# Routing Bandwidth Guaranteed Paths with Local Restoration in Label Switched Networks

Li (Erran) Li, Milind M. Buddhikot, Chandra Chekuri, Katherine Guo

***Abstract*— The emerging Multi-Protocol Label Switching (MPLS) networks enable network service providers to route bandwidth guaranteed paths between customer sites [3], [2], [8], [5]. This basic Label Switched Path (LSP) routing is often enhanced using restoration routing which sets up alternate LSPs to guarantee uninterrupted connectivity in case network links or nodes along primary path fail. In this paper, we address the problem of distributed routing of restoration paths, which can be defined as follows: given a request for a bandwidth guaranteed LSP between two nodes, find a primary LSP and a set of backup LSPs that protect the links along the primary LSP. A routing algorithm that computes these paths must optimize the restoration latency and the amount of bandwidth used.**

**In this paper, we introduce the concept of "backtracking" to bound the restoration latency. We consider three different cases characterized by a parameter called backtracking distance $D$: (1) no backtracking ($D = 0$), (2) limited backtracking ($D = k$), and (3) unlimited backtracking ($D = \infty$). We use a link cost model that captures bandwidth sharing among links using various types of aggregate link state information. We first show that joint optimization of primary and backup paths is NP-hard in all cases. We then consider algorithms that compute primary and backup paths in two separate steps. Using link cost metrics that capture bandwidth sharing, we devise heuristics for each case. Our simulation study shows that these algorithms offer a way to tradeoff bandwidth to meet a range of restoration latency requirements.**

***Index Terms*— MPLS, traffic management, routing**

## I. Introduction

The emerging Multi-Protocol Label Switching (MPLS) networks enable network service providers (NSPs) to setup policy and quality-of-service (QoS) constrained label switched paths (LSPs) between network nodes. The QoS constraint in the form of minimum or peak bandwidth guarantee per LSP has been considered most commonly in literature[3], [1], [10]. Such constraint-based routing [3], [5] is central to network traffic engineering [3], [2] and basic constructs of several new network services such as layer-3 provider provisioned VPNs (PPVPN) [14] and layer-2 PPVPNs [4]. Specific examples of such constructs are the VPN tunnels in L3 PPVPNs, and the Virtual Private Wire Service (VPWS) and Virtual Private LAN Service (VPLS) in L2 PPVPNs. The two main steps in setting up LSPs are (1) computing a path that satisfies required constraints and (2) establishing and maintaining forwarding state along that path. Clearly, the routing algorithm used in step 1 is a basic building block for new network services.

Li (Erran) Li, Milind M. Buddhikot, Chandra Chekuri, Katherine Guo are with Bell Labs, Lucent Technologies, NJ, USA (e-mail: {erranlli,mbuddhikot,kguo}@bell-labs.com, chekuri@research.bell-labs.com). An earlier version of this paper appeared in the Proceedings of IEEE International Conference on Network Protocols (ICNP2002), Paris.

Note that the failure of nodes or links along LSPs leads to service disruption. The NSPs have considered enhancing the reliability and availability of new services using *restoration* routing, which sets up alternate paths to carry traffic under fault conditions. There are two variants of restoration routing: (1) *End-to-end restoration:* routes two link disjoint paths – one primary path and one backup path between every source, destination node pair [9], [12]. Resources are always reserved on the backup and are guaranteed to be available when the backup path is activated. In the event of a failure along the primary path, the source node detects path failure and activates the backup path. In the absence of an explicit signaling protocol, source node learns of link failures via intra-domain routing updates such as OSPF link state advertisements (LSA) packets or IS-IS Link State Packets (IS-IS LSP) which are processed in the slow-path typically in a routing daemon in the router or switch controller. Also, such packets are typically generated by timer driven events. This causes very large restoration latencies of the order of seconds or at best 100s of milliseconds. (2) *Local restoration:* routes a primary path between the source and destination and a set of paths that protect the links along the primary path [11]. It can minimize the need for source node to be involved in the path restoration and therefore can achieve fast restoration. Though local restoration may use more (bandwidth) resources, it is attractive as it can meet stringent restoration latency requirements that are often of the order of 50 ms – similar to existing SONET protection mechanisms [17].

The problem of end-to-end and local restoration routing has been addressed in several recent research papers [9], [12]. In this paper, we primarily focus on the problem of local restoration routing. We first describe the new concept of restoration routing with *backtracking* characterized by a single parameter called *backtracking distance* $D$. We consider three cases: (1) no backtracking ($D = 0$), (2) limited backtracking ($D = k$), and (3) unlimited backtracking ($D = \infty$). We first consider the joint optimization of primary and backup paths and show that the problem is NP-hard. We therefore propose computation of primary and backup paths in two steps. In the case of $D = 0$ and $D = k$, we show that even the two-step optimization problem is NP-hard, whereas for $D = \infty$ an optimal solution exists for computation of backup path. We then describe heuristics that use per-link network state in the form of (1) residual link capacity ($R_l$), (2) bandwidth consumed by primary paths ($F_l$), (3) bandwidth consumed by backup paths ($A_l$) and optionally a fixed sized Backup Load Distribution matrix [9], [12]. We also present simulation results to characterize the performance of these algorithms. We show that our algorithms offer a means to tradeoff bandwidth to meet a range of restoration latency requirements.

## A. Related Work

The MPLS Forum has proposed use of constraint-based routing to setup "protection LSPs" to bypass failed links and nodes [3]. In this scheme, the upstream router detects the link failure and tunnels all traffic on that link along a pre-established MPLS LSP implemented using the label stacking mechanism. Clearly, this mechanism can be extended to selectively reroute certain LSPs on the link instead of the entire link traffic. Extensions have been proposed to the RSVP resource reservation protocol to signal such a protection LSP [7]. However, the existing fast reroute model does not require protection LSPs to have bandwidth guarantees and aims to provide continuation of best-effort connectivity. The orthogonal question of how to route such best-effort or bandwidth guaranteed protection LSPs is still a topic of investigation. Our work provides an algorithmic framework and candidate algorithms to solve this important problem.

The paper by Kodialam et al.[11] represents the current state-of-the art approach for local restoration. Their algorithm iteratively uses a shortest path algorithm to find restoration path for each link in the primary path. The shortest path algorithm is invoked for each edge in the network. Therefore, the running time of their algorithm is $O(mnlogn+m^2)$ where $m$ is the number of edges in the network and $n$ is the number of nodes in the network. In contrast, our running time is $O(hnlogn + hm)$ where $h$ is the primary path length for $D < \infty$ and a constant for $D = \infty$. Their algorithm does not explicitly encourage link sharing among multiple backup paths for the same primary path and therefore, in the worst case may compute backup paths that are largely link disjoint. Their work also does not provide a theoretical and algorithmic framework for local restoration that allows a network designer to tradeoff bandwidth for better restoration latency. Our research reported here overcomes these limitations.

## B. Outline of the Paper

The outline of the rest of this paper is as follows: Section II presents the background material for the remaining discussion in the paper. Section III introduces the concept of restoration routing with backtracking and describes the three cases we consider. In Section IV, we show that the joint optimization of primary path and corresponding local restoration subgraph is NP-hard and advocate a two-step approach for computing them. Section V describes in detail backup subgraph computation algorithms for $D = 0, k, \infty$ cases. The concept of post-processing to achieve further bandwidth savings and the post-processing algorithms for each case are discussed in Section VI. Section VII describes our simulation experiments in detail. Finally, Section VIII presents our conclusions and future directions.

# II. BACKGROUND

In this section, we will present relevant background material on various aspects of the problem of routing bandwidth guaranteed backup paths.

## A. Network and Service Model

The label switched network is modeled as a graph $G = (V, E)$, $|V| = n$, $|E| = m$ where graph nodes correspond to the network nodes and graph edges correspond to the physical links between the nodes. Each link in fact consists of two simplex links in opposite direction: one for the transmit path and the other for the receive path. The graph nodes can be classified into two types[3]: (1) Label Edge Router (LER): These are nodes at which the network traffic enters or leaves the network. Such nodes are connected to the customer edge (CE) routers which use the transport services of the LSP service provider, (2) Label Switched Router (LSR): these are the transit routers that forward packets based on the MPLS labels.

Each LER independently receives requests from connected CEs to setup bandwidth guaranteed paths. Each such request $r$ is modeled as a 3-tuple $r = (s, d, b)$ where $s$ is the source node, $d$ is the destination node and $b$ is the bandwidth requirement [4]. In MPLS networks, an LSP between $s$ and $d$ is a simplex flow, that is, packets flow in one direction from $s$ to $d$ along a constrained routed path [3]. For reverse traffic flow, additional simplex LSP must be computed and routed from $d$ to $s$. Clearly, the path from $s$ to $d$ can be different from the path from $d$ to $s$. Also, the amount of bandwidth reserved on each paths can be different. This request model is often referred to as *pipe* model in the VPN literature [3]. We call this model and corresponding constrained path routing *asymmetric request* model. The algorithms reported in this paper assume this request model.

## B. Fault Model

In the context of protected path routing it is important to consider two kinds of failures, namely link failures and router failures. A common fault model for link failures assumed in the literature and justified by network measurements is that at any given time only one link in the network fails [17]. In other words, in the event of a link failure, no other link fails until the failed link is repaired. Although this model do not deal with failure of bundled links. Addressing more catastrophic failures like bundled links will result in more backup bandwidth reservation. The single link failure model is a good tradeoff between the amount of backup bandwidth reservation and the severity of the failures it can deal with. We consider single link failure in this paper. We do not address router failure case due to space constraints.

## C. Bandwidth Sharing

We distinguish bandwidth sharing into two categories: (1) *Inter-request sharing*: The single fault model dictates that two link-disjoint primary LSPs corresponding to two requests each of $b$ units do not fail simultaneously. This allows them to share a single backup path of $b$ units. In other words, inter-request bandwidth sharing allows one of the two primary paths to use the backup links "for free." (2) *Intra-request sharing*: In local restoration, since each link on the primary path requires a backup path, and only one link failure can happen at any given time, the backup paths for different links in the same primary path can share links. Also, backup paths can share primary path links. Notice this type of sharing does not exist in the end-to-end path restoration schemes.

Our goal is to develop online distributed local restoration routing algorithms that utilize both inter-request sharing and intra-request sharing in order to minimize the total bandwidth reserved.

### D. Network State Information

In our work we focus on online routing algorithms that route a new path request based only on the knowledge of *current* state of the network and do not exploit properties of *future* requests. One can design schemes with varying degrees of partial state. Kodialam et al. [10] describes one such partial information scenario, referred to as the $O(|E|)$ information case here on, wherein for every link $l$ in network $G = (V, E)$ with capacity $C_l$, three state variables are maintained and exchanged among peering routers: (1) $F_l$: Amount of bandwidth used on link $l$ by all primary paths that use link $l$. (2) $A_l$: Amount of bandwidth reserved by all backup paths that contain link $l$. (3) $R_l$: Residual capacity on the link $l$ defined as $C_l - (F_l + A_l)$. Norden et al.[12] proposed algorithms that use these three per-link state variables and a new form of state called Backup Load Distribution (BLD) matrix. Specifically, given a network with $m$ links, each network node (router or switch) maintains a $m \times m$ BLD matrix. If the primary load $F_j$ on a link $j$ is $b$ units, entries $BLDM[i, j], 1 \leq i \leq m, j \neq i$, record what fraction of $b$ is backed up on link $i$. Note that this approach, which we call as the $O(|E|^2)$ information case, allows maximum bandwidth sharing. The algorithms we propose in this paper work with both cases of state information.

### E. Modeling Link Cost

Each link (interchangeably called edge) has $C_l, R_l, F_l, A_l$ state variables associated with it and may be assigned one or more link costs. In response to the tunnel request $r = (s, d, b)$, the source node uses its knowledge of network state such as topology, $C_l, R_l, A_l, F_l$ for all links $l \in E$ and optionally $BLD$ matrix to compute two things: (1) a *primary path* $P = (s = u_1, u_2, \cdots d = u_k)$ where edge $l_i = (u_i, u_{i+1}) \in E$, (2) a subgraph $G_{sub} = (V_{sub}, E_{sub})$ such that a backup path exists in $G_{sub}$ that can route traffic from $s$ to $d$ if any link $l_i$ fails.

The computation of $P$ and $G_{sub}$ is based on two kinds of link costs described below.

*Cost of link $l$ in the primary path*: If a link is used in the primary path $P$ of request $r = (s, d, b)$, then $b$ units of bandwidth has to be consumed on each link in $P$. Therefore the cost $\mu_l$ of using a link $l$ in the primary path is $b$ if $R_l \geq b$, otherwise $\infty$.

*Cost of link $l$ in backup path*: Consider a primary path $P$ with $m$ links $(1, 2, ...m)$. Let us consider a representative link $l$ that is a candidate link in a backup path $LB_i$ for link $i \in P$. We are interested in characterizing the cost of using $l$. The amount of bandwidth that can be used on $l$ for free to backup $i$ is $FR[l, i] = A_l -$ Backup load induced by $i$ on $l$ before request $r$. In the $O(|E|)$ information case, free bandwidth is $FR[l, i] = A_l - F_i$, whereas in the $O(|E|^2)$ information case, the free bandwidth on $l$ is $FR[l, i] = A_l - BLDM[l, i]$. For the rest of the paper we focus only on the $O(|E|^2)$ information case. However, all our results apply equally to the $O(|E|)$ information case.

From the perspective of backup routing, every link has two kinds of bandwidth available: (1) *Free bandwidth (FR):* that is completely sharable and does not require extra resource reservation. (2) *Residual bandwidth (R):* is the actual capacity left unused on the link. If the LSP request size $b > FR_l$, then $b - FR_l$ units of bandwidth must be allocated on the link to account for the worst case backup load on the link. If the residual bandwidth $R_l$ falls short of $b - FR_l$ (i.e $b - FR_l > R_l$), then the link $l$ cannot be used on the backup path and is called an *"infeasible link"*. Given this, the cost of using link $l$ on a backup path to backup link $i$ on primary path consists of two parts: (1) cost of using the free bandwidth on the link and (2) cost of using the residual bandwidth on the link. Equation 1 illustrates the exact form of cost $w[l, i]$ incurred to backup link $i$ on $l$. The cost metrics $C_F$ $(C_R)$ should be selected in such a way that selecting a link with high residual capacity $R_l$, results in smaller cost. See [12] for more details on these cost metrics.

Since there are $m$ links in the primary path, each candidate link $l$ has $m$ associated costs $w[l, 1], w[l, 2] \cdots w[l, m]$. Given that a link $l$ may be used in backup paths for multiple links $i$ in primary path, to make the problem tractable, we have to follow a pessimistic approach and reduce the free available bandwidth for $l$ as follows:

$$
\begin{aligned}
FR_l &= min_{i \in E(P)} FR[l, i] \\
&= A_l - max_{i \in E(P)} BLDM[l, i]
\end{aligned} \tag{3}
$$

where $E(P)$ denotes the edge set used in path $P$. Correspondingly, the cost $w_l$ of using link $l$ on the backup path for any link in $P$ can be computed using Equation 2.

## III. Concept of Local Restoration with Backtracking

In this section, we introduce the concept of backtracking and discuss properties of backup subgraph $G_{sub}$ in various cases of backtracking.

### A. Backtracking

In the case of end-to-end restoration routing, detection of link failures and subsequent restoration by source node can lead to long restoration latencies. Local restoration attempts to address this by "localizing" fault detection and subsequent restoration actions. Consider the simple example shown in Figure 1-(a) where there are as many backup paths $LB_i$ as links in the primary path and restoration is truly local and rapid. Under single link failure model, completely link disjoint $LB_i$s result in bandwidth wastage. Instead, a model wherein backup paths (1) share links amongst them and also (2) share links in the primary path can be used to minimize bandwidth usage at the cost of increased restoration latency. Consider the

TABLE I
VECTOR AND SCALAR LINK COSTS

| Vector $W$ of costs for link $l$ | Simplified scalar cost for link $l$ |
|---|---|
| $$w[l,i] = \begin{cases} \infty & \text{if } b - FR[l,i] > R_l \\ b * C_F & \text{if } b \le FR[l,i] \\ FR[l,i] * C_F + & \text{if } b > FR[l,i] \\ (b - FR[l,i]) * C_R & \text{and}(b - FR[l,i] < R_l) \end{cases} \quad (1)$$ | $$w_l = \begin{cases} \infty & \text{if } b - FR_l > R_l \\ b * C_F & \text{if } b \le FR_l \\ FR_l * C_F + & \text{if } b > FR_l \\ (b - FR_l) * C_R & \text{and}(b - FR_l < R_l) \end{cases} \quad (2)$$ |

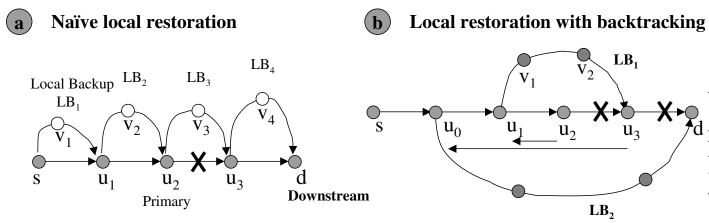



Fig. 1. Local restoration and backtracking

scenario shown in Figure 1-(b): In this case, links $(u_1, u_2)$, and $(u_2, u_3)$ are protected by a single backup path, $LB_1 = (u_1, v_1, v_2, u_3)$. Clearly, in a naíve setup, the backup path for link $(u_2, u_3)$ is $(u_2, u_1), LB_1$, which *backtracks* over primary path by a *distance* of one link namely $(u_2, u_1)$. Similarly for link $(u_3, d)$, the backup path $(u_3, u_2)$, $(u_2, u_1)$, $(u_1, u_0)$, $LB_2$, backtracks by a distance of three links. If restoration must be truly local, i.e failure of $(u_2, u_3)$ must be restored by $u_2$ and failure of $(u_3, d)$ must be restored by $u_3$, then $u_2$, $u_3$ must "switch back or backtrack" traffic on links $(u_3, u_2)$, $(u_2, u_1)$, $(u_1, u_0)$. Such repeated link traversals cause undesirable traffic loops, loss and bandwidth wastage. This is especially true when the node performing the restoration is multiple hops away from the node to which failed link belongs. Alternatively, if nodes $u_2$ and $u_3$ can inform $u_1$ out-of-band of their links failures, $u_1$ can perform the restoration. However, in this case restoration is *non-local* and requires new form of signaling or modifications to existing routing protocol state updates.

We define the *backtracking distance* $D$ as the maximum number of hops (links) in the primary path that must be traversed upstream towards the source before reaching the node at which the required backup (restoration) path originates. Three cases that are of interest are as follows:

- *No backtracking ($D = 0$)*: In this case, the backup path must originate at the node at which the failed link originates. This case represents *true* local restoration and provides best restoration latency. The link restoration paths computed in this case can use a subset of primary path links downstream towards the destination node.
- *Bounded backtracking ($D = k$)*: In this case, the backup path can originate at a node on the primary path up to $k$ hops away from the node that owns the link.
- *Infinite backtracking ($D = \infty$)*: This case allows unlimited backtracking and therefore, in the worst case may result in backup paths that originate at the source node. The end-to-end restoration can be considered a special case of this where the maximum backtracking allowed is equal to length of the primary path but the restoration always has to be initiated at the source node.

Clearly, $D = 0$ may require the highest amount backup bandwidth, and lowest restoration latency whereas $D = \infty$ requires the least amount of bandwidth and highest restoration latency. Therefore, case $D = k$ is interesting as it allows us to tradeoff bandwidth for better restoration latency. In the rest of the paper we address the problem of computing primary path $P$ and its corresponding backup restoration graph $G_{sub}$ for various cases of backtracking.

## IV. Joint Optimization of Primary and Backup Paths

In this section, we consider the problem of joint optimization of computing primary path and its associated backup subgraph $G_{sub}$ such that the total cost is minimized. For a new request $r = (s, d, b)$ and a given network $G = (V, E)$, ideally we would like to have a primary path $P = (s = u_1, u_2, \cdots d = u_k)$ and $G_{sub} = (V_{sub}, E_{sub})$ such that the total cost $\sum_{l \in E(P)} \mu_l + \sum_{l \in E_{sub}} w_l$ is minimized under the following constraints: for every edge $l = (u_i, u_{i+1}) \in E(P)$, there is a path from $u_j$ to $u_t$ in $G_{sub}$ such that $0 \le i - j \le D$ and $t \ge i + 1$, i.e. the backtracking distance of the backup path does not exceed $D$ hops and the path must end at a node that is downstream from $u_{i+1}$ in $P$.

There are two characteristics of this joint optimization problem that make it hard to solve: (1) For each edge, its cost depends on whether it is used for primary path or backup path. (2) Equation 1 shows that the cost of a backup link also depends on which set of primary links it protects. The following theorem characterizes the complexity of this optimization problem.

*Theorem 4.1:* For all values of backtracking $D$ $(0, k, \infty)$, computation of primary path $P$ and a restoration subgraph $G_{sub}$ such that they are jointly optimal is NP-hard.

For proof, please see Appendix-.

## V. Two-step Algorithms

In this section, we consider two-step algorithms that compute the primary path in the first step and then construct the restoration graph $G_{sub}$ in the second step. The basic pseudocode for these algorithms is presented in Table II.

The problem of primary path routing – computing a bandwidth guaranteed path between a pair of source and destination nodes- has received significant attention in recent years. The simplest solution to this problem is to use Dijkstra's shortest path algorithm. However, its drawback is that though it yields

TABLE II

A TWO-STEP ALGORITHM

```
1.  Given graph G and req r=(s,d,r);
2.  P=WSPF(G,r); // Primary path using WSPF
3.  If (P==NULL) return reject;
4.  Switch (D) // Backup path computation
5.    case 0:
6.      G_sub=modified_directed_Steiner(G,P,r);
7.    break;
8.    case ∞:
9.      G_sub=Suurballe(G,P,r);
10.   break;
11.   case k:
12.     G_sub=modified_Suurballe(G,P,r,k);
13. if (G_sub==NULL) return reject;
14. postprocessing(G_sub);
15. return accept;
```

an optimal solution for a single request, over a span of multiple requests, it can lead to high request rejection and low network utilization [1], [10]. Several new algorithms such as Widest Shortest Path First (WSPF) [1], Minimum Interference Routing [10], rectify this limitation. The WSPF algorithm alleviates this problem by selecting the shortest path with maximum (*"widest"*) residual capacity on its component links. We choose WSPF (line 2) as a good tradeoff between simplicity and performance.

In the remaining section, we discuss backup path computation for cases of $D = 0$ (line 6), $D = \infty$ (line 9), and $D = k$ (line 12) in detail.

### A. Computing Backup Path for $D = 0$

In this case, we need to find a subgraph $G_{sub}$ such that there is an alternative path with no backtracking if any link in the primary path fails, and the total link costs for all backup paths are minimized. The following theorem states that the problem is NP-hard.

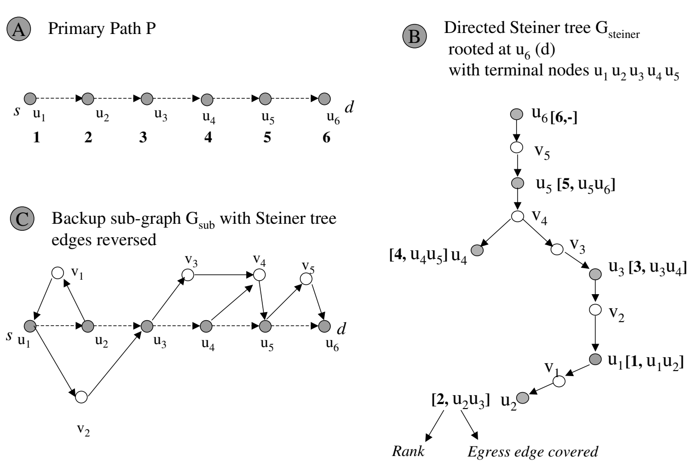



Fig. 2. Example of backup path for $D = 0$

*Theorem 5.1:* Given a network $G = (V, E)$ and a primary path $P$ from $s$ to $d$, the problem of computing optimal backup subgraph $G_{sub}$ for $D = 0$ case is NP-hard.

For proof, please refer to Appendix-. This theorem also gives us the algorithm for the computation of $G_{sub}$. Specifically, (1) Compute a directed Steiner tree $G_{Stei}$ on subgraph $G' = (V, E - E(P))$, rooted at $d$ and with the nodes $V(P) - \{d\}$ in the primary path as its terminal nodes where $V(P)$ denotes the node set used in path $P$. (2) Reverse all the edges of the tree to obtain backup subgraph $G_{sub}$. Figure 2 illustrates this with an example. It shows the primary path $P = (s = u_1, u_2, u_3, u_4, u_5, d = u_6)$, the corresponding Steiner tree $G_{stei}$ and the backup subgraph $G_{sub}$ which has links that are reverse of those in $G_{stei}$.

Among the approximation algorithms that compute directed Steiner tree such as [16], [13], [6], we choose the SCTF algorithms [13]. SCTF has good approximation ratio if the graph is not very asymmetric (asymmetry is measured as the sum of the larger cost of edges $(u, v)$ and $(v, u)$ divided by the sum of the smaller cost.). Applying SCTF algorithm with minor modifications, the backup subgraph is computed as follows: the current subgraph $G_{sub}$ is initialized to be the single node $d$; the algorithm then iteratively computes a shortest path from the current set of nodes in the primary path that are not in the current subgraph to the current subgraph; the algorithm terminates when all the primary nodes are in $G_{sub}$. Each iteration involves computing a shortest path from one set of nodes to another set of nodes. This requires only one invocation of Dijkstra's algorithm by adding dummy source and destination. The running time is $O(hnlogn + hm)$ where $h = |V(P)| - 1$, $n = |V|$ and $m = |E|$.

The reversed directed Steiner tree as a restoration subgraph $G_{sub}$ has several advantages: (1) For every link in the primary path, there are multiple restoration paths that end on the immediate neighbor or neighbors downstream. This encourages sharing of primary path links and minimizes new bandwidth reservation. In our example (Figure 2), in the case of links $(u_3, u_4)$ and $(u_4, u_5)$, the backup paths $(u_3, v_3, v_4, u_5)$, and $(u_4, v_4, u_5)$ share the primary path link $(u_5, u_6)$ for restoration. (2) Also, the explicit tree structure ensures that local restoration paths share links. In our example, restoration path for $(u_1, u_2)$ is $LB = (u_1, v_2, u_3)$ which is also part of the restoration path $(u_2, v_1, u_1, v_2, u_3)$ for $(u_2, u_3)$. In the Section VI, we describe an algorithm that optimally selects restoration paths from the $G_{sub}$.

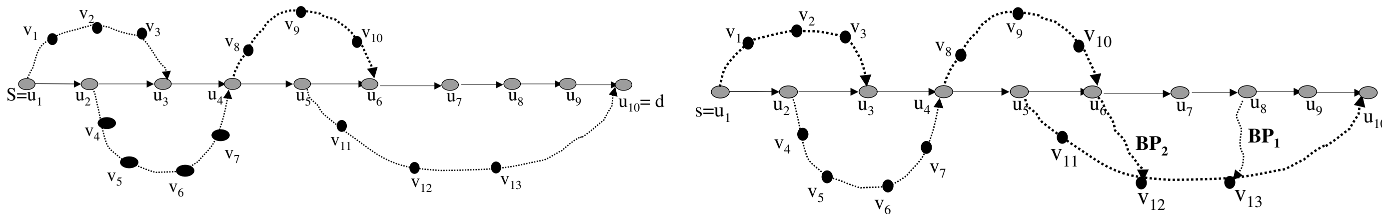
### B. Computing Backup Paths for $D = \infty$

Given a primary path $P$ for request $(s, d, b)$ in directed graph $G = (V, E)$, if there is no constraint on the backtracking distance, the following procedure due to Suurballe [15] computes a minimum cost backup set of edges for $P$. The idea is to reverse the direction of the edges on the path $P$ and set their cost to zero. All other edges are assigned cost as defined in Equation 2. Compute a shortest path $Q$ from $s$ to $d$ in the resulting graph $G'$.

Suurballe [15] shows that the edges of $Q$ that are not in $P$ represent the optimal backup path for the primary path $P$ with $D = \infty$. Figure 3-(a) illustrates an example, where primary path $P$ is $(u_1, \ldots, u_{10})$ and backup subgraph consists of paths $(u_1, v_1, v_2\ v_3, u_3)$, $(u_2, v_4, v_5, v_6, v_7, u_4, v_8, v_9, v_{10}, u_6)$, $(u_5, v_{11}, v_{12}, v_{13}, u_{10})$.

### C. Computing Backup Paths for $D = k$

For this case, we state the following theorem.




(a) Example of primary and backup path for $D = \infty$

(b) Basic ideas in $D = k$

Fig. 3. Computing restoration paths for $D = 0$ and $D = k$

*Theorem 5.2:* Given a primary path $P$ in $G = (V, E)$, computing the minimum-cost backup subgraph $G_{sub}$ based on cost $w_l$ using bounded backtracking of $D = k$ is NP-hard.

For proof, please refer to Appendix-. For the computation of backup subgraph for $D = k$ case, we build upon the procedure due to [15]. Table III illustrates the pseudo code of our heuristic algorithm. We first compute a backup path $Q$ for $P$ as in Section V-B. It is easy to see that the cost of $Q$ is a lower bound on the cost of any solution for the $D = k$ case. The edges of $Q$ allow the edges on the primary path to be restored, however the maximum distance some edge in the primary path has to backtrack might not be bounded by $k$. Procedure $maxD$ checks the maximum backtracking distance and if it exceeds $k$, additional paths are added in a greedy fashion to the edges of $Q$. We use an example to illustrate our ideas. Consider the example in Figure 3-(a) after

TABLE III

HEURISTIC ALGORITHM FOR BACKUP PATH COMPUTATION OF $D = k$ CASE

```
modified_Suurballe(G,P,r,k)
1.  Q=Suurballe(G,P);
2.  If (Q==NULL) return NULL;
3.
4.  if (maxD(P,Q)>k)
5.      Ga=compExtraPaths(G,P,Q,k);
6.      if (Ga==NULL) return NULL;
7.      else Gsub=Q∪Ga;
8.  return Gsub;
```

line 2 in Table III. We use $k = 1$ in the example. This requires that the restoration path for a link $(u_i, u_{i+1})$ in the primary path must originate at $u_i$ or $u_{i-1}$. Note that the edges of $Q$ computed from Suurballe's procedure allow all edges except $(u_7, u_8)$, $(u_8, u_9)$ and $(u_9, u_{10})$ to have a restoration path with backtracking distance at most 1. In the example, the backup path with the smallest backtracking distance for the link $(u_8, u_9)$ is the one that originates at $u_5$ which has a backtracking distance 3. If we add paths $BP_1, BP_2$ as shown in Figure 3-(b) to the $G_{sub}$, we can obtain an efficient solution that satisfies the backtracking distance constraint. Our algorithm (Line 5) finds such paths in the following way.

For every link $(u_i, u_{i+1})$ in the primary path which does not have a restoration path with backtracking distance bounded by $k$, we add a path from some node in the set $\{u_{i-k}, u_{i-k+1}, \ldots, u_i\}$ to reach a node in $\{u_{i+1}, u_{i+2}, \ldots, d\}$. This ensures that we satisfy the requirement for all links on the primary path. We process unsatisfied links in the order of their increasing distance to the destination. In the example, the link $(u_9, u_{10})$ is considered first. To satisfy this link we consider adding paths from either $u_8$ or $u_9$ to reach the node $u_{10}$. We consider $u_8$ first and if we are unable to find a path to reach $u_{10}$, we then consider $u_9$. In general, when trying to satisfy link $(u_i, u_{i+1})$, we search for paths starting with $u_{i-k}$ and stop when we find a path. In the example, we find $BP_1$ from $u_8$. Note that we find a shortest path at each step to minimize the cost of solution. Once a link is satisfied, we move to the next unsatisfied link (farther away from the destination) and repeat the above procedure. In the above example, adding path $BP_1$ satisfies both $(u_9, u_{10})$ and $(u_8, u_9)$ and the next unsatisfied link is $(u_7, u_8)$. The process stops when no unsatisfied links remain. All the $BP_i$ so obtained combined with the original $Q$ – i.e. $G_{sub} = \{BP_i\} \cup Q$ provides the restoration graph that satisfies required backtracking constraint.

In the above procedure, we need to find a shortest path from a node $u \in \{u_{i-k}, \ldots, u_i\}$ to reach any node in $\{u_{i+1}, u_{i+2}, \ldots, d\}$. This can be implemented by Dijkstra's algorithm by first shrinking the nodes $\{u_{i+1}, u_{i+2}, \ldots, d\}$ into a single super node. In computing the shortest path we reduce the cost of the edges in $G_{sub}$ computed till now to zero so as to make sure that we reuse the edges that are already in our solution as much as possible.

## VI. POST-PROCESSING

In the following we first discuss the concept of post-processing and provide algorithms for each case of backtracking.

### A. Need for Post-processing

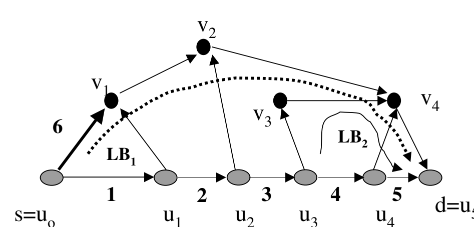



Fig. 4. Need for post-processing

Figure 4 illustrates primary path $P$ $(u_0, u_1, u_2, u_3, u_4, u_5)$, and the corresponding backup subgraph $G_{sub}$ for $D = 0$ for a request of size $b = 18$ units. Recall that $G_{sub}$ is the reverse of the Steiner tree rooted at $u_5$. Here link 1 $(u_0, u_1)$ is backed up using path $LB_1$, constituted by $(u_0, v_1)$, $(v_1, v_2)$, $(v_2, v_4)$ $(v_4, u_5)$, whereas link 4 $(u_3, u_4)$ is backed up by using path $LB_2$ constituted by $(u_3, v_3)$, $(v_3, v_4)$, $(v_4, u_5)$. Consider link 6 $(u_0, v_1)$ on $LB_1$, with residual bandwidth $R_6 = 22$, backup load $A_6 = 32$. The backup load induced by link 4 on link 6 is $BLDM[6, 4] = 20$, whereas load induced by link 1 on link 6 is $BLDM[6, 1] = 10$.

Unfortunately, when link cost $w_l$ is assigned for Steiner tree computation, a pessimistic assumption has to be made: the worst case backup load on link 6 is 20 units induced by link 4. This has to be done to account for possibility that backup path for link 4 may include link 6. So the extra bandwidth reservation on link 6 is $b - (A_6 - BLDM[6, 4])$ or (18 - (32 -20))=6 units. In other words, the Equation 2 we employ to compute $FR_l$ in Steiner tree computation is rather conservative as it assumes that any candidate link in backup graph may have to backup worst case load from a link in the primary path. If we introduce post-processing that follows computation of Steiner tree and path selection, we can fix this situation. Specifically, in our example, if we account for the fact that only backup path $LB_1$ for link 1 uses link 6, we can see that the amount of free bandwidth on link 6 is $(A_6 - BLDM[6, 1]) = 32 - 10 = 22$, which is greater than the request size. Clearly, in that case no extra bandwidth has to be reserved on link 6. Such post processing of the solution can reduce extra bandwidth reservation for backup paths and therefore save bandwidth.

We provide post-processing algorithms for each case of $D$. The two main goals of these algorithms are: (1) Identify for each primary path link, a set of potential restoration paths with minimum backtracking distance. (2) Minimize the total bandwidth reserved on the backup paths subject to assignment of restoration paths. We state the following theorem for post-processing for all three cases of backtracking:

*Theorem 6.1:* Given a primary path $P$, the backup subgraph $G_{sub}$ that our algorithms compute is a forest consisting a set of trees $\{T_i\}$. The amount of bandwidth needed to be reserved can be computed optimally by a single traversal of $P$ and one ordered traversal of each tree $T_i$. The backup paths for each primary edge can be found in time linear in the output size.

We do not give a formal proof of the theorem. However the description of the algorithms in the following subsections contains all the ideas behind the proof.

### B. Post-processing for $D = 0$

For this case we describe a post-processing algorithm that results in the minimum use of bandwidth subject to using the edges of $G_{sub}$ for restoration. We use the example from Figure 2 to illustrate the algorithm. The algorithm is based on the fact that the backup edges form a Steiner tree $G_{stei}$ on the vertices of the primary path $P$. For a link $(u_i, u_{i+1})$ on the path $P$, the restoration path has to originate at $u_i$. In $G_{stei}$, for every $u_i$ in $P$, there is a *unique* directed path $LB_i$ from $u_i$ to the destination $d$. We could use this path $LB_i$ as a restoration path for $(u_i, u_{i+1})$, however we notice that $LB_i$ can intersect the path $P$ at several places and in particular there could be a vertex $u_j$, $j \geq i + 1$ that lies on $LB_i$. In this case, it is easy to see that the portion of $LB_i$ from $u_i$ to $u_j$ is sufficient to restore the link $(u_i, u_{i+1})$. In Figure 2, the path $LB_3$ is $(u_3, v_3, v_4, u_5, v_5, u_6)$. However the portion of $LB_3$ that is $(u_3, v_3, v_4, u_5)$ is sufficient to restore the link $(u_3, u_4)$. More generally, for the unique path $LB_i$ from $u_i$ to $d$, let $u_j$ be the first vertex in $LB_i$ such that $j \geq i + 1$. Let $Q'_i$ be the portion of $LB_i$ from $u_i$ to $u_j$. We use $Q'_i$ to restore the link $(u_i, u_{i+1})$. In Figure 2, the path $LB_2$ intersects $P$ at $u_1, u_3, u_5$ and finally reaches $d = u_6$. We use the portion of $LB_2$ from $u_1$ to $u_3$, hence $Q'_2$ is $(u_2, v_1, u_1, v_2, u_3)$. It is easily seen that this $Q'_i$ as constructed above is necessary *and* sufficient for the restoration of $(u_i, u_{i+1})$.

Now we address the issue of minimizing the bandwidth. For each link $\ell$ in $G_{stei}$, let $S_\ell$ be the set of all $i$ such that the link $\ell$ is in the restoration path $Q'_i$ for link $(u_i, u_{i+1})$. In our example from Figure 2, the link $(v_2, u_3)$ is used by $Q'_1$ and $Q'_2$, hence $S_{(v_2,u_3)} = \{1, 2\}$. For link $(v_4, u_5)$, we have $S_{(v_4,u_5)} = \{3, 4\}$. In other words, $S_\ell$ is the precise set of all primary path links that use $\ell$ in their backup paths. Therefore it is sufficient to reserve bandwidth on $\ell$ that will satisfy the links in $S_\ell$ and not all the links of $P$. This results in savings in bandwidth. More precisely, the free bandwidth on $\ell$ for backing up links in $S_\ell$ is $FR_\ell = \min_{i \in S_\ell} FR[\ell, i]$. This is to be contrasted with the pessimistic estimation $FR_\ell = \min_{i \in E(P)} FR[\ell, i]$ used in the computation of $G_{stei}$. Since $S_\ell$ is a proper subset of $E(P)$, we can potentially use more free bandwidth on $\ell$ than estimated.

The revised bounds on the free bandwidth can be easily computed once the restoration paths $Q'_i$ are computed for all $i$. However, we note that both the $Q'_i$ for all links of $P$ and the $FR_\ell$ for all the links in $G_{stei}$ can be computed in a single DFS traversal of $G_{stei}$ by keeping some simple state information at the vertices of the trees. The traversal of $G_{stei}$ takes $O(m)$ time where $m$ is the number of vertices (edges) in $G_{stei}$. The explicit enumeration of the restoration paths needs an additional $O(h)$ time, where $h$ is the sum total of the lengths of $Q'_i$. Hence, the running time is linear in the size of the tree and the size of the output.

### C. Post-processing for $D = \infty, k$

The post-processing algorithms for $D = \infty$ and $D = k$ are conceptually similar to the case $D = 0$ and hence we omit the details. However we do need to take care of two issues. First, for a given link $(u_i, u_{i+1})$ in $P$ there might be several restoration paths that satisfy the backtracking constraint. By choosing the restoration path with the minimum backtracking distance we can make the restoration path for each link unique. Second, instead of a single tree as in the $D = 0$ case we might have a forest consisting of many trees. We process each of these trees much as we do in the $D = 0$ case.

## VII. Simulation Experiments

In this section, we describe simulation experiments that characterize the benefits of our proposed local link restoration schemes over existing end-to-end path restoration schemes. We conduct a set of experiments that compare our three schemes $(D = 0, k, \infty)$ with two end-to-end restoration schemes, namely Enhanced Widest-Shortest-Path First(EWSPF) [12], and simple Shortest Path First (SPF). EWSPF exploits bandwidth sharing among backup paths and has been shown to perform better than other schemes in the literature [12]. The SPF scheme uses link costs based on the residual capacity and computes two independent paths: one used as primary and the other as backup and do not attempt to share backup paths. It is used as a base line to show how much benefit bandwidth sharing schemes can provide. Because our topology is relatively small, we only consider $D = 1$ case for the limited backtracking algorithm.

### A. Simulation Setup

This section describes the network topology, traffic parameters and performance metrics used in the simulation.

TABLE IV
SIMULATION PARAMETERS

| Parameter | Value |
|---|---|
| Request (REQ) arrival | Poisson at every router |
| Call holding time (HT) | 200 time units, exponentially distributed |
| Simulation time (STT) | Fixed 50,000 units |
| Request Volume (RV) during entire STT | 50,000 to 300,000 |
| Max single LSP REQ size | 5% of the link capacity |
| Mean REQ inter-arrival time | Computed using RV and STT |
| Destination node selection | Randomly distributed |

Table IV shows the parameters used in the experiments. Call holding time is how long each request lasts. Note that in reality, request load at various nodes may not be random and the amount of requests between certain node pairs may be dis-proportional. However, no real life call traffic data sets are currently available in public domain, therefore we use Poisson distribution to model request arrival process. We use 6 quantization levels for the EWSPF algorithm which are $0.05, 0.1, 0.3, 0.5, 0.7$ and $1.0$ times the maximum requested bandwidth [12]. Figure 5 illustrates the network topologies used in our experiments. The network topology is the same as the topology in [12]. The topology represents the Delaunay triangulation for the 20 largest metro areas in continental U.S. In the homogeneous topology, all links are of the same capacity (OC-48). In the heterogeneous topology case, the network core consists of fast OC-48 links (thick links) that connect access networks with slower OC-12 links(thin links).

### B. Performance Metrics

We defined performance metrics to characterize (1) restoration latency and (2) bandwidth sharing performance [12].

*1) Restoration Latency Performance:* A brute force direct simulation of faults and subsequent measurement of restoration latency requires a very complex model that simulates multi-gigabit data traffic, link faults and corresponding fault propagation traffic in a complete network. Such a simulation that involves simulation of traffic instead of just request arrivals and corresponding route computations yields at best only representative performance numbers. So instead of taking this inordinately complex approach, we use an indirect way to measure restoration latency based on the following two performance metrics: (1) *Histogram of Backtracking Distance (HBD)*: Given a primary path of length $l$, the restoration model used dictates the amount of worst case backtracking for a link in the primary path. For example, with local restoration and backtracking distance of $D = 2$, the backup path may backtrack $0, 1$ or $2$ links. On the other hand for end-to-end restoration, for the $1^{st}$ link in the path, backtracking is equal to zero, whereas for the $l^{th}$ link in the path, restoration backtracks to source node and has backtracking distance of $l - 1$. The more the backtracking, the more is the restoration latency. Therefore, we compute a histogram, where bin $i$ corresponds to total number of links in the admitted primary paths for which backup path backtracks $i$ links. Clearly, this histogram characterizes the probability that a given amount of backtracking will occur. (2) *Average Case Backtracking Distance (ABD):* We define this metric as follows:

$$ABD = \frac{\sum_{i=1}^{n_a} \sum_{j=1}^{|P_i|} D_{ij}}{\sum_{i=1}^{n_a} \mid P_i \mid} \tag{4}$$

where $n_a$ is total number of accepted requests, $P_i$ is the primary path routed for request $i$, $|P_i|$ represents the length of the path and $D_{ij}$ is the actual backtracking distance to backup the $j$-th link of the primary path $P_i$. $ABD$ measures the average backtracking distance among all the backup paths. Clearly, it captures expected case of backtracking and therefore, restoration latency.

*2) Bandwidth Sharing Performance:* We used the following performance metrics in our evaluation [12]: (1) *Fraction Rejected (FR):* is the fraction of requests that are dropped during each simulation run. (2) *Total bandwidth consumed:* is the total bandwidth consumed by each scheme including primary and backup reservations.

### C. Simulation Results

In the following, we discuss variation of these performance metrics for different cases of $D$ and request volumes.

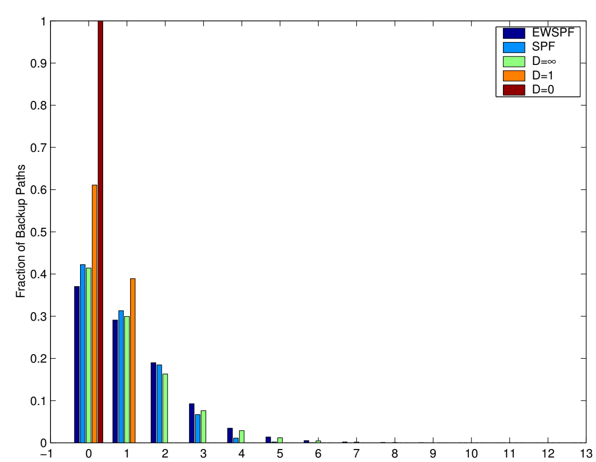
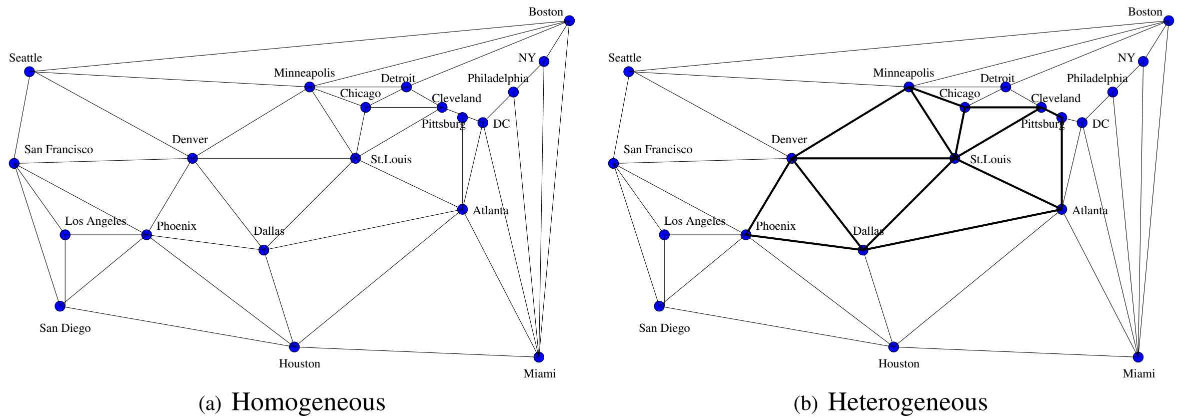



Fig. 6. Histogram of backtracking distance (HBD)




(a) Homogeneous (b) Heterogeneous

Fig. 5. Network topologies

*1) Restoration Latency Performance:* Figure 6 shows the histogram of backtracking distance for all the schemes using the request volume 300,000 as a representative case. For schemes EWSPF, SPF and $D = \infty$, backtracking distance is not bounded and can range from 0 to 13. Approximately 17% and 8% of backup paths require backtracking distance 2 and 3 respectively. For schemes $D = 0$ and $D = 1$ that bound the backtracking distance, the backtracking distance stays within the preset limits.

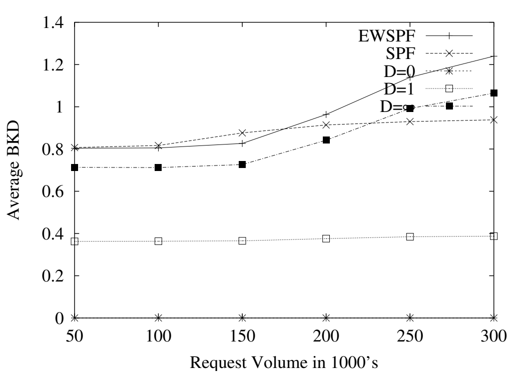



Fig. 7. Average backtracking distance (ABD)

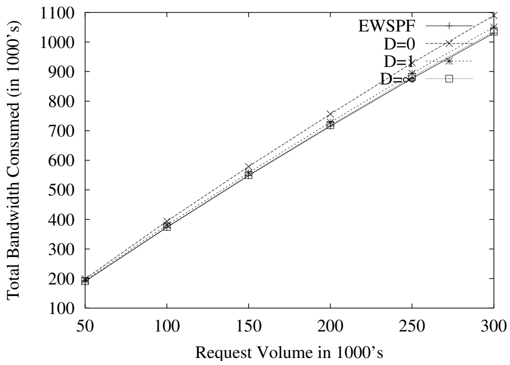
Figure 7 shows the average backtracking distance ABD for all the schemes. We see that the two metrics have similar behavior. Note that the ABD for EWSPF and $D = \infty$ increases more rapidly than that for SPF. This happens because SPF has high rate of request rejection and utilizes network links rapidly. So it begins with shorter backup paths but as the links fill up, it can't find longer feasible paths because a large subset of network links become infeasible due to lack of residual bandwidth. So with increased request rejection, the path length and therefore ABD metric remains flat. On the contrary, EWSPF exploits more bandwidth sharing and can route increasingly longer paths as load increases which results in higher ABD. Note that the case of $D = \infty$ results in a similar behavior. This clearly suggests that as the load on the network increases, higher ABD will result in higher restoration latency and also, poorer control on providing tighter bounds on it. On the contrary, with our algorithms for $D = 0, 1$, the ABD stays within the pre-set limits. Note that the ABD for EWSPF is about 12% longer than that for $D = \infty$ for all request size. The reason is that our algorithm for $D = \infty$ tries to reduce the backtracking distance even if it does not place a bound.

*2) Bandwidth Sharing Performance:* Before we look into networks with finite capacity, we look at the bandwidth consumed if the network has infinite capacity on every link. This is interesting to look at because we can understand the amount of bandwidth reservation required for each scheme without all the complications associated with networks with bounded capacity, e.g. each scheme accept different request set.

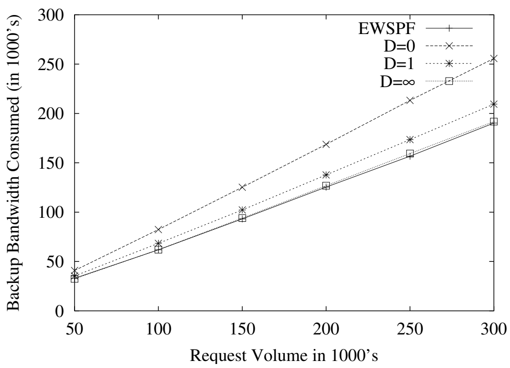
From figure 8, we see that only a small fraction of the total bandwidth is used for backup paths, e.g. for request volume 300,000, figure 8 shows the total bandwidth is roughly 1100 units and the total backup bandwidth is only 250 units which is about 22%. From figure 8, only about 22% of the total reserved bandwidth are used for backup paths. Therefore, bandwidth sharing is very significant. From figure 8-b, we also see that $D = 0$ reserves 34% more bandwidth that EWSPF and $D = \infty$. $D = 1$ reserves 10% more bandwidth that EWSPF and $D = \infty$.

**Homogenous Network Topology**

*Fraction Rejected (FR)*: As expected, FR increases as the load or RV increases. All the sharing schemes are significantly better than SPF and can accept up to 22% more requests. This suggests backup bandwidth sharing enables much more efficient use of network resources than no-sharing case. There are two factors that impact FR. (1) Since end-to-end restoration schemes use one path to restore all the link failures in the primary path, fewer links need extra bandwidth reservation than $D < \infty$. (2) Local restoration schemes can share backup bandwidth more aggressively than end-to-end schemes. This is because: in the end-to-end scheme, each link in the backup path has to assume that the maximum load in the primary link can be backed up on that link (see Equation 2). On the other hand, in the local restoration schemes with post processing, each link only needs to backup the set of links whose backup path uses that link, that is, more bandwidth can be shared.

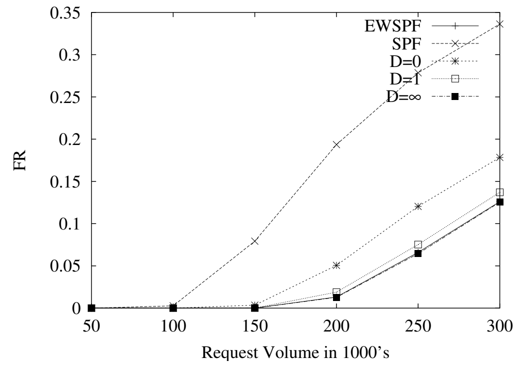



(a) Total bandwidth consumed

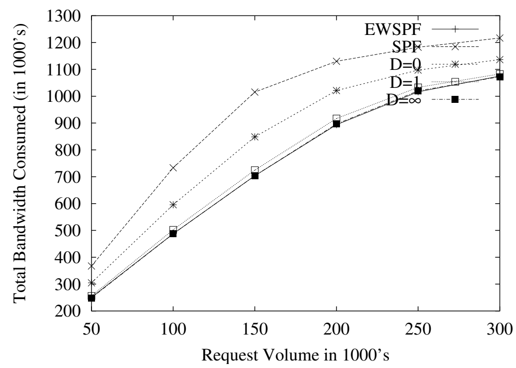



(b) Total backup bandwidth consumed

Fig. 8. Network with infinite capacity on each link




(a) Fraction rejected (FR)




(b) Total bandwidth consumed

Fig. 9. Bandwidth Sharing Performance: Homogenous Topology

Although EWSPF and $D = \infty$ are analogous in terms of unlimited backtracking, unlike the EWSPF case, backup paths for $D = \infty$ case can share primary path links. Therefore, FR for EWSPF should be higher than that of scheme $D = \infty$, (See Section VI). However, since the networks used in our simulation is relatively small, there are not many cases where scheme $D = \infty$ can share primary path links. Therefore, the FR for EWSPF and $D = \infty$ is indistinguishable. With respect to $D = 0$ and $D = 1$, the first factor is more effective as many more links are needed for backup bandwidth reservation, this results in our algorithm for $D = 0$ and $D = 1$ rejects up to 4% and 2% more requests respectively than EWSPF.

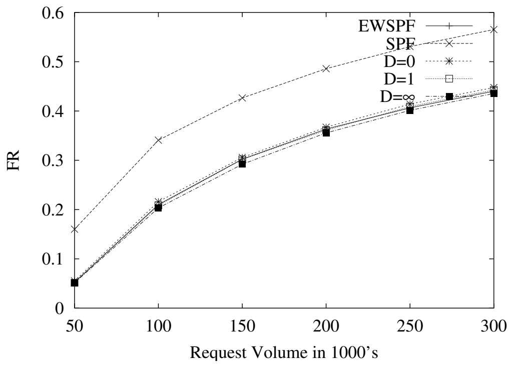
*Total Bandwidth Consumed*: Figure 9-b shows the total bandwidth consumed for all the schemes. We see that even if SPF rejects up to 22% more request, it consumes much more bandwidth than the other schemes. For example, SPF consumes 28% more bandwidth than $D = 0$ when the request volume is 150,000. Scheme $D = 0$ consumes 17% more bandwidth than EWSPF. Scheme $D = 1$ only consumes 3% more than EWSPF. This suggests, scheme $D = 1$ is a good tradeoff between restoration latency and network resource consumption.

**Heterogeneous Network Topology**

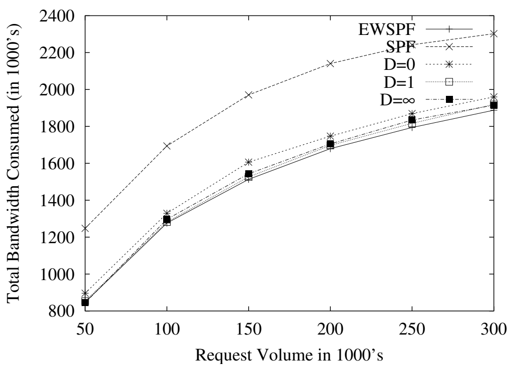
Figure 10-a shows the FR vs the volume for an LF value of 5% of the OC-48 links. Since the access links are at OC-12, an LF of 5% of OC-48 will allow for requests of 20% of the access link. This results in the access links to get saturated very quickly and lead to higher rejection ratio for all schemes. The rejection ratios of EWSPF, schemes $D = 0$, $D = 1$ and $D = \infty$ are indistinguishable. The amount of bandwidth they consume is similar to one another. The total bandwidth consumed for scheme $D = 0$ is marginally higher $\approx 4\%$ than that for EWSPF. We also performed experiments with an LF of 2% (omitted due to space constraints) which would be at most 8% of the OC-12 links. For such a case, the FR of scheme $D = 0$ is only marginally higher–less than 2% than that of scheme $D = 1$, $D = \infty$ and EWSPF. The bandwidth used by scheme $D = 0$ is still marginally higher–less than 7% than that used by EWSPF. This suggests that, in the heterogeneous topology case considered, scheme $D = 0$ is a good algorithm to choose for best restoration latency with only marginally sacrifice on performance and resource consumption.

## VIII. Conclusions

In this paper, we proposed a framework for provisioning bandwidth guaranteed paths with bounded restoration latency.




(a) fraction rejected (FR)




(b) total bandwidth consumed

Fig. 10. Bandwidth Sharing Performance: Heterogeneous Topology

We introduced the novel concept of backtracking distance $D$ and considered three different cases: (1) no backtracking ($D = 0$), (2) limited backtracking ($D = k$), and (3) unlimited backtracking ($D = \infty$). We used a link cost model that captures bandwidth sharing among links using various types of available link state information. We first showed that joint optimization of primary and backup paths is NP-hard in all cases. We then considered two-step algorithms where primary path and backup paths are computed in two separate steps. Using our link cost model, we devised heuristic algorithms for each case.

Out simulation study to characterize the sharing and restoration latency performance of our schemes shows that $D = 1$ provides best tradeoff between bandwidth usage and restoration latency. Since the fault information only needs to be propagated at most one hop in this case, restoration latency performance may be acceptable for most service needs.

## References

[1] G. Apostolopoulos, R. Guerin, and S.Kamat. QoS routing mechanisms and OSPF extensions LDP. work in progress, IETF draft, December 1998.

[2] D. et. al. Awduche. Requirements of Traffic Engineering over MPLS. RFC 2702, IETF, September 1999.

[3] B. Davie and Y. Rekhter. *MPLS Technology and Applications*. Morgan Kaufmann, San Francisco, CA, 2000.

[4] Loa Andersson (editor). PPVPN L2 Framework. Work in progress - Internet Draft, IETF, March 2002. draft-aboba-pppext-key-problem-01.txt.

[5] Jamoussi et al. Constraint-based LSP setup using LDP. RFC 3212, IETF, January 2002.

[6] M. Charikar et. al. Approximation algorithms for directed Steiner problems. In *Proceedings of the Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 192–200, January 1998.

[7] P. Pan et al. Fast Reroute Extensions to RSVP-TE for LSP Tunnels. *Internet Draft, draft-ietf-mpls-rsvp-lsp-fastreroute-00.txt*, pages 1–31, July 2002.

[8] V. Sharma et al. Framework for MPLS-based Recovery. Work in progress - Internet Draft, IETF, January 2002. draft-ietf-mpls-recovery-frmwrk-03.txt.

[9] M. Kodialam and T. V. Lakshman. Dynamic routing of bandwidth guaranteed tunnels with restoration. In *Proceedings of IEEE INFOCOM*, pages 902–911, 2000.

[10] M. Kodialam and T. V. Lakshman. Minimum interference routing with applications to MPLS traffic engineering. In *Proceedings of IEEE INFOCOM*, pages 376–385, 2000.

[11] M. Kodialam and T.V. Lakshman. Dynamic routing of locally restorable bandwidth guaranteed tunnels using aggregated link usage information. In *Proceedings of IEEE INFOCOM*, pages 884–893, 2001.

[12] S. Norden, M.M. Buddhikot M. Waldvogel, and S. Suri. Routing bandwidth guaranteed paths with restoration in label switched networks. In *Proceedings of IEEE ICNP*, pages 71–79, 2001.

[13] S. Ramanathan. Multicast tree generation in networks with asymmetric links. *IEEE/ACM Transactions on Networking*, 4(4):558–568, 1996.

[14] E. Rosen and Y. Rekhter. BGP/MPLS VPNs. RFC 2547, IETF, March 1999.

[15] J. W. Suurballe. Disjoint paths in a network. *Networks*, 4:125–145, 1974.

[16] H. Takahashi and A. Matsuyama. An approximate solution for the steiner problem in graphs. *Math. Japonica*, 24(6):573–577, 1980.

[17] D. Zhou and S. Subramaniam. Survivability in optical networks. *IEEE Network*, 14(6):16–23, December 2000.

## Appendix

*Theorem 1.1:* Joint optimization of primary path and backup subgraph is NP-hard for all three cases of backtracking distances:$D = 0, k, \infty$.

**Proof:** We can show that our joint optimization problem is at least as hard as the Set cover problem and therefore, is an NP-hard problem. The hardness factor depends on how we measure the objective functions. This happens because of the dependence of the backup cost on an edge by edge basis.

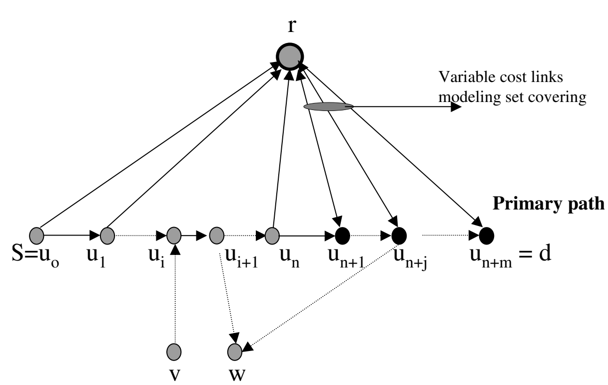



Fig. 11. Reduction from set cover

The sketch of the reduction is as follows: Let us consider a set cover instance which has $m$ sets $S_1, \cdots, S_m$ each a subset of a universe of $n$ elements $1, 2, \ldots n$. The goal is to find a smallest collection of sets that together cover (contain) all elements. This problem is known to be $\log n$-hard and can be approximated to within $\log n$ factor via the greedy algorithm.

We will reduce this problem to our problem using the construction shown in Figure 11. Let vertices in a path $P$ be labeled $u_1$ to $u_{n+m}$, where the first $n$ vertices as elements of the universe and the next $m$ vertices as the sets. Add an extra vertex $r$ and connect all the $n+m$ vertices to it in a star like fashion. Also, connect the vertex near $r$ to all the nodes $u_i$ below.

Let the request be $r = (s, d, b)$. Let the residue bandwidth of edges in $P$ be greater than $b$ and the residue bandwidth of all the edges in the star be less than $b$. Therefore, the primary path of request $r$ will be $P$ since it is the only one that guarantees its bandwidth requirement.

We model the covering aspect as follows. For a vertex $u_{n+j}$ such that $j > 0$ the edge $(r, u_{n+j})$ will be the edge representing the set $S_j$. This edge has a cost of 1 for backing up the edges $(u_i, u_{i+1})$ if $u_i$ is in set $S_j$, otherwise the cost will be a large quantity.

$$cost(r,u_{n+j}) = \begin{cases} 1 & j > n, \text{Backing } (u_i, u_{i+1}) \\ & \text{if } u_i \in S_j \\ BigB & j > n, \text{Backing } (u_i, u_{i+1}) \\ & \text{if } u_i \notin S_j \end{cases} \quad (5)$$

Hence, to back up an edge $(u_i, u_{i+1}), i \leq n$ we need to use the path $(u_i, r)$ followed by $(r, u_{n+j})$ where $S_j$ contains $u_i$. We can make the cost of backing up edges $(u_{n+j}, u_{n+j+1})$, $j > 0$ cheap by making the cost of covering them by other edges 0. Note that it is easy to see that these costs are realizable in our problem. To make $cost(r, u_{n+j})$ small for backing up $(u_i, u_{i+1})$, we can add dummy nodes $v$ and $w$, add directed edges $(v, u_i)$, $(u_{i+1})$ and $(u_{n+j}, w)$ and add request $r'$ from $v$ to $w$. Since there are only two paths $P_1 = (v, u_i, u_{i+1}, w)$ and $P_2 = (v, r, u_{n+j}, w)$. $P_1$ will be used as the primary and $P_2$ as the backup. Because the primary edge $(u_i, u_{i+1})$ uses edge $(r, u_{n+j})$ for backup request $r'$, bandwidth reserved on $(r, u_{n+j})$ by $r'$ can be used by $r$ for free. Therefore, $cost(r, u_{n+j})$ is small to backup $(u_i, u_{i+1})$. Now, the cost of solution will consist two parts: (1) the cost $C_1$ consists of the cost of the primary path $P$ and the cost of backup edges $(u_i, r)$, $i \leq n$. (2) the cost $C_2$ is the number of edges of the form $(r, u_{n+j})$ that are actually used to backup edge $(u_i, u_{i+1}), i \leq n$. The collection of sets of $S_j$ represented by edges $(r, u_{n+j})$ in the backup subgraph corresponds to a set cover and hence the cost $C_2$ will measure the set cover cost. ■

*Theorem 1.2:* Given a network $G = (V, E)$ and a primary path $P$ between source and destination pair $(s, d)$, the computation of optimal backup subgraph $G_{sub}$ for $D = 0$ case based on cost $w_l$ as defined in Equation 2 is NP-hard.

**Proof:** Given the known NP-hard directed Steiner tree optimization problem, we can reduce it to our problem easily as follows. In the Steiner tree problem, we want to connect a root $d$ to a subset of nodes $M$ of $G = (V, E)$ with minimal cost (nodes in $M$ are called terminal nodes). Let $G' = (V, E')$ where $l' = (u, v) \in E'$ iff $l = (v, u) \in E$ and the cost $w'_l = w_l$, i.e. $G'$ is the graph $G$ with every edge reversed. Order the nodes in $M$ arbitrarily as $u_1$,$u_2$,$\cdots$,$u_h$ where $h = |M|$. Connect these nodes by a directed simple path $P$ from $u_1$ to $u_h$. Let $G'' = (V, E' \cup E(P))$ (Note that $G''$ can be a directed multigraph. It is easy to take care of the case that the algorithm for computing optimal backup paths requires a graph to be a simple digraph. What we need to do is, for each $(u_i, u_{i+1}) \in E'$, add a pseudo node $v_i$ to $M$, add directed edges $(u_i, v_i)$, $(v_i, u_{i+1})$ and set their cost to zero. Let the resulting graph be $G''$. Details are omitted.). Let the optimal backup subgraph be $G_{sub}$. Let $G'_{sub}$ be the graph $G_{sub}$ with every edge reversed. We claim that $G'_{sub}$ corresponds to the optimal Steiner tree on $G = (V, E)$ with root $r$ and terminal node set $M$.

Because we do not allow backtracking, it is easy to see that there is a simple path $q_k$ from $u_{k-1}$ to $u_k$ in $G_{sub}$ where $q_k$ does not use any edges in $E(P)$. Similarly, there is a path $q_{k-1}$ from $u_{k-2}$ to $u_{k-1}$ or $u_k$ in $G_{sub}$ where $q_{k-1}$ does not use any edges in $E(P)$. By a simple induction on the length of the postfix of $P$, the subgraph $G_{sub}$ connects all the nodes in $P$ where $G_{sub}$ does not use any edges in $E(P)$. If we reverse the direction of edges in $G_{sub}$, then there is a path from $d$ to any node in $V(P) - \{d\}$, i.e. $G'_{sub}$ is a directed Steiner tree in graph $G = (V, E)$. Because $\sum_{l \in E_{sub}} w_l$ is minimum, it is easy to see $G'_{sub}$ is a minimum directed Steiner tree in $G$, which is known to be NP-hard. ■

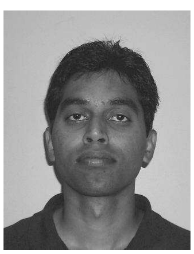
*Theorem 1.3:* Given a network $G = (V, E)$ and a primary path $P$ between source and destination pair $(s, d)$, the computation of optimal backup subgraph $G_{sub}$ for $D = k$ case based on cost $w_l$ is NP-hard.

**Proof:** Our reduction is similar to $D = 0$ case. We need to take care of the specifics of $D = k$ case. $G'$ is defined the same as in the proof for $D = 0$. Order the nodes in $M$ as $u_1$,$u_2$,$\cdots$,$u_h$ where $h = |M|$. For each node $u_i \in M$, add $D = k$ nodes ${u_i}^1$, ${u_i}^2$, $\cdots$, ${u_i}^k$ and add edge set $\{({u_i}^j, {u_i}^{j+1}) \mid j = 1, 2, \cdots, k-1\} \cup \{(u_i, {u_i}^1)\}$ where each edge has zero cost for backup computation. In addition, add edges set $\{({u_i}^k, u_{i+1}) \mid i = 1, 2, \cdots, h-1\}$.

Let the resulting graph be $G'' = (V', E'')$ and denote $M \cup \{{u_i}^j \mid u_i \in M, 1 \leq j \leq k\}$ by $M'$. Let $P$ be the simple path from $s = u_1$ to $d = u_h$ which uses only edges in $E' - E$. Lets consider node ${u_i}^k$ in $P$. If link $({u_i}^k, u_{i+1})$ fails, any path from ${u_i}^k$ to $d$ must have path $({u_i}^k, {u_i}^{k-1}, \cdots, u_i)$ as a prefix, that is to backtrack $k$ steps. Therefore, there must be a path between $u_i$ and $d$ without using any edge in $P$ in order for ${u_i}^k$ not to backtrack more than $k$ steps. By similar induction as in the proof of Theorem 5.1, $G'_{sub}$ is a directed Steiner tree in $G = (V, E)$ which connects $r$ to each node in $M$ with minimum total cost. ■

**Li (Erran) Li** received a B.E. in Automatic Control from Beijing Polytechnic University in 1993, a M.E. in Pattern Recognition from the Institute of Automation, Chinese Academy of Sciences, in 1996, and a Ph.D. in Computer Science from Cornell University in 2001 where Joseph Y. Halpern was his advisor. During his graduate study at Cornell University, he worked at Microsoft Research and Bell Labs Lucent as an intern, and at AT&T Research Center at ICSI Bekerley as a visiting student. He is presently a member of the Networking Research Center in Bell Labs. His research interests are in networking with a focus on wireless networking and mobile computing.

**Milind M. Buddhikot** is a Member of Technical Staff in the Center for Networking Research at Lucent Bell Labs. His current research interests are in the areas of systems and protocols for public wireless networks, MPLS path routing, and multimedia messaging and stream caching. Milind holds a Doctor of Science (D.Sc.) in computer science (July 1998) from Washington University in St. Louis, and a Master of Technology (M.Tech.) in communication engineering (December 1988) from the Indian Institute of Technology (I.I.T.), Bombay. He has authored over 26 research papers and 9 patent submissions in design of multimedia systems and protocols, layer-4 packet classification, MPLS path routing and integrated public wireless networks. He served as a co-guest-editor of IEEE Network magazine's March 2001 Special issue on "Fast IP Packet Forwarding and Classification for Next Generation Internet Services". Currently, Milind serves as an Editor for the IEEE/ACM Transactions on Networking.



**Chandra Chekuri** obtained his PhD from the Computer Science Department of Stanford University in 1998. After his PhD he joined the Computer Science Research Center of Bell Labs, Lucent Technologies. His main research interests are in combinatorial optimization, discrete algorithms, and their applications.

**Katherine Guo** is a member of technical staff in Center for Networking Research at Bell Laboratories in Holmdel, New Jersey. She has extensive research and product experience in multimedia streaming, Internet content distribution and multicasting. Formerly she was the architect for Lucent's Imminet streaming caching and content distribution product line. She has published more than 20 research papers in renowned technical journals and conferences, has served on the technical committees of a number of international conferences, and holds two U.S. and international patents. Her research interests include wireless protocol and system design, multimedia content distribution, and real-time distributed gaming. She holds a B.S. degree in computer science and a B.A. degree in mathematics from the University of Texas at Austin and both M.S and Ph.D. degrees in computer science from Cornell University in Ithaca, New York. Dr. Guo is a member of the IEEE.