Path Protection with Pre-identification for MPLS Networks

Dan Wang, Funda Ergun
School of Computer Science, Simon Fraser University
Burnaby BC, Canada, V5A 1S6
e-mail: {danw, funda}@cs.sfu.ca

Abstract

Current approaches to providing robust network connections which are tolerant to failures involve restoration schemes which mainly focus on reserving backup paths. In this paper we propose a technique for which avoids the extra cost for reserving, by pre- identifying (but not reserving) the backup paths. We present and analyze an algorithm to solve this problem and study a practical special case in detail. Through simulations we show that our model is significantly more cost-efficient than backup path reservation. We also show how this model can fit into the MPLS architecture.

1 Introduction

In order to provide high-quality service to an ever growing class of reliable, time- and data-sensitive, lucrative applications such as virtual lease line services, stock exchange data services, video services, a fast recovery mechanism from network failures is desired. Current routing algorithms focus mainly on finding a path and assume that the paths will be recomputed in the event of a link or node failure. An alternative, more responsive approach is *protecting* the links being used via *backup paths*.

The path supporting the transmission in normal conditions is called the *primary path*. Backup paths (or restoration paths) are secondary paths to be used in case the primary path fails. Currently, network backup schemes assume that the backup paths are reserved; that is, even when not in use, the links of the unused backup path cannot be used by other applications. While guaranteeing high reliability, this approach has two problems: 1) it unnecessarily increases the cost of the application, and 2) it does not consider the possibility of simultaneous failures in the primary *and* the backup path. These two limitations make a reservation-based protection scheme better suited for networks with low failure rates and for applications where the users are willing to pay a high premium for increased reliability.

In this paper, we are interested in investigating reliable and cost efficient ways of allocating multiple paths for flows in a network so that in case of a failure an (unreserved) backup path may be quickly substituted for the original path. Our technique is general enough to be used under several platforms, such as Multi-Protocol Label Switching (MPLS [9]), which was introduced mainly for providing flexible routing services. In MPLS, a path is computed at the source and is pre-established and implemented through the use of labels. The flexible nature of MPLS allows the implementation of various routing mechanisms with more complicated criteria as well as failure detection and restoration mechanisms. RFC 3469 [10] has suggested the possibility of pre-identifying backup paths without reserving them; however, not much detailed work has been done on this approach. The main issues in using this type of scheme arise from the design and implementation of the pre-identification process and the possibility that a backup path might be unavailable when a connection needs to use it, which we address in this paper.

In our approach, we propose to pre-identify backup paths but not to reserve them. We present a generalized model where the primary and the backup paths are assigned priorities, consistent with a model of current networks where flows are classified by their "importance". [1] In this model, higher priority flows are allowed to preempt lower priority flows whenever there is any shortage of bandwidth; this will be our only assumption regarding how flows with different priorities are treated. In our model backup paths are of lower cost and lower priority than primary paths. This is in contrast with the reservation model where the backup paths are priced similarly as primary paths since reservation effectively grants the owner infinite priority.

In the presence of priorities, there is a tradeoff between the availability of a backup path and its cost. In particular, there is no guarantee that a backup path will be available to a flow in case of a primary path failure since it may be in use by a higher priority application. The reliability is fur-

[1] In practice, importance may solely depend on how much the owner of the flow is willing to pay for using the network.

ther lessened by the possibility that the backup path itself may be down due to a failure. In the absence of a reservation scheme and the possibility of multiple link failures, we desire to develop a technique which obtains high reliability in terms of the probability of finding a *usable* backup path. This is achieved by pre-identifying multiple backup paths protecting each link on the primary path.

The reliability of our scheme depends on (1) the number of backup paths for each link on the primary path, (2) the failure probability of each backup path, and (3) the priority/preemption schemes. We deal with (1) and (2) by choosing numerous robust paths; for (3) we do not explore particular priority schemes to exploit their individual properties (for details of different preemption schemes, one may refer to [7, 8] etc); rather, we identify *edge-disjoint paths* so that two different backup paths will not be preempted by the same flow and the reliability will remain mostly independent of the preemption scheme.

2 Our Restoration Schemes

In this section we present our unreserved restoration model; however as a gentle introduction we start off with our first scheme which is a generalization of two existing schemes and can be used in a reservation-based setup.

We represent the network as a graph $G(V, E)$. Let $\mathcal{M} = (s = v_0, v_1, \cdots, t = v_K)$ be the primary path; the nodes (resp. links) on this path are called *primary nodes* (resp. *links*). We say v_j is *upstream* (resp. *downstream*) of v_i if $j < i$ (resp. $i < j$). The *latency* of link e, denoted $d(e)$, is the time for an error notification message to traverse e. The cost of e is denoted $c(e)$.

2.1 A Generalized Restoration Scheme

In this section, we generalize end-to-end restoration and local restoration. We define the notion of *loss-bounded restoration* where the number of packets dropped due to notification delay is bounded (See Figure 1).

Definition 1 *A loss-bounded restoration scheme for $\mathcal{M}$ upper bound $\mathcal{L}$ is a restoration scheme where, in the case of the failure of any link (v_i, v_{i+1}), the packet loss incurred by the signaling (notification) delay from $v_i \in \mathcal{M}$ to an upstream node $v_j \in \mathcal{M}$ is at most $\mathcal{L}$.*

We call the maximum notification delay that the connection can afford without losing more than $\mathcal{L}$ packets the *tolerance*. The farthest upstream node to which a node v can afford to send a notification message in case of a failure is called the *upstream range* of v. Tolerance can be estimated by dividing $\mathcal{L}$ by the transmission speed. Given the tolerance, the upstream range can be computed using the individual link latencies in the network. We say a primary link (v_i, v_{i+1}) can be *protected* by a backup (protection) path starting from v_k and ending at v_l, if v_k is in the upstream range of v_i and v_l is downstream of v_{i+1}. This model generalizes that introduced in [6] where the notification messages are allowed to travel a limited number of hops.

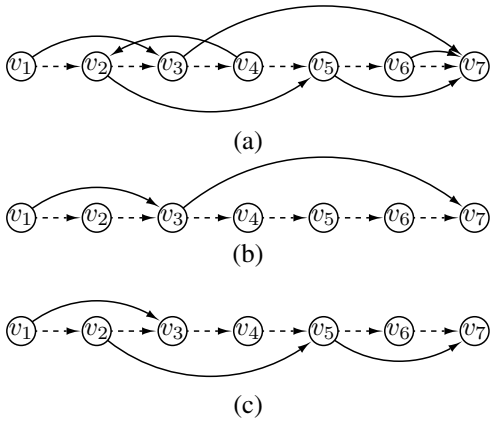

Figure 1. Different restorations for $P = (v_1 \cdots v_7)$. Each link has latency and cost 1. (a) local restoration: max latency is 0 and cost is 6. (b) end-to-end restoration: max latency is $d(v_6) = 3$ and cost is 2. (c) loss-bounded restoration: max latency is $d(v_4) = 2$ and cost is 3.

2.2 Unreserved Restoration Using Pre-identified Paths

The restoration schemes discussed in the previous section are independent of whether backup paths are reserved. We will now develop an unreserved loss-bounded (in short, unreserved) restoration scheme which pre-identifies, but does not reserve the restoration paths so that in case of a link failure a *usable* restoration path satisfying the loss bound can be found with high probability:

Definition 2 *An unreserved loss-bounded restoration scheme (in short unreserved restoration) for $\mathcal{M}$ with probability T is a loss-bounded restoration scheme where, in case of failure of any link (v_i, v_{i+1}), a usable loss bounded alternative path will be found with probability at least T.*

To compute and bound the successful restoration probability of each link, we use the failure probabilities of restoration paths. Let $\mathcal{P}_j$ represent the protection path starting from node v_j, and $f(\mathcal{P}_j)$ the failure probability of $\mathcal{P}_j$. If $f(\mathcal{P}_j) > 1 - T$, we need to identify multiple paths so that the probability that at least one will be usable will be at least T. We will show later how we estimate the success probabilities of individual as well as groups of paths.

We assume that we can have flows of different priorities, represented as an ordered list $(r_1, \cdots, r_M)$ (r_M is the highest priority), where a flow can preempt a lower priority one.

Integrating into MPLS The implementation of our techniques require a few modifications to existing protection schemes, such as in MPLS. In MPLS the primary and backup paths are identified during the setup phase, where the primary path is labeled with the highest priority and the backup paths are with a low priority. When a backup path becomes unusable, a notification message is distributed as in [8]; the primary nodes will then mark this path as unavailable. When a failure occurs on the primary path, a notification message is sent upstream from the failure point. Whenever a switching node with available backup paths is reached, the traffic is sent on this backup path.

In this paper our objective is to find a *min-cost* set of edge-disjoint backup paths. Several other works [4, 5] try to minimize bandwidth use so that backup paths can be shared. Our algorithm can be adjusted to that scenario, where two requests can share the pre-identified paths.

3 On the Complexity of an Exact Solution

We show here that the problem is hard for both the directed and undirected cases, even when cost is not an issue.

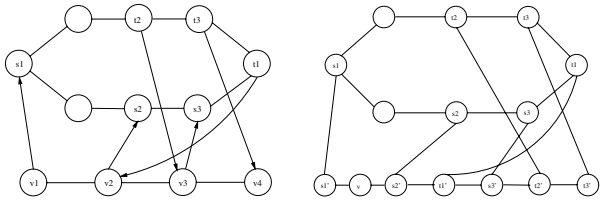

Figure 2. Constructing G': Directed, Undirected Cases

Theorem 1 *Finding an unreserved restoration in an directed network is NP hard.*

Proof: We present a reduction from the (directed) edge-disjoint path problem (EDP) [1], defined as follows. Given a set of K source-destination pairs (s_i, t_i), find K edge-disjoint (s_i, t_i) paths, one for each (s_i, t_i) pair.

To solve an arbitrary instance of EDP on a network G, we build an unreserved restoration problem for a network G' constructed from G as follows. We add to G a primary path $(v_1, v_2, \cdots, v_{K+1})$ constructed from new nodes v_i. We then add links (v_i, s_i) and (t_i, v_{i+1}) for each i (See Figure 2 for an example). We set the loss bounds small enough (e.g., 0) so that the required loss-bounded restoration degenerates into local restoration. We set the failure probability to 0 for all links. The only valid protection for this is where, for all primary nodes, each protection path starts at link (v_i, s_i), and ends at (t_i, v_{i+1}). To see this, consider a valid unreserved (local) restoration on G'. Consider the first restoration path from the left starting at v_i and ending at v_j such that $i \neq j - 1$. Then, there must be a restoration path starting at v_k and ending on v_{j-1} such that $j - 1 < k$, a contradiction. Thus, EDP has a solution on G if and only if unreserved restoration has a solution on G'. ■

The next theorem shows that the undirected unreserved restoration problem is NP-hard. The proof of Theorem 1 can not be extended directly to undirected networks, since by omitting the directions, the protection path may start from v_i to s_i and come back through s_{i+1} to v_{i+1}, as well as from v_i to t_{i-1}, coming back through t_i to v_{i+1}. This is not a valid solution for the edge-disjoint path problem.

Theorem 2 *Finding an unreserved restoration in an undirected network is NP hard.*

Proof: We reduce from the undirected edge-disjoint path problem (UEDP) [3]. For each instance of UEDP on a network G with source-destination pairs (s_i, t_i), (e.g., Figure 2) construct network G' by adding a new primary path $(s'_1, v, s'_2, t'_1, s'_3, t'_2, \cdots, s'_i, t'_{i-1}, \cdots, s'_k, t'_{k-1}, t'_k)$. For each i, add links (s'_i, s_i) and (t'_i, t_i). All links have a failure probability 0. Set the latencies so that each notification message can be sent to at most one node upstream.

There is a unique solution for unreserved restoration where the restoration paths start at s'_i, go through (s'_i, s_i), (t_i, t'_i) and end at t'_i, i.e., all (s'_i, s_i) are outgoing and all (t_i, t'_i) are incoming links for the protection. Deleting all the added links in the solution for unreserved restoration, the solution for edge-disjoint paths is obtained.

We call a primary node outgoing (resp. incoming), if it is the starting (resp. ending) node of a protection path. Observe that, (a) the number of outgoing nodes is equal to that of incoming nodes, (b) except for the final node and the one preceding it, any incoming node u_i must be followed by an outgoing node u_{i+1} so that the link (u_{i+1}, u_{i+2}) is protected, c) for any node u in the primary path the number of outgoing nodes minus the number of incoming nodes upstream of u is at most 2; otherwise, due to (a), (b) would be violated.

We now consider the primary nodes. s'_1 and s'_2 must be outgoing; otherwise either (s'_1, v) or (s'_2, t'_1) is unprotected. After this, all t'_i are incoming and all s'_i are outgoing, by induction on the primary nodes, left to right on the primary path. Consider some s'_i. To the left of s'_i is t_{i-2}, which by induction, is incoming. Thus, by (b), s'_i must be outgoing. Now, consider some t'_j. To its left is s'_{j+1}, which is outgoing, which, due to our inductive hypothesis and the first nodes s'_1, s'_2, implies that the number of outgoing nodes to the left of t_j is two more than incoming nodes. So as not to

violate (c), t_j cannot be outgoing. It cannot be unused either, since this would leave (t_j, s_{j+2}) unprotected. Thus it must be incoming. Finally, due to (a), t_K must be incoming.

To see that the restoration will match all s_i to t_i, consider any restoration path from s_i to t_j. It must be that $i \leq j+1$ to make the restoration valid. If $i = j+1$ then the (t_j, s_{j+2}) is unprotected, thus we must have $i \leq j$. With an argument similar to the proof of Theorem 1, this implies that any restoration path starting at s_i must end at t_i. Removing the added nodes/links will give a solution for UEDP. ■

Our algorithm works for both the directed and undirected cases. We will present the undirected case, pointing out any differences from the directed.

4 A Special Case

In this section, we set the failure probability of every backup path to be the same, and estimate the failure probability of a set of backup paths as the product of their individual failure probabilities. We then obtain the number of paths, say $\mathcal{I}$, that we need to pre-identify for each primary link. In addition to its usefulness when the information is scarce or when the preemption probabilities for different priority classes vary dramatically (see Section 6), more importantly, the solution for this special case will later be used as a building block for solving the general problem.

First consider protecting one primary link (v_i, v_{i+1}). We start off by calculating the upstream range, say v_j. We then add a new node s and links $(s, v_j), (s, v_{j+1}), \cdots, (s, v_i)$, as well as a new node t and links $(t, v_{i+1}), \cdots, (t, v_K)$. Setting all links to unit capacity and solving a min-cost flow problem of sending $\mathcal{I}$ units of flow from s to t solves the problem efficiently. Unfortunately, for the general problem, repeating the above procedure for all K primary links is too costly, since adjacent primary links on the primary path can share backup paths.

Integer Programming Formulation. Let c_{ij} denote the cost on link (i, j). Let primary nodes v_k, v_l denote the source and destination of a backup path, where $1 \leq k < l \leq K$. x_{ij}^{kl} denotes the flow on link (i, j) of backup path from v_k to v_l. N is a boolean matrix such that $N_j^{kl} = 1$ iff the backup path from v_k to v_l protects link (v_j, v_{j+1}) and a_{kl} denotes the number of backup paths selected.

$$min \sum_{\forall i,j} \sum_{k=1}^{K} \sum_{l=k}^{K} c_{ij} x_{ij}^{kl}$$

$$s.t. \quad \forall k, l \in \mathcal{M}, \ \sum_j x_{kj}^{kl} - \sum_j x_{jk}^{kl} = a_{kl} \quad (1)$$

$$\forall k, k', l \in \mathcal{M}, \ k \neq k', \ \sum_j x_{k'j}^{kl} - \sum_j x_{jk'}^{kl} = 0 \quad (2)$$

$$\forall i \ \sum_{j,k,l} x_{ij}^{kl} - \sum_{j,k,l} x_{ji}^{kl} = 0 \quad (3)$$

$$\forall k, l \in \mathcal{M}, \ \sum_j x_{lj}^{kl} - \sum_j x_{jl}^{kl} = -a_{kl} \quad (4)$$

$$\forall k, l, l' \in \mathcal{M}, \ l \neq l', \ \sum_j x_{l'j}^{kl} - \sum_j x_{jl'}^{kl} = 0 \quad (5)$$

$$\forall i, j \ \sum_{kl} x_{ij}^{kl} \leq 1 \quad (6)$$

$$\forall j \in \mathcal{M} \ a_{kl} N_j^{kl} \geq \mathcal{I} \quad (7)$$

(1) says that the output flow of a backup path between v_k, v_l on the primary path must be a_{kl}, which is constrained by (7). (2) says that v_k' doesn't send flow for backup path v_k to v_l. (3) says that the input and output flow on each intermediate link (i, j) must be balanced. (4) and (5) are the destination based equivalents of of (1) and (2). (6) enforces edge-disjointness and (7) says that all primary links (v_j, v_{j+1}) are protected by $\mathcal{I}$ backup paths.

4.1 The Algorithm: General Approach

Without (7) and modified (1), (4), the above integer program reduces to edge-disjoint paths with the set of source-destination pairs decided by a_{kl}. We develop an algorithm which relaxes inequality (7) and iteratively approaches the min-cost solution. We start off by finding a set of (possibly non-disjoint) paths providing a lower bound on the optimal cost, then refine this solution to achieve a set of disjoint paths. We present our algorithm in two steps.

Step 1. Let S be a set of primary node pairs $\langle v_j, v_i \rangle$, with cost $c(v_j, v_i)$, where a corresponding backup path for this pair can protect the links between v_j and v_i. Find a min-cost $S_{sub} \subseteq S$ that provides a valid protection for $\mathcal{M}$. This may lead the resulting backup paths (corresponding to pairs $\langle v_j, v_i \rangle$) not disjoint (min-cost infeasible).

Step 2. Find edge-disjoint paths for S_{sub}, where sources and destinations are $\langle v_j, v_i \rangle \in S_{sub}$. The resulting backup paths are feasible but may not be min-cost.

Clearly, Steps 1 and 2 have a primal-dual relationship as a result of relaxing inequality (7) of the integer program to obtain two separate problems which aim to improve the cost and feasibility respectively.

As an example, consider Figure 3. Set the upstream range to 1 and obtain one backup path for each primary link. First we compute the initial value for each $\langle v_j, v_i \rangle$, where v_j is within the upstream range of v_i, which is the min-cost path from v_j to any node downstream of v_i. For instance, the backup path for $\langle v_1, v_2 \rangle$ is $(v_1, b_1, b_4, b_6, v_3)$ of cost 4, which, by connecting an artificial node to v_2, v_3, v_4, v_5, can be computed by one-round of Dijkstra, breaking ties arbitrarily. Figure 5 shows the paths chosen

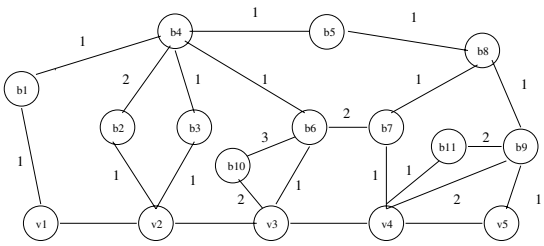

Figure 3. The network with link costs.

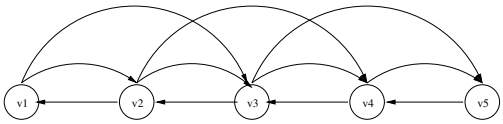

Figure 4. Auxiliary network G_A for Step 1

and their costs. During Iteration 1, in Step 1 we compute $S_{sub} = \{\langle v_1, v_3\rangle, \langle v_3, v_5\rangle\}$ with cost 10. In Step 2, we calculate edge-disjoint paths for the pairs in S_{sub}, obtaining $\langle v_1, b_1, b_4, b_6, v_3\rangle$ and $\langle v_3, b_{10}, b_6, b_7, b_8, b_9, v_5\rangle$. We then modify the costs, setting $c(v_1, v_3) = 4$, $c(v_3, v_5) = 10$. (Others remain unchanged). In Iteration 2, in Step 1 $S_{sub} = \{\langle v_1, v_3\rangle, \langle v_2, v_4\rangle, \langle v_4, v_5\rangle\}$ with cost 13. In Step 2, disjoint paths for S_{sub} are $\langle v_1, b_1, b_4, b_6, v_3\rangle$, $\langle v_2, b_3, b_4, b_5, b_8, b_7, v_4\rangle$ and $\langle v_4, b_9, v_5\rangle$. We then set $c(v_1, v_3) = 4$, $c(v_2, v_4) = 6$, $c(v_4, v_5) = 3$. In Iteration 3 the paths remain unchanged; we halt.

4.2 The Building Blocks

We now present the building blocks for our algorithm.

Step 1: Algorithm Build_Assignment

Construct a directed auxiliary graph G_A (Figure 4): Set the primary links in the opposite direction of the primary path. [2] For each primary node pair $\langle v_j, v_i\rangle$, add link (v_j, v_i) with cost $c(v_j, v_i)$. Set the cost of the primary links in reverse direction to 0. Set the capacity to 1 on each link. Compute the min-cost flow on G_A where $\mathcal{I}$ units of flow is sent from for $s = v_1$ to $v_K = t$.

Step 2: Algorithm Edge_Disjoint

```
1 repeat
2     for each input pair ⟨v_j, v_i⟩
3         Compute the shortest path from v_j to v_i.
4         if disjoint, exit.
5         else
6             increase the cost of the overlapping link
7             by l where there are l paths crossing it.
8     if the number of iteration ≥ MAX,
9         output no disjoint path exists.
```

[2] Notice that we build an intermediate directed graph for our algorithm.

$v_j - v_i$ *pairs*	*path*	$c(v_j, v_i)$
$\langle v_1, v_2\rangle$	$(v_1, b_1, b_4, b_6, v_3)$	4
$\langle v_1, v_3\rangle$	$(v_1, b_1, b_4, b_6, v_3)$	4
$\langle v_2, v_3\rangle$	$(v_2, b_2, b_4, b_6, v_3)$	5
$\langle v_2, v_4\rangle$	$(v_2, b_2, b_4, b_6, b_7, v_4)$	6
$\langle v_3, v_4\rangle$	(v_3, b_6, b_7, v_4)	3
$\langle v_3, v_5\rangle$	$(v_3, b_6, b_7, b_8, b_9, v_5)$	6
$\langle v_4, v_5\rangle$	(v_4, b_9, v_5)	3

Figure 5. $\langle v_j, v_i\rangle$ pairs.

End Algorithm

Many relaxation-based heuristics can be found for the edge-disjoint path problem. We just simply increase the cost of overlapping links to discourage paths from using them (line 6-7).

4.3 Putting It Together: The Main Algorithm

Algorithm UBP

```
1  initialization
2    add destinations t_1, t_2 ··· , t_{K-1}.
3    connect t_i to v_{i+1}, v_{i+2}, ··· , v_K.
4    set c(v_{i+j}, t_i) = 0, ∀j = 1 ··· K.
5    ∀v_i, compute upstream range of v_i.
6    for each node v_j and v_i within its range
7      compute the shortest path p from v_j to t_{i-1}
8      set c(v_j, v_i) = c(p).
9  repeat
10   run Build_Assignment
11   run Edge_Disjoint [3]
12   if no change in paths terminate.
13   else if new cost > previous cost
14         ĉ(v_i, v_j) = (1/k) c^f(v_i, v_j) + ((k-1)/k) c^n(v_i, v_j)
15     else
16         ĉ(v_i, v_j) = (1/2) c^f(v_i, v_j) + (1/2) c^n(v_i, v_j).
```

End Algorithm

During initialization we set the cost of the $\langle v_j, v_i\rangle$ pairs to the cost of the shortest paths. Recall that a backup path that can protect pair $\langle v_j, v_i\rangle$ will start at v_j and end at any point downstream v_i; thus we add destinations t_i, where t_i is connected to v_i and all the primary nodes downstream of v_i. In line 11, Edge_disjoint receives a subset of $\langle v_j, v_i\rangle$ pairs to find disjoint backup paths. To do this, we keep in mind that v_i might not be the destination for this backup path. Therefore, we extend v_i to t_{i-1}, and compute edge-disjoint paths for $\langle v_j, t_{i-1}\rangle$ instead.

[3] We need to find edge disjoint paths for $\langle v_j, v_i\rangle$ pairs that are provided by Build_Assignment. The v_i should be extended to the corresponding t_{i-1}, since a protection path for $\langle v_j, v_i\rangle$ pairs can end at any point downstream of v_i.

We now specify how we adjust the costs during each iteration so that the primal and dual will converge towards each other. We maintain the initial cost c^f the previous cost c_i^p, the new cost c_i^n, and the relaxed cost $\hat{c}_i$ during each iteration i. [4] If the new cost c_i^n is less than the previous cost c_i^p, i.e. we obtain an improved solution, the previous cost for next iteration c_{i+1}^p is set to the current new cost c_i^n and the relaxed cost $\hat{c}_{i+1}$ is set to $\frac{1}{2}(c^f + c_i^n)$. Otherwise we keep the previous cost, and the relaxed cost is set as $\hat{c}_{i+1} = \frac{1}{k}c^f + \frac{(k-1)}{k}c_i^n$, giving more weight to the changed cost to sacrifice cost for feasibility. [5]

4.3.1 Analysis of the Algorithm

We first show the optimality of the output.

Theorem 3 *Given a set of $\langle v_j, v_i \rangle$ pairs, representing a set of backup paths which can protect all primary links between v_j and v_i, as well as $\mathcal{I}$, the number of protection path needed to ensure the failure probability, Build_Assignment can precisely find a subset of the paths with optimal cost.*

Proof: First, we prove that our algorithm outputs a feasible protection. In G_A, for every primary link, there are no "artificial" protection links that are out of the upstream range. Therefore, if there are enough protection links covering any primary link, it is enough to say that the protection is feasible. To see this, note that we send $\mathcal{I}$ number of units of flow from the source to the destination. If we cut G_A into two, where the first piece includes v_i and all nodes upstream of it, and the second piece v_{i+1} and its downstream nodes, there will be at least $\mathcal{I}$ units of flow across the cut, indicating that (v_i, v_{i+1}) is protected $\mathcal{I}$ times.

To prove optimality, assume that there is a cheaper feasible protection. By using this protection choice, we must be able to send $\mathcal{I}$ units of flow from s to t. This contradicts the optimality of our min cost flow. ■

Note that the cost of each $\langle v_j, v_i \rangle$ pair is non-increasing after the first iteration, thus Algorithm UBP will terminate.

5 General Unreserved Restoration

We now show how to solve the general problem without advance knowledge on how many backup paths are needed. We use the failure probability of each link to compute our restoration paths. We estimate the failure probability of a path as the product of the failure probabilities of its links, and that of a a group of paths as the product of the failure probability of each path. That our backup paths are edge-disjoint helps with the independence of these likelihoods.

[4]The subscripts are not used in the algorithm.

[5]This is one relaxation technique that we found to be efficient and simple to implement; other usable relaxation methods can also be used.

Here we describe our general technique by presenting the changes to be made to the algorithms we previously presented. Our implementations cover both the special and the general cases.

In Build_Assignment, we use augmenting paths to solve the min-cost flow problem. Here we augment paths one by one and, when the process complete, we estimate the success probabilities and remove the extra paths. [6] In Edge_Disjoint, we use the probability as an additional constraint. Thus, we add a weight to the probability and combine it with the cost, so that when the algorithm chooses the links, it can rule out the links with high cost as well as high failure probability. The weight factor is explained in the next section. The main algorithm remains unchanged.

6 Simulation Results

$\|V\|$	p_i	$avg\ len$	$\#\ paths$	$suc\ ratio$
40	99%	6	2	99.66%
40	90%	6	7	99.50%
40	80%	6	16	99.23%
40	90%	6	3	89.71%
40	80%	6	8	91.21%
40	70%	6	18	89.49%
100	99.9%	12	2	99.99%
100	97%	12	8	99.99%
100	90%	12	23	99.99%

Figure 6. Success ratios. Column 2 denotes the probability of not being preempted; Column 3 the average length of a backup path; Column 4 the number of backup paths needed for each primary link.

Figure 6 shows the relationship between the preemption probabilities and the number of paths needed. We see that if the preemption robustness (probability of non preemption) is 90% (vs. 99 %), the number of backup paths needed increases to 7 from 2. In practice, if we expect even larger difference between different preemption probabilities, it is more practical to buy higher priorities for the links than try to pre-identify many links. In that case, our algorithms for the special case apply after the priority is chosen, and the number of backup paths and the failure probability of a path can be approximately decided in advance.

[6]Picking out a min cost path from a set of paths such that the failure probability is bounded is hard; however, we start with a bounded number of such paths and in our implementation remove those with the highest failure probability.

6.1 Simulations

To test our technique, we performed extensive simulations using the widely adopted Waxman Model [11] to generate random networks. The network consists of 40 to 70 nodes which reflects a realistic network [2]. Parameters α and β which respectively reflect the density of the short links and the density of the network are set to 0.3 and 0.6. The grid is 30×30 and the cost on each link its Euclidean length. The latency on each link is 100 so that we will not switch our focus to the loss-bound constraint; our algorithm will work for varying latencies as well.

We set the primary path as the shortest path between two randomly chosen nodes s and t. The cost of a restoration is the sum of the cost of each reserved link. In the unreserved case, the cost applies only when the path is used. The likelihood, duration, and cost of the usage of protection paths are usually small. We consider the worst case and assume that one protection path is always in use. Thus, in our experiments, the cost of an unreserved restoration is total cost of the links divided by the number of protection paths. Each data point is the average of 100 experiments.

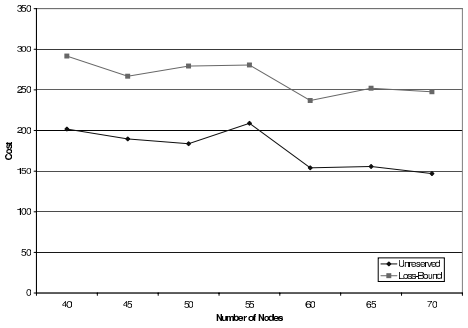

Figure 7. Special case; 2 paths are pre-identified. Max latency is 150, one jump upstream is allowed

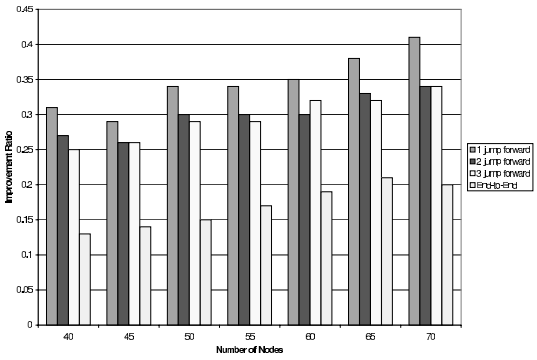

Figure 8. Special case; 2 paths are pre-identified. Varying number of upstream jumps.

We first compare the costs of unreserved restorations vs. reservation (Figure 7); the cost of unreserved restoration is over 30% lower than the loss-bounded restoration. This trend is present in all our experiments, not surprisingly, since in unreserved restoration, only one path is in use at any time, as opposed to several. We then consider how the different upstream ranges affect the cost. As the upstream range becomes larger, the scheme approaches end-to-end restoration, which is less costly. The "percentage" of the unreserved links becomes larger and the gain we have for the unreserved restoration reduces (Figure 8).

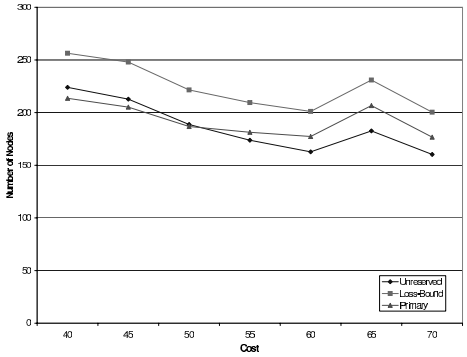

Figure 9. The latency is set to 150, one jump upstream is allowed

In end-to-end restoration, only one cheapest backup path is needed for protection. This leads one to expect that the unreserved restoration might possibly perform worse than end-to-end restoration since the cost of unreserved restoration is the average of multiple paths. However, this is not the case as shown in Figure 8. We can see that when we set the upstream range to infinity, effectively using end-to-end restoration, unreserved restoration still outperforms end-to-end restoration, although by a smaller margin. The reason is that in unreserved restoration, we only use a bridge (i.e. part of the backup path where only the starting and ending node are on the primary path) each time. In end-to-end restoration, we need to reserve a "path" to protection the entire primary path. This restoration path may well consist of several bridges (Figure 1). Figures 8 and 9 show that as the length of the primary path increases, this trend is even more obvious. In Figure 9, we can see that the cost of the primary path itself grows higher than the restoration cost as the size of the network is greater than 50.

Figure 10 shows the general case. We can see that to ensuring 90% success is costlier than to ensuring 80% success. However, compared to loss-bounded restoration, the cost is much smaller. Figure 11 is the general case corresponding to the special case in Figure 8. We see that the upstream range increases, the cost reduction ratio reduces; however, even in the extreme case, i.e. with end-to-end restoration,

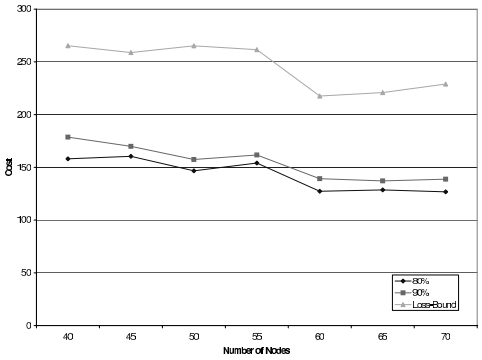

Figure 10. The latency is set to 150, one jump upstream is allowed

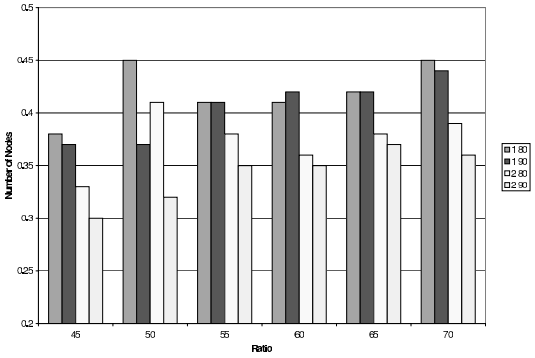

Figure 11. The latency is set to 150, one jump upstream is allowed

the unreserved scheme is still much more cost efficient than the reservation based scheme. The trend is clearer in larger networks.

7 Conclusion

In this paper we consider a restoration scheme which does not reserve bandwidth in advance, but tries to guarantee the availability of a backup path by identifying multiple possibilities *a priori*. This leads to a more cost-efficient way of achieving path protection where applications can have a certain level of traffic guarantees. The restoration scheme bridges the gap of the crucial applications and the general applications by exploring the tradeoff between cost and different degrees of reliability. We show in our simulations that the cost of our scheme is substantially lower than reserved restoration schemes.

References

[1] S. Fortune, J. Hopcroft and A. Wyllie *The Directed Subgraph Homeomorphism Problem*, Theoretical Computer Science, Vol. 10, No.2 pp.111-121, 1980.

[2] A. Juttner, B. Szviatovszki, I. Mecs and Z. Rajko *Lagrange Relaxation Based Method for the QoS Routing Problem*, IEEE, INFOCOM 2001.

[3] R. Karp, *On the Computational Complexity of Combinatorial problems*, Networks 5, pp. 45-68 1975.

[4] M. Kodialam and T. V. Lakshman, *Dynamic Routing of Bandwidth Guaranteed Tunnels with Restoration*, IEEE, INFOCOM 2000.

[5] M. Kodialam and T. V. Lakshman, *Dynamic Routing of Locally Restorable Bandwidth Guaranteed Tunnels Using Aggregated Link Usage Information*, IEEE, INFOCOM 2001.

[6] L. Li, M. Buddhikot, C. Chekuri and K. Guo *Routing Bandwidth Guaranteed Paths with Local Restoration in Label Switched Networks*, IEEE ICNP, Nov. 2002.

[7] J. Oliveira, et al, *A New Preemption Policies for Diffserv-Aware Traffic Engineering to Minimize Rerouting*, IEEE INFOCOM 2002.

[8] J. Oliveira, JP. Vasseur, L. Chen and C. Scoglio, *LSP Preemption Policies for MPLS Traffic Engineering*, IETF Internet Draft draft-deoliveira-diff-te-preemption-02.txt.

[9] E. Rosen, A. Viswanathan, and R. Collan, *Multiprotocol Label Switching Architecture*, IETF RFC 3031, Jan. 2001.

[10] V. Sharma and F. Hellstrand, *Framework for Multi-Protocol Label Switching (MPLS)-based Recovery*, IETF RFC 3469, Feb. 2003.

[11] B. M. Waxman, *Routing of multipoint connections*, IEEE Journal on Selected Areas in Cummunications, 6(9): 1617-1622, 1988.