP-GR, is shown in Figure 5 and 7.

Call the failed edge $\tilde{e}$. Let $SP_i^R(\tilde{e})$ be the suspended parts of T_i from $\tilde{e}$ for the GR. The first set of constraints (B-flowBase) captures flows that are not affected by the failure, i.e., $B_i^{\tilde{e}}(e)$ is the amount of flows remained after $\tilde{e}$ fails. (Note that $B_i^{\tilde{e}}(e) = 0$ for all edge $e \in SP_i^R(\tilde{e})$.) The constraints (GR-failEdge) ensure that no flows are routed on $\tilde{e}$.

We have to make sure that all terminal nodes receive the same amount of flows (after the restoration); the constraint set (GR-demandSat) takes care of this. Recall that t_{ij} is the j-th terminal nodes of group i. The (GR-con) ensures flow conservation except flow at source and terminal node while (GR-cap) ensures edge capacity constraint.

The next set of constraints (GR-uBound) captures the reservation bandwidth sharing. Finally, the last sets of constraints ensure that variables are within the ranges.

4.3. Local Recovery (LR).

In this scheme, when $\tilde{e}$ fails, an immediate upstream node of $\tilde{e}$ on each multicast group reroutes the flow to maintain the multicast. It stop sending the primary flow along the primary multicast tree from itself to $\tilde{e}$'s immediate downstream terminal nodes, and sends restoration flow from itself to those nodes affected.

In LR the immediate downstream terminal nodes are protected by the immediate upstream node of the failed edge. Thus, LR is different from GR only in that immediate upstream node reroutes the restoration flows instead of the root node. Note that immediate upstream node can be either terminal node or internal node. The LR is illustrated in Figure 6-c.

We are ready to describe the constraints added to LP-NoRes. The linear program, called LP-Common and LP-LR, is shown in Figure 5 and 8. The sets of constraints (LR-cap), (LR-demandSat), (LR-uBound), and (LR-failEdge) are similar to those in LP-GR.

Let $SP_i^R(\tilde{e})$ be the suspended parts of T_i from $\tilde{e}$ for the LR. The following set of constraints (B-flowBase) captures flows that are not affected by the failure as in constraints (B-flowBase) in GR, but the suspended parts are for LR. The (LR-con) ensures flow conservation except flows at immediate upstream nodes and terminal nodes. Again, the last sets of constraints ensure that variables are within the ranges.

(B-FLOWBASE)

$$B_i^{\tilde{e}}(e) = \begin{cases} 0 & \tilde{e} \in E, e = \tilde{e} \\ m_i(e) & \text{otherwise} \end{cases} \qquad \forall \tilde{e} \in E, \forall i \in \{1, 2, ..., k\}$$

(RLR-CAP)

$$\sum_i \left(\acute{b}_i^{\tilde{e}}(e) + B_i^{\tilde{e}}(e) \right) \cdot D_i \leq C_e \qquad \forall e \in E, \forall \tilde{e} \in E$$

(RLR-FAILEDGE)

$$\acute{b}_i^{\tilde{e}}(\tilde{e}) \; = \; 0 \qquad \forall \tilde{e} \in E, \forall i \in \{1, 2, ..., k\}$$

(RLR-DEMANDSAT)

$$\sum_{e=(u,v) \in E} \acute{b}_i^{\tilde{e}}(e) = x_i \qquad \begin{array}{l} \forall \tilde{e} \in E, \forall i \in \{1, 2, .., k\}, \\ \tilde{e} = (a, b), v = b \end{array}$$

(RLR-CON)

$$\sum_{e=(u,v) \in E} \acute{b}_i^{\tilde{e}}(e) = \sum_{e=(v,w) \in E} \acute{b}_i^{\tilde{e}}(e) \qquad \begin{array}{l} \forall \tilde{e} = (a, b) \in E, \; v \in E - \{a, b\}, \\ \forall i \in \{1, 2, ..., k\} \end{array}$$

$$0 \leq b_i^{\tilde{e}}(e) \leq 1, \; 0 \leq B_i^{\tilde{e}}(e) \leq 1 \qquad \forall e \in E, \forall \tilde{e} \in E, \forall i \in \{1, 2, ..., k\}$$

Figure 9: Linear program LP-RLR.

4.4. Restricted Local Recovery (RLR).

In RLR, the recovery process is the same as in the unicast network. The primary flows are canceled on the failed edge. Then the restoration flows are routed from the immediate upstream node to the immediate downstream node of the failed edge to maintain the multicast groups.

In this scheme, every edge on primary trees must be protected from the failure. The RLR is illustrated in figure 6-d.

We are ready to describe the constraints added to LP-NoRes. The linear program, called LP-RLR, is shown in Figure 9.

The sets of constraints (RLR-CAP), and (RLR-FAILEDGE) are similar to those in LP-GR.

The suspended parts of T_i from $\tilde{e}$ for the RLR consists only edge $\tilde{e}$. Set of constraints (B-FLOWBASE) captures flows that are on any edges except the failure edge . The (RLR-DEMANDSAT) ensure that the immediate downstream node receives same amount of flows as in the primary multicast trees. The last sets of constraints ensure that variables are within the ranges.

4.5. Unrestricted Recovery (UR).

The UR differs from the GR only on how it defines the suspended parts. In UR, when an edge fails, the whole tree is recomputed, i.e., there is no restriction on how the new multicast trees are built, hence the name Unrestricted Recovery. Thus, the suspended parts in UR are the entire multicast service trees that contains that failed edge.

To compute the restoration trees, we use linear programming to find new multicast trees after the failed edge gets deleted. The UR is illustrated in figure 6-e.

The UR linear program is same as in GR. But the suspended parts shown in LP-COMMON in UR are all edges in service trees that contain $\tilde{e}$.

We note that UR is expected to give the best performance, since the *entire* multicast tree is recomputed. However, because of the huge overhead in recomputed, we can hardly say that it is practical.

4.6. Path Restricted Recovery (PR).

This scheme is a slight modification of UR, but the unit of failures we deal with is each failed path $\tilde{p}$. When edge $\tilde{e} \in \tilde{p}$ fails, the original flow in service tree are cancel. The restoration trees are established while all other edges $e \in \tilde{p}$ are not use in the restoration trees.

(B-FLOWBASE)

$$B_i^{\tilde{p}}(e) = \begin{cases} 0 & e \in SP_i^R(\tilde{p}) \\ m_i(e) & \text{otherwise} \end{cases} \qquad \forall \tilde{p} \in P, \forall i \in \{1, 2, ..., k\}$$

(FAILEDGE)

$$\acute{b}_i^{\tilde{p}}(\tilde{e}) \; = \; 0 \qquad \forall \tilde{e} \in \tilde{p}, \forall i \in \{1, 2, ..., k\}$$

(CAP)

$$\sum_i \left(\acute{b}_i^{\tilde{p}}(e) + B_i^{\tilde{p}}(e) \right) \cdot D_i \leq C_e \qquad \forall e \in E, \forall \tilde{p} \in P$$

(DEMANDSAT)

$$\sum_{e=(u,t_{ij}) \in E} \left(b_{ij}^{\tilde{p}}(e) + B_i^{\tilde{p}}(e) \right) = x_i \qquad \begin{array}{l} \forall \tilde{p} \in P, \forall i \in \{1, 2, ..., k\}, \\ \forall j \in \{j | t_{ij} \in t_i\} \end{array}$$

(UBOUND)

$$\acute{b}_i^{\tilde{p}}(e) \geq b_{ij}^{\tilde{p}}(e) \qquad \begin{array}{l} \forall e \in E, \forall \tilde{p} \in P, \\ \forall i \in \{1, 2, ..., k\}, \forall j \in \{j | t_{ij} \in t_i\} \end{array}$$

(GR-CON)

$$\sum_{e=(u,v) \in E} b_{ij}^{\tilde{p}}(e) = \sum_{e=(v,w) \in E} b_{ij}^{\tilde{p}}(e) \qquad \begin{array}{l} \forall v \in V, \forall \tilde{p} \in P, v \neq s_i, t_{ij}, \\ \forall i \in \{1, 2, ..., k\}, \forall j \in \{j | t_{ij} \in t_i\} \end{array}$$

$$0 \leq b_i^{\tilde{p}}(e) \leq 1, \; 0 \leq B_i^{\tilde{p}}(e) \leq 1 \qquad \forall e \in E, \forall \tilde{p} \in P, \forall i \in \{1, 2, ..., k\}$$

Figure 10: Linear program LP-PR.

We are ready to describe the constraints added to LP-NoRes. The linear program, called LP-PR, is shown in Figure 10.

5. Simulation study

We use simulations to evaluate the performance of each recovery scheme. Our experiments are described below.

We start by explaining how each simulation instance is generated. The parameters for each instance are the number of nodes n, the number of multicast groups k, node degree d, and the offered load α, which is the ratio between the number of flows and the number of nodes.

5.1. Generating instances

For the network, we use the BRITE simulator [18] to generate an undirected graph under the Barabasi-Albert model [19] as in Cohen and Nikibly [4]. We experiment with graphs with 20 nodes, whose node degrees d are 2 and 3.

Denote that we consider the graphs generated from BRITE simulator as real networks where nodes are consider as routers and edges are consider as links. While the linear programs works in directed graphs, the graphs generated are undirected; therefore, for each undirected edge, we create two directed edges with opposite directions with the same capacity.

Given the graph, we choose k source nodes. The probability that a given node is chosen as a source is proportional to its degree. Note that the distribution of the degrees satisfies the power-law. To generate the set of requests, we choose a client and its source uniformly at random with the exception that each source must have at least one client. The number of requests depends on the offered load, i.e., it equals to $\alpha \cdot n$.

The two additional practical graphs are choose for assess the quality of our solution. First, the US backbone network in [10] consisting of the 28 nodes and 45 links.Second, A carrier backbone network in [10] with 15 nodes and 28 links.

	Carrier-Backbone	US-Backbone	Random-Dense	Random-Sparse
UR	0.965	0.943	0.899	0.719
PR	0.965	0.943	0.897	0.710
GR	0.946	0.924	0.896	0.705
LR	0.941	0.914	0.896	0.704
RLR	0.941	0.914	0.896	0.704

Table 2: Average Performance Ratio

5.2. Building primary multicast trees

To establish primary service trees, we use NNF algorithm that presented in [7]. Then make failed paths by use algorithm in 3.2.1. For completeness, the algorithm for NNF is presented in Appendix A.

5.3. Generating the failed paths for PR

For PR, we have to find a set of failed paths before applying the linear program. The formal description of the implementation of the algorithm outlined in Section 3.2.1 is discussed in Appendix B.

5.4. Comparison of performance ratios

In this section, we evaluate the performance of the recovery schemes for multicast network that describe in previous section.

We perform experiments on two types of networks: randomly generated networks based on the Barabasi-Albert model [19] and real large-scale networks.

For randomly generated instances, we have two parameters (a) A Barabasi-Albert model random graphs with 20 nodes and 59 links (called **Random-Dense**) and (b) A Barabasi-Albert model random graphs with 20 nodes and 37 links (called **Random-Sparse**).

For real networks, we follow [10] and use (c) US backbone network with 28 nodes and 45 links (called **US-Backbone**), and (d) A carrier backbone network with 25 nodes and 28 links (called **Carrier-Backbone**). The number of clients in multicast network is varying from 4 nodes to 40 nodes. For each parameter on each type of network, we perform 50 experiments on all 6 recovery schemes.

The result of each scheme is shown in the term of performance ratio. Where the performance ratio is the ratio between the result of the scheme and the optimal value. The optimal is calculated by another linear program that recompute the entire network when edge failed.In the other words, we have a set of recovery trees for every edge in the primary service trees.

Results are shown in Figures 13 and 14. Table 2 summarizes the average performance ratios for each recovery scheme for each type of networks.

Figure 11 shows the relative performance of several recovery schemes when applied to Carrier Backbone Network. The result shows that the performance of UR and PR are very close. Those two schemes have better performance than GR, LR and RLR by roughly 2%. A similar result can be observed in the US Long Distance Network in figure 12.

When we applied those recovery schemes to networks randomly generated from BRITE simulator [18], the performance of each recovery scheme are almost the same. (See Figure 13 and 14.)

5.5. Comparison of the number of trees

In this section, we compare the upper bound on the number of multicast trees used in the recovery scheme. Note that this is related to the memory space required to implement each recovery scheme.

All recovery schemes, except PR, use each edge as a unit of failure. On the linear programs, they build one tree for each edge in the primary trees. Therefore, we use the number of edges contained in the primary service trees as an upper bound on the number of trees.

However, in the PR scheme the number of trees is bounded by number of maximal disjoint paths from several primary service trees.

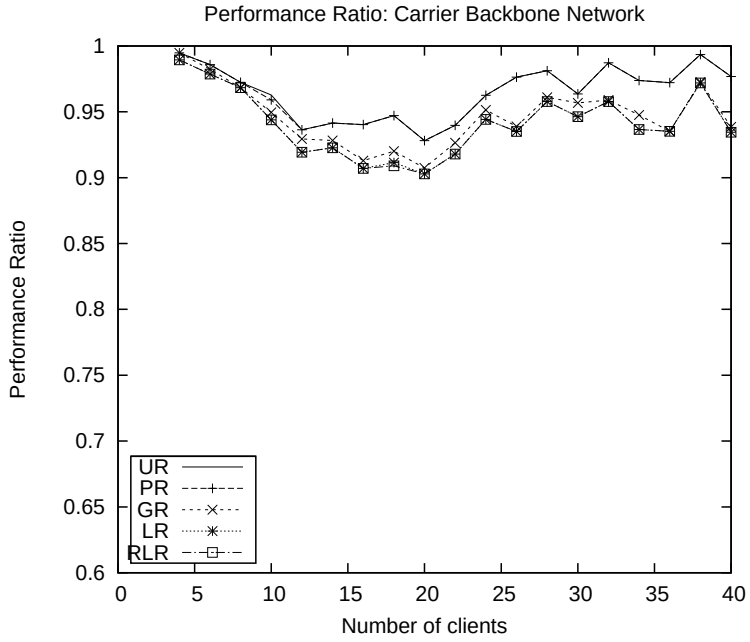

Figure 11: Performance ratio:Carrier Backbone Network

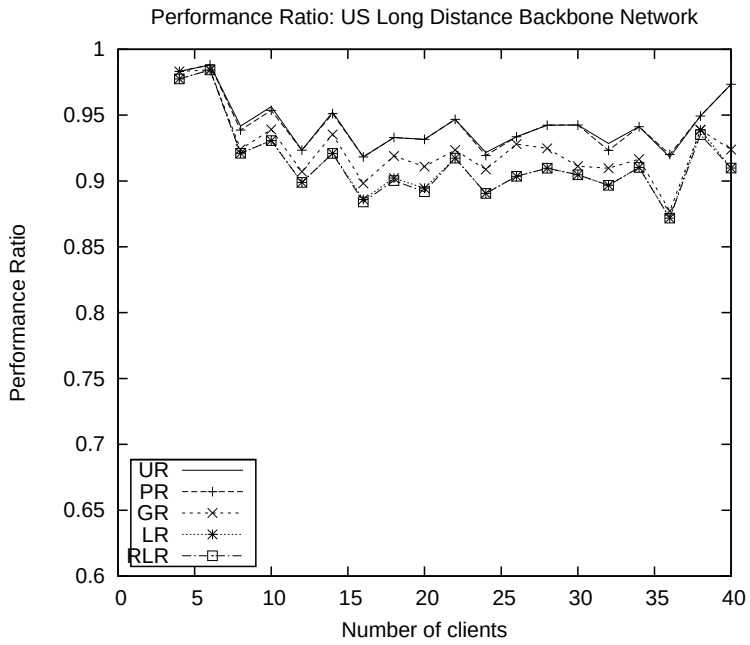

Figure 12: Performance ratio:US Long Distance Network

	Carrier-Backbone	US-Backbone	Random-Dense	Random-Sparse
PR	11.8	13.94	23.31	24.45
Others	26.52	39.41	25.95	27.85

Table 3: Average Number of Recovery Trees

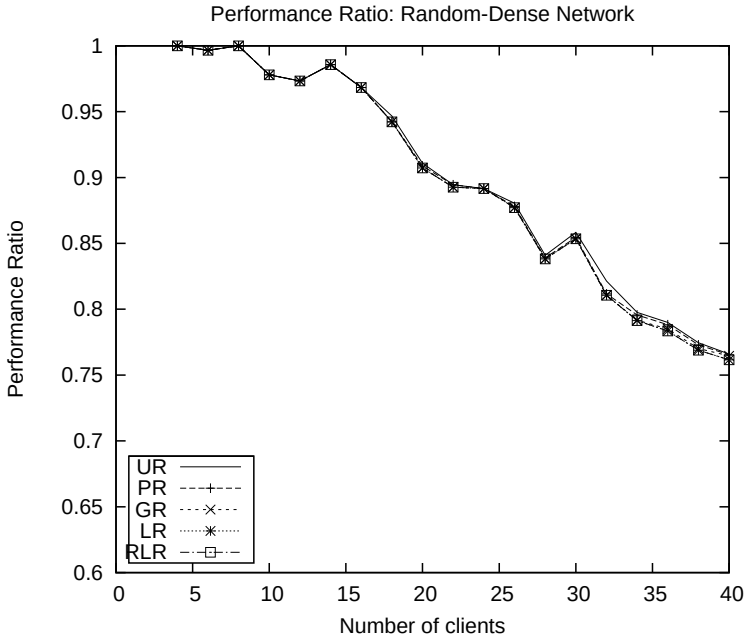

Figure 13: Performance ratio:Random-Dense Network

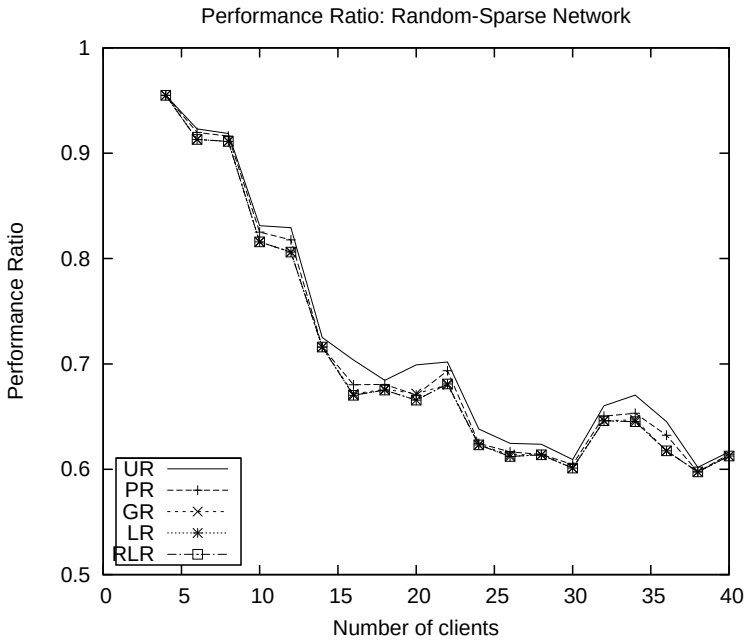

Figure 14: Performance ratio:Random-Sparse Network

	Carrier-Backbone	US-Backbone	Random-Dense	Random-Sparse
UR	23.37	172.98	202.11	59.37
PR	4.11	18.95	18.95	9.47
GR	15.78	100.14	98.53	43.58
LR	14.53	96.63	139.58	42.95
RLR	0.95	3.47	6.32	3.16

Table 4: Average Computation Time(seconds)

The average number of trees of various schemes are shown in Table 3.

From the experiments above, we found that the number of multicast trees that may be required by the UR scheme is 2.5 times more than the number of trees in PR scheme.

5.6. Computation time

Table 4 shows the computation time for each recovery scheme. The computation time is defined by the time that the resolver takes to resolve the linear programs. The PR scheme takes only 1/5 of the time that UR scheme use to resolve the same input.

We note that while PR outperforms UR,GR, and LR in terms of computation time, RLR scheme can be computed much faster than PR. However, RLR gives the worst throughput among the recovery schemes considered in this paper, and, especially in real networks, uses more recovery trees than PR.

6. Discussion

As expected, the result from the previous section shows that the performance of UR is better than other recovery schemes, as it recomputed almost everything after the failures. Our proposed scheme, PR, almost matches the performance of UR; while the number of trees used in PR is roughly 61% the number of trees UR uses and it can be computed roughly 10 times faster.

The performance differences are very small for dense instances. However, the computation time for the PR scheme is about 10 times smaller than that of UR. This is because in this case, edges can be shared quite well among the different edges failed.

We would like to note that the computation time of PR is better that all other recovery scheme except the RLR. However, the throughput of PR is about 3% better than RLR in real networks and about 0.5% better in random instances. However, the RLR need more spaces to keep the recovery paths. The spaces needed by PR are about 38.1% of RLR in real networks and about 88.7% of RLR in random networks.

7. Conclusions and open problems

In this paper, we present LP-based algorithms for maximizing throughput for various recovery schemes under the assumption that flow paths are splitable, and conduct experiments to compare them. We also propose a new recovery scheme, called PR, which is a slight modification of the simple, well-known scheme, UR, and also gives matching performance with UR, which is the best recovery scheme.

We note that there are rooms for improvement. First, we have not investigated on how to get the best primary multicast tree. If fractional trees are allowed, the linear programming approaches as in this paper maybe applicable. Secondly, instead of choosing the recovery scheme manually, it may be possible to use linear programs to also find the best recovery scheme for a given network.

Also, this work only compares various recovery schemes experimentally. It would be very useful to have a more rigorous mathematical analysis on their performance ratios. We leave this as our future work.

References

[1] R. Engel, D. Kandlur, D. K, A. Mehra, and D. Saha, "Exploring the performance impact of qos support in tcp/ip protocol stacks," in *in Proc. IEEE INFOCOM*, 1998.

[2] G. Hasegawa, T. Matsuo, M. Murata, and H. Miyahara, "Comparisons of packet scheduling algorithms for fair service among connections on the internet," *J. High Speed Netw.*, vol. 12, no. 1-2, pp. 1–27, 2003.

[3] V. Jacobson, "Congestion avoidance and control," *SIGCOMM Comput. Commun. Rev.*, vol. 25, no. 1, pp. 157–187, 1995.

[4] R. Cohen and G. Nakibly, "Maximizing restorable throughput in mpls networks," *IEEE/ACM Trans. Netw.*, vol. 18, pp. 568–581, April 2010. [Online]. Available: http://dx.doi.org/10.1109/TNET.2009.2031064

[5] R. Bhatia, M. S. Kodialam, T. V. Lakshman, and S. Sengupta, "Bandwidth guaranteed routing with fast restoration against link and node failures," *IEEE/ACM Trans. Netw.*, vol. 16, no. 6, pp. 1321–1330, 2008.

[6] Y. Liu, D. Tipper, and P. Siripongwutikorn, "Approximating optimal spare capacity allocation by successive survivable routing," in *in Proc. IEEE INFOCOM*, 2001, pp. 699–708.

[7] M. S. Kodialam and T. V. Lakshman, "Dynamic routing of bandwidth guaranteed multicasts with failure backup," in *ICNP*. IEEE Computer Society, 2002, pp. 259–269.

[8] A. Fei, J. Cui, M. Gerla, and D. Cavendish, "A dual-tree scheme for fault-tolerant multicast," in *In Proceedings of IEEE ICC01*, 2001.

[9] R. A. et al, "Extension to rsvp-te for point to multipoint te lsps," *IETF Internet draft*, 2005.

[10] G. Li, D. Wang, and R. D. Doverspike, "Efficient distributed mpls p2mp fast reroute," in *INFOCOM*, 2006.

[11] W. Lau, S. Jha, and S. Banerjee, "Efficient bandwidth guaranteed restoration algorithms for multicast connections," in *NETWORKING*, ser. Lecture Notes in Computer Science, R. Boutaba, K. C. Almeroth, R. Puigjaner, S. X. Shen, and J. P. Black, Eds., vol. 3462. Springer, 2005, pp. 1005–1017.

[12] N. K. Singhal, L. H. Sahasrabuddhe, and B. Mukherjee, "Provisioning of survivable multicast sessions against single link failures in optical wdm mesh networks," *Journal of Lightwave Technology*, vol. 21, no. 11, 2003.

[13] P. Raghavan and C. D. Thompson, "Randomized rounding: a technique for provably good algorithms and algorithmic proofs," *Combinatorica*, vol. 7, no. 4, pp. 365–374, 1987.

[14] M. Alicherry and R. Bhatia, "Simple pre-provisioning scheme to enable fast restoration," *IEEE/ACM Trans. Netw.*, vol. 15, pp. 400–412, April 2007. [Online]. Available: http://dx.doi.org/10.1109/TNET.2007.892844

[15] W. K. Lai, Z. C. Zheng, and C.-D. Tsai, "Fast reroute with pre-established bypass tunnel in mpls," *Comput. Commun.*, vol. 31, pp. 1660–1671, June 2008. [Online]. Available: http://dl.acm.org/citation.cfm?id=1371257.1371298

[16] C.-C. Shyur, T.-C. Lu, and U.-P. Wen, "Applying tabu search to spare capacity planning for network restoration," *Comput. Oper. Res.*, vol. 26, no. 12, pp. 1175–1194, 1999.

[17] J. L. Kennington and M. W. Lewis, "The path restoration version of the spare capacity allocation problem with modularity restrictions: Models, algorithms, and an empirical analysis," *INFORMS J. on Computing*, vol. 13, no. 3, pp. 181–190, 2001.

[18] A. Medina, A. Lakhina, I. Matta, and J. Byers, "Brite: An approach to universal topology generation," 2001.

[19] A.-L. Barabasi, R. Albert, and H. Jeong, "Scale-free characteristics of random networks: the topology of the world-wide web," *Physica A: Statistical Mechanics and its Applications*, vol. 281, no. 1-4, pp. 69–77, June 2000. [Online]. Available: http://dx.doi.org/10.1016/S0378-4371(00)00018-2

Appendix A. Building primary multicast trees

To compute the primary service trees we use the most well-known heuristic algorithm, called Nearest Neighbor First (NNF), presented in [7]. Given the source s and a set of clients D, the algorithm for computing the service tree, presented here for completeness, is shown below.

function NNF(s, D)
$N = \{s\}$
$P = \emptyset$
While $D \neq \emptyset$ do
  $\forall x \in D$, find $y \in D$ such as $distance(y, N) \leq distance(x, N)$
  $N = N \cup NodeBetween(y, N) \cup y$
  $P = P \cup PathBetween(y, N)$
  $D = D - y$
$T = (N, P)$
Return T

The algorithm iteratively adds paths from the closest unconnected client y to the current tree (whose node set is N with paths P). The algorithm uses function $distance(v, N)$ that returns the shortest distance from v to some node in N. Functions $NodeBetween(v, N)$ and $PathBetween(v, N)$ return the set of nodes and edges in the shortest path from v to any node in N.

Appendix B. Generate the failed paths

The Path Restricted Recovery scheme presented in this paper builds recovery trees for each failed path from the primary service trees.

This section presents an algorithm for constructing a list of failed paths from the service trees. The algorithm decomposes the service trees into several edge disjoint paths.

The following subroutine GetPaths decomposes a tree in to a set of edge disjoint paths.

function GetPaths(T,D)
$L = \{\emptyset\}$
foreach $x \in D$
  $p = PathToRoot(x, T)$
  foreach $l \in L$, $p = p - l$
  $L = L \cup \{p\}$
Return L

The algorithm maintains a set of client nodes D and uses function $PathToRoot(x, T)$ that returns a unique path from x to the root node. Before adding paths p to the solution L, we delete edges in p that have already appeared in L, so that paths for each service tree in L are edge-disjoint.

Failed paths from all service trees may share edges. The following procedure constructs the set of failed paths by repeatedly finding an intersecting pair of failed paths and spitting them so that they do not intersect.

$\mathcal{P} = \{\emptyset\}$
$\mathcal{R} = \{\emptyset\}$
foreach $T_i \in \mathcal{T}$, $\mathcal{R} = \mathcal{R} \cup GetPath(T_i, D_i)$
while $\mathcal{R} \neq \emptyset$
  let x be some path in $\mathcal{R}$
  $\mathcal{R} = \mathcal{R} - \{x\}$
  if there exists $y \in \mathcal{R}$ such that $x \cap y \neq \emptyset$ **then**
    $\mathcal{R} = \mathcal{R} - \{y\}$
    $\mathcal{R} = \mathcal{R} \cup \{x - y\} \cup \{y - x\} \cup \{x \cap y\}$ (*)
  else
    $\mathcal{P} = \mathcal{P} \cup \{x\}$
return $\mathcal{P}$

The algorithm starts by collecting initial failed paths from all service trees to set $\mathcal{R}$. It picks a pivot path x from $\mathcal{R}$, removes it from $\mathcal{R}$, finds an intersecting path y, and splits them into $x - y$, $y - x$ and $x \cup y$ (in line (*) above). Note that the algorithm should handle case that $x - y$, $y - x$, and $x \cup y$ may contains more than one path, when adding them back to $\mathcal{R}$.

If there is no paths intersecting with x, it saves x to the output set $\mathcal{P}$.

The algorithm may repeat the main loop for many times. But note that each time, it splits a pair of paths into shorter ones; which can happens for at most the number edges in the graph. Thus, the algorithm runs in polynomial time.