Routing Bandwidth-Guaranteed Paths with Restoration in Label-Switched Networks ⋆

Samphel Norden [1]

Center for Networking Research, RM 4F-529, Lucent Bell Labs, Holmdel, NJ 07733, USA

Milind M. Buddhikot *

Center for Networking Research, RM 4G-508, Lucent Bell Labs, Holmdel, NJ 07733, USA

Marcel Waldvogel [2]

IBM Research, Zurich Research Laboratory, 8803 Rüschlikon, Switzerland

Subhash Suri [3]

Engineering I, Room 2111, Computer Science Department, University of California, Santa Barbara, CA 93106, USA

Abstract

A Network Service Provider (NSP) operating a label-switched networks such as ATM or Multi-Protocol Label Switching (MPLS) networks, sets up end-to-end bandwidth-guaranteed Label-Switched Paths (LSPs) to satisfy the connectivity requirements of its client networks. To make such a service highly available, the NSP may set up one or more backup LSPs for every active LSP. The backup LSPs are activated when the corresponding active LSP fails. Accordingly, the problem of LSP routing with and without restoration backup has received some attention in the recent past.

In this paper, we investigate distributed algorithms for routing of end-to-end LSPs with backup restoration in the context of label-switched networks. Specifically, we propose a new concept of the Backup Load Distribution (BLD) matrix that captures partial network state and eliminates the problems of bandwidth wastage, pessimistic link selection, and bandwidth release ambiguity. We describe two new, distributed routing algorithms that utilize the BLD matrix and require a bounded amount of run time. We can realize these algorithms in the current Internet architecture using the OSPF extensions for Quality-of-Service (QoS) routing [1, 2] to exchange the proposed BLD matrix among peer routers/switches. Our simulation results for realistic sample topologies show an excellent (30–50%) improvement in terms of rejected requests and 30–40% savings in the total bandwidth used for backup connections. We also show that, although the performance of our routing scheme is sensitive to the frequency of BLD matrix updates, the performance degradation resulting due to stale state information is insignificant for typical update periods.

To appear in Computer Networks

1 Introduction

The concept of label switching encompasses optical networking technologies, such as wavelength switching, and electronic packet-switching technologies, such as ATM and Multi-Protocol Label Switching (MPLS). A Network Service Provider (NSP) that operates a Label-Switched Network (LSN) sets up end-to-end Label-Switched Paths (LSPs) to satisfy the connectivity requirements of its client networks. For these LSPs, the NSP may guarantee certain Quality-of-Service (QoS) attributes such as fixed bandwidth, delay, or delay jitter. Formally, an LSP request can be characterized by a tuple $\langle s, d, q_1, q_2, \ldots, q_n \rangle$, where s, d are the source and destination address of the client networks, and $q_1, q_2, \ldots q_n$ are the QoS requirements of the LSP. In practice only one QoS metric, namely the bandwidth guarantee, has been used. In this case the LSP request can be represented by a 3-tuple $\langle s, d, b \rangle$, where b is the LSP bandwidth. Each such LSP can be described by a set of labels, $l_1, l_2, \ldots, l_n$, one per switching hop. Figure 1(a) illustrates this for an MPLS packet-switched network. Here, the labels (B, C, D) describe the LSP along path (L_7, L_9, L_{10}) set up to satisfy request $\langle R_1, R_5, b \rangle$.

In MPLS networks, an LSP between s and d is a simplex flow, i.e., packets flow in one direction from s to d along a constrained routed path [3]. For the reverse traffic flow, an 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 path can differ. In the Virtual Private Network (VPN) literature [3], this request model is often referred to as the *pipe* model. We will refer to this model and the corresponding constrained path routing as the *asymmetric request* model. The algorithms reported in this paper assume this request model.

When uninterrupted network connectivity is necessary, a client may use LSPs from several NSPs to deal with occasional NSP failures. However, this requires multiple physical connections (ports) to different NSPs. To avoid this, an NSP may provide an enhanced service with additional guarantees: for every client request $\langle s, d, b \rangle$, the NSP sets up two LSPs between source s and destination d: a primary LSP that is used under normal circumstances, and a backup LSP that is activated in the event of disruption of the primary path due to link or switch failures. The mechanism used for detection of path disruption and switching over to the backup path has two variants:

Protection, whereby on link failure, endpoints automatically switch to a pre-configured backup path;
Restoration, whereby the backup path is only configured on demand when the primary path fails.

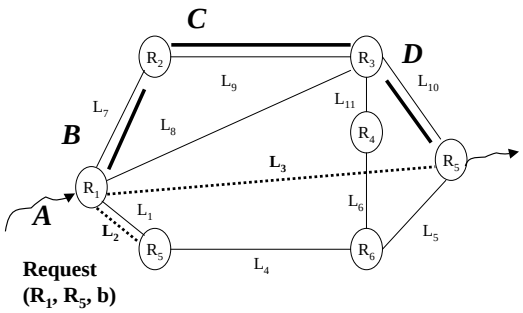
⋆ This paper is an expanded and revised version of our IEEE ICNP2001, November 2001, paper.

* Corresponding author.
Email addresses: norden@dnrc.bell-labs.com (Samphel Norden), mbuddhikot@bell-labs.com (Milind M. Buddhikot), mwl@zurich.ibm.com (Marcel Waldvogel), suri@cs.ucsb.edu (Subhash Suri).

[1] Part of the work reported here was undertaken during Samphel Norden's summer internship at Bell Labs.

[2] Marcel Waldvogel was with Washington University in St. Louis during the course of this research

[3] Subhash Suri was supported in part by NSF grants ANI 9813723 and CCR-9901958.



Path (L_7, L_9, L_{10}) ➔ Labels (B,C,D)

Fig. 1. Concept of label switching

Note that in both cases, resources are always allocated on primary *and* backup paths. However, in the first case, the backup path is always active and always consumes resources. We focus on the latter mechanism, whereby backup path restoration is performed to recover from failures.

Restoration routing also comes in two distinct flavors:

End-to-End path restoration, whereby link failures on the primary path cause an end-to-end backup path to be configured [4, 5];

Local Restoration, wherein each link on the primary path is protected by means of backup paths so that any link failure is treated locally for fast restoration [6, 7].

In this paper, we focus on the problem of end-to-end restoration routing and do not consider the problem of local restoration routing.

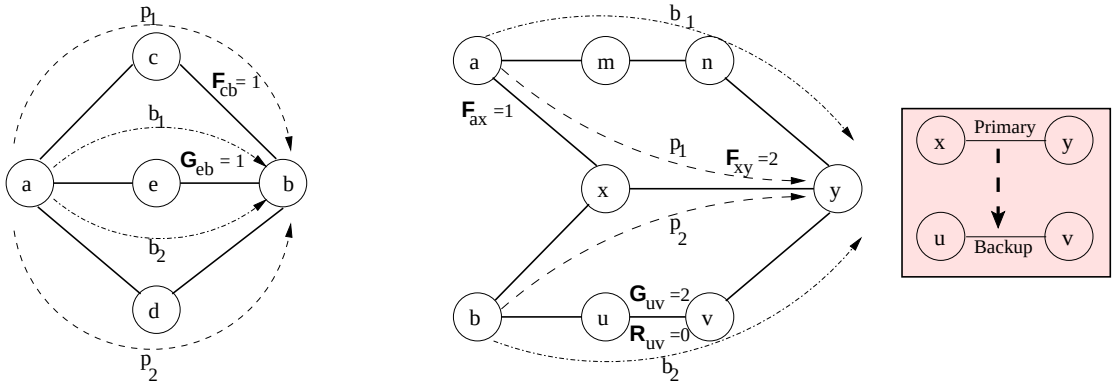
1.1 Overview of Main Ideas and Contributions

In this section, we present the problem formulation and illustrate the limitations of current mechanisms for backup restoration. All backup restoration mechanisms use the following state information in order to decide how to route backup paths:

$F_{u,v}$**:** amount of bandwidth used on link (u, v) by all primary paths that use link (u, v).
$G_{u,v}$**:** amount of bandwidth used by all backup paths that contain link (u, v).
$R_{u,v}$**:** residual capacity on the link (u, v) defined as $R_{u,v} = C_{u,v} - (F_{u,v} + G_{u,v})$.

Henceforth, we will refer to this scenario as the 3-Variable Partial Information (3VPI).

We describe a simple example of a 4-node topology in Figure 2(a) to illustrate the use of these variables. Consider two requests $r_1 = \langle a, b, 1\rangle$ and $r_2 = \langle a, b, 1\rangle$. Let all links have capacity of 1 unit. Let us consider primary paths p_1 and p_2 which use the paths ($L_{a,c}, L_{c,b}$) and ($L_{a,d}, L_{d,b}$) respectively. If we assume that a single link fails at any given time (see Section 2.2 for assumptions of fault models), p_1 and p_2, which do not share any links, will not fail simultaneously. This allows their backup paths b_1, b_2 to share the same links ($L_{a,e}, L_{e,b}$). In this example, $F_{a,d} = F_{a,c} = F_{c,b} = F_{d,b} = 1$ unit, $G_{a,e} = G_{e,b} = 1$ unit. Also, residual capacity on all links will be 0. Note that if p_1 and p_2 shared even



(a) Example (b) Limitations

Fig. 2. State Information

a single link, the backup paths for both must be necessarily distinct and there can be no sharing of backup paths between the two requests.

Now consider routing backup paths with only the knowledge of F, G, and R for each link with a more detailed example as shown in Figure 2(b). Let us now assume that all links except link $L_{u,v}$ have capacity 3 and $L_{u,v}$ has capacity 2.

Consider two requests $r_1 = \langle a, y, 1\rangle$ received at node a and $r_2 = \langle b, y, 1\rangle$ received at node b. The primary path $p_1 = (L_{a,x}, L_{x,y})$ for r_1 and $p_2 = (L_{b,x}, L_{x,y})$ for r_2 share common link $L_{x,y}$. This implies that the load on $L_{a,x}$ and $L_{b,x}$ due to primary paths is 1 ($F_{a,x} = F_{b,x} = 1$) unit, whereas on link $L_{x,y}$ it is 2 units ($F_{x,y} = 2$). Furthermore, r_1 uses backup path $b_1 = (L_{a,m}, L_{m,n}, L_{n,y})$ and request r_2 uses backup path $b_2 = (L_{b,u}, L_{u,v}, L_{v,y})$.

When node b computes backup path for r_2, it is unaware that r_1 does not use $L_{u,v}$ in its backup path. In absence of such knowledge of the distribution of the bandwidth on $L_{x,y}$, the coarse granularity or the scalar nature of $F_{x,y}$ forces node b to backup entire load on $L_{x,y}$ on $L_{u,v}$. This implies that the backup bandwidth used on $L_{u,v}$ ($G_{u,v} = 2$), even though r_1 uses $L_{m,n}$ for backup and not $L_{u,v}$. The inaccurate nature of such a model will cause the residual capacity on $L_{u,v}$ to be 0 ($R_{u,v} = 0$), even though there is *free shareable capacity* of 1 unit on $L_{u,v}$ that can be used for routing backup paths. This term which is described in more detail in Section 4.2 refers to the real amount of residual capacity that is not expressed in the coarse-grain $G_{u,v}$ parameter. Thus, if a new request r_3 needs to be routed from a to y and uses $L_{x,y}$ on its primary path, it will not be able to use $L_{u,v}$ as its backup path since the residual capacity will appear to be insufficient. This is just one of the drawbacks of using such coarse-grain parameters. Section 3 discusses this problem, formally termed as *Primary-to-Backup Link Wastage*, and other limitation called *Bandwidth Release Ambiguity* in more detail.

In this paper, we propose the new concept of a Backup Load Distribution (BLD) matrix $\boldsymbol{BM}$ that captures partial network state, yet exposes sufficient information to minimize bandwidth wastage and maximize backup path sharing. We describe two new distributed routing algorithms that utilize the BLD matrix and run in bounded time. The proposed BLD matrix $\boldsymbol{BM}$ can be exchanged among peer routers using the OSPF extensions for QoS routing [1]. This allows our algorithms to be realized in

the existing Internet architecture. Our simulation results for sample network topologies show a 50% reduction in the number of rejected requests and 30–40% savings in total bandwidth used for backup. We also evaluate the overhead of communicating the BLD matrix in a distributed implementation and study the effect of stale state information as the BLD update frequency is changed. We show that although the performance of routing schemes is sensitive to the frequency of state updates, for practical and reasonable values of update frequencies the performance degradation is minimal. The BLD matrix concept, our algorithms, and our simulation experiments apply to any generic label-switching technology and hence can be used in optical path routing in Wavelength-Division Multiplexing networks as wall as Virtual Path Routing in ATM networks.

1.2 Outline of the paper

Section 2 presents background material for the discussions in the paper. Section 3 describes in detail the limitations of using partial network state information consisting of only three state variables per link, namely *residual bandwidth, bandwidth for primary paths,* and *bandwidth for backup paths*. The concept of the BLD matrix that eliminates these limitations is introduced in Section 4. In Section 5, we describe two new algorithms that use the BLD matrix, namely Enhanced Widest Shortest Path First (E-WSPF), and Enumeration-Based WSPF (ENUM-WSPF). Section 6 describes simulation experiments using realistic network topologies, and finally, Section 7 presents the conclusions.

2 Background

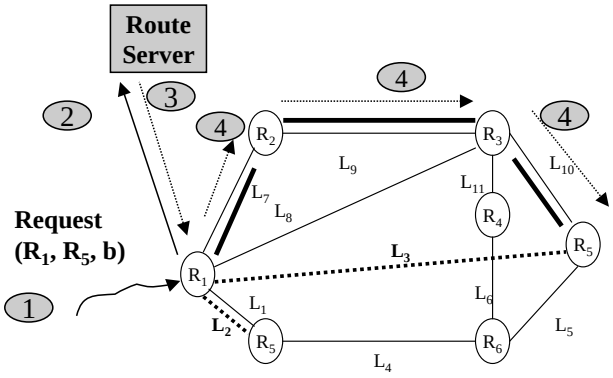
In this section, we will present relevant background material on various aspects of the problem such as characteristics of routing algorithms, fault model, concept of backup path sharing, and the basics of the primary path routing algorithm known as Widest Shortest Path First (WSPF) [8].

2.1 Characteristics of Routing Algorithms

The important characteristics of routing algorithms that we need to consider are the following:

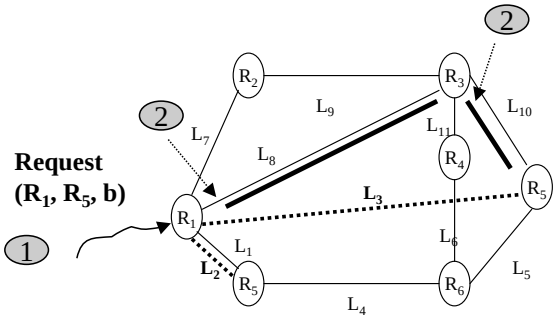
Online routing: This property requires that an LSP request can only be routed based on complete or partial knowledge of the *current* state of the network. Accepting a current request that generates a small revenue may potentially block a future request that could have generated a much larger revenue. In contrast, offline routing is based on a-priori knowledge of all LSP requests, enabling the revenue maximization by rejection of selected requests. Clearly, during network operation, an offline routing problem can be solved periodically to optimize the LSP routing and the available bandwidth, which however exceeds the scope of this paper.

Distributed vs. Centralized implementation: Route computation and management can be performed either (1) at a centralized route server or (2) in a distributed fashion at each router/switch.



Step 1: Request rxed by R_1
Step 2: Request forwarded to route server
Step 3: Route server computes route (R_1, R_2, R_3, R_5) and returns the route
Step 4: The route is signaled

(a) Centralized routing using route server



Step 1: Request rxed by R_1
Step 2: Router R_1 computes route (R_1, R_3, R_5) and signals the path

(b) Distributed routing

Fig. 3. Routing implementation

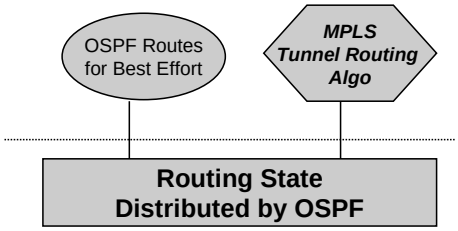


Fig. 4. Distributed routing algorithm

In the centralized approach (Figure 3(a)), each router forwards the incoming request for a new LSP to a well-known route server, which then computes and returns the route. In this approach, the route server has full information on the network state at its disposal for the route computation.

In the distributed implementation model (Figure 3(b)), a router computes routes for a LSP request based on its "local" view of the network state constructed from link-state updates sent by network nodes. In this case, the overhead of distributing per-path information whenever new paths are established or old ones removed can be prohibitively high. Therefore, distributed route computations are often limited to link-specific state instead of path-specific state, resulting in sub-optimal performance compared to their centralized counterpart.

For the ease of deployment, it is necessary that any new state information be collected and disseminated using existing routing protocols such as OSPF (Figure 4). The existing OSPF protocol disseminates topology and link state such as *up, down* status. The OSPF path-computation algorithm uses this information to construct the route table for forwarding the best-effort traffic. New

extensions to OSPF have been proposed to distribute additional link state such as residual link bandwidth, delay etc. required for QoS routing [2, 8]. The LSP routing algorithms will use such additional state information to construct MPLS paths and corresponding per-port label-swapping table.

2.2 Fault Model

In the context of protection or restoration path routing, it is important consider two kinds of failures, namely, link failures and router failures. A common fault model for link failures assumed in literature and justified by network measurements [9, 10] is that at any given time only one link in the network fails. In other words, in the event of a link failure, no other link fails until the failed link has been restored, and probability of two or more links failing at the same time is very small. In our work, we use this link-failure model to devise our algorithms.

Modern IP routers still do not support the so-called *five-nines* (99.999%) or *seven-nines* (99.99999%) reliability common in telephony switches. Therefore, router failures may be more frequent than link failures. An ingenious way to model router failure is based on a technique often used in distributed system to model node failures: a router can be represented by two nodes connected by a link with infinite capacity. The router failure is then simulated by a failure of this internal link.

2.3 Backup-Path Sharing

Given the typical fault model of single-link failure, we are guaranteed that in the event of a link failure, two paths will not fail simultaneously, if they are link disjoint. As a result, backup paths for two link-disjoint primary paths can share capacities on their backup links because at most one of the backup paths will be active at any one time. Therefore, if two LSPs, each with a bandwidth requirement of b units, are routed on link-disjoint paths, their backup can be provided by a single path with capacity b. Such bandwidth sharing allows one of the two primary paths to use the backup free of cost. This suggests that backup-path routing can exploit the fault model to maximize backup-path sharing.

The amount of sharing that can be achieved by an online algorithm over a series of N requests depends on the amount of state information at its disposal. A limited amount of state information can lead to a pessimistic link selection and increased request rejection.

2.4 Widest Open Shortest Path First

The Widest Shortest Path First (WSPF) algorithm was first proposed by Apostolopoulos et al. [8] for the routing of bandwidth-guaranteed paths. As our restoration routing schemes use WSPF as an integral component, we will present it briefly.

The drawback of using the traditional Shortest Path First (SPF) algorithm is that it may yield an

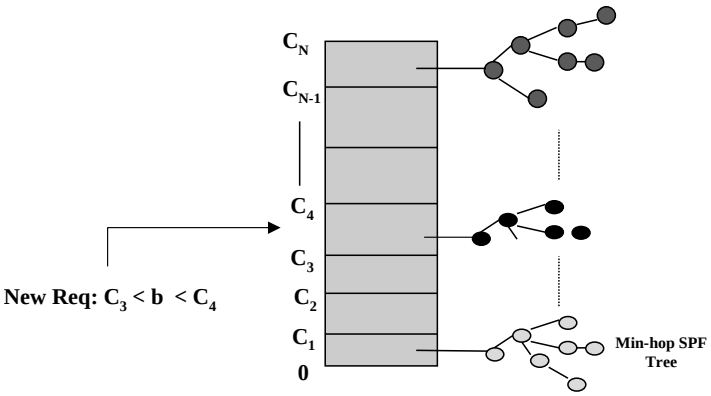


Fig. 5. WSPF data structures

optimum solution for a single request, but it can lead to high request rejection and low network utilization over a span of N requests [8, 11, 12]. The WSPF algorithm remedies this problem by selecting a shortest path with maximum ("widest") residual capacity on its component links. In order to minimize the overhead of computing the shortest path and of distributing the state information in a distributed implementation, Apostolopoulos et al. propose two improvements:

Quantization: Quantize the bandwidth on a link into a fixed set of ranges or bins. When a new LSP request is received, the request is quantized to a fixed bin and can be satisfied by selecting a path with links that belong to this or a higher bin.

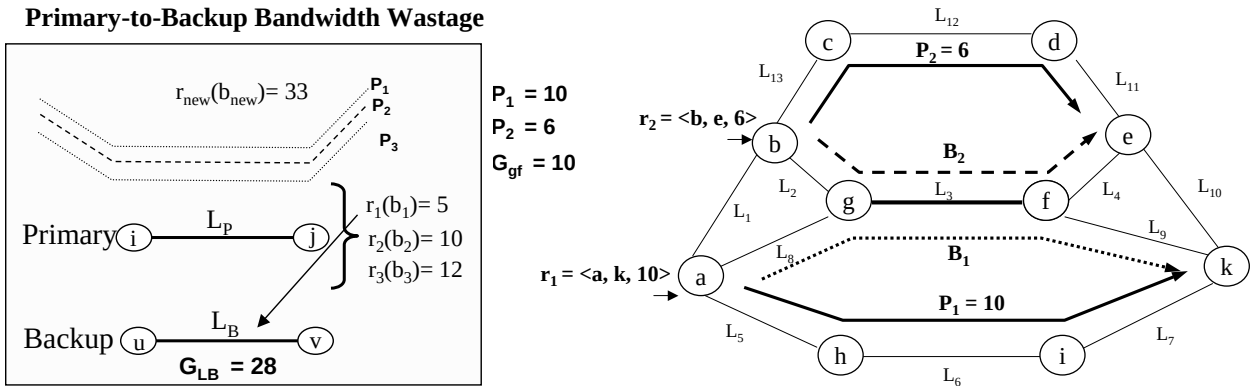
Pre-computation: For each quantization level or bin, compute an SPF tree from every source edge router to all destination edge routers.

Figure 5 illustrates these concepts. The SPF tree essentially records the shortest paths from a source to all egress nodes. Note that every time the residual bandwidth on a link changes a quantization level, the SPF trees for the old and new levels need to be recomputed. The complexity of the WSPF pre-computation for k bandwidth levels in a network of n nodes and m links is $O(kmn \log n)$ in the worst case.

A drawback of WSPF is that it does not take the knowledge of the nature of traffic between ingress-egress pairs into account. New primary-path routing schemes such *Minimum Interference Routing Algorithm* (MIRA) [11, 13] and *Profile-Based Routing* (PBR) [12] attempt to address this limitation and have reported better performance. Nevertheless, we chose WSPF, as PBR is not well suited to our distributed approach and we felt the simplicity of WSPF helped us better understand the impact of changes and would distracted less from the our main focus of primary-backup routing.

3 Limitations of Using 3VPI Partial Network State

In the following, we show that the use of three state variables (R_L, F_L, G_L) per link L leads to two problems: *primary-to-backup link wastage* during request admission and *bandwidth release ambiguity* during request teardown.



(a) Primary-to-backup link wastage

(b) Ambiguity in bandwidth release during request teardown

Fig. 6. Issues with 3VPI

3.1 Primary-to-Backup Link Wastage

We illustrate this concept with an example in Figure 6(a). Consider link L_P between nodes i, j. Three existing primary paths P_1, P_2, P_3 routed for requests r_1, r_2, r_3 with bandwidth requirements $b_1 = 5, b_2 = 10, b_3 = 12$ use this link. This results in a load of $F_{L_P} = 27$ units due to the primary path. Let us assume that the new request r_{new} to be routed on L_P requires $b_{\text{new}} = 33$ units of bandwidth. The backup-path routing tries to evaluate the suitability of link L_B between nodes u, v as a member of the backup path. Let us further assume that only request r_1 uses link $L_B = (u, v)$ on its backup path. Also, let the current load on L_B induced by backup paths be $G_{L_B} = 28$ units and the residual capacity $R_{L_B} = 12$.

First consider the use of *complete network state* information. The routing algorithm knows that of the primary-path load F_{L_P} only the primary path for r_1 is backed up on a path that uses link L_B. Therefore, out of $G_{L_B} = 28$, only 5 units are induced by link L_P and an extra 23 units of bandwidth already reserved are available for backing up the new request. Because $R_{L_B} = 12 > ((b_{\text{new}} = 33) - 23) = 10$, the complete-information case will allow the selection of link L_B in the backup path.

Now consider the *partial-information* scenario. In contrast, only the absolute $F_{L_P}, G_{L_B}, R_{L_B}$ values are known, and the algorithm does not know the distribution of F_{L_P} on link L_B. This forces a pessimistic assumption that in the event of failure of link L_P, not $b_1 = 5$ but $b_{1,2,3} = b_1 + b_2 + b_3 = 27$ units may have to be backed up on L_B. Clearly, the sum of the sharable backup bandwidth and the residual capacity, $(G_{L_B} - b_{1,2,3}) + R_{L_B} = ((28 - 27) + 12) = 13$, is less than the new request size $b_{\text{new}} = 33$, and therefore, L_B will not be selected as a potential link in the backup path. In other words, lack of additional information can lead to assuming that the subgraph available to route the backup is disconnected. This will then cause the request to be rejected. We call this phenomena, which results from pessimistic link selection and leads to reduced bandwidth sharing, and increased request rejection, as *primary-to-backup link wastage*.

3.2 Ambiguity in Bandwidth Release

Figure 6(b) illustrates an example of backup bandwidth release ambiguity. In this network, router a receives the first path request $r_1 = \langle a, k, 10\rangle$ and routes primary path $P_1 = (L_5, L_6, L_7)$ and backup path $B_1 = (L_8, L_3, L_9)$. It reserves 10 units of bandwidth on both paths. Router b receives the second request, $r_2 = \langle b, e, 6\rangle$ and computes primary path $P_2 = (L_{13}, L_{12}, L_{11})$ and backup path $B_2 = (L_2, L_3, L_4)$. Note that backup paths B_1 and B_2 share link L_3. As P_1 and P_2 do not fail simultaneously, r_2 concludes that 10 units of backup bandwidth on L_3 can be used as free bandwidth for B_2 and therefore does not reserve additional bandwidth on L_3 for backup.

When router a tears down request r_1, tearing down the primary part (P_1) is straightforward, but terminating backup path B_1 is problematic. Specifically, router a faces an ambiguity in deciding how much bandwidth to release on link L_3. When B_1 was set up, a reserved 10 units, 6 units of which are now shared by B_2. However, as router a has no path-specific knowledge, it does not know that path B_2 shares link L_3. In this case, a cannot release the correct amount of bandwidth without additional knowledge. We call this limitation imposed by using only three state variables for path routing the *bandwidth release ambiguity*.

In the following, we show how *primary-to-backup bandwidth wastage* and *bandwidth release ambiguity* can be averted using limited additional state.

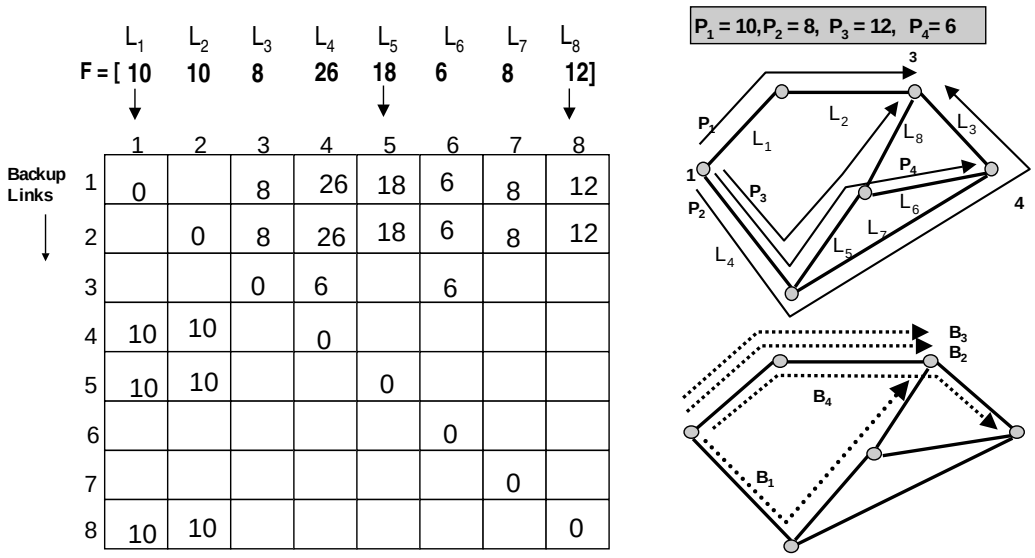
4 Backup-Path Routing using the Backup Load Distribution Matrix

In this section, we describe a new form of state information called the *Backup Load Distribution (BLD)* matrix $\boldsymbol{BM}$ based on the concept of backup sharing [4] and illustrate how it can be employed to achieve superior backup-path sharing.

4.1 The BLD Matrix

Given a network with N links, each router maintains a $N \times N$ BLD matrix $\boldsymbol{BM}$. If the primary load F_j on link j is B units, entries $BM_{i,j}$, $1 \leq i \leq N$, $j \neq i$, record which fraction of B is backed up on link i. Figure 7 illustrates this concept with an example network having eight links and four primary paths P_1, P_2, P_3, P_4 with bandwidth requirements of 10, 8, 12, 6 units. The corresponding backup paths B_1, B_2, B_3, B_4 are also illustrated. Figure 7 also lists four vectors maintained by each network node:

(1) capacity vector $\boldsymbol{C}$ that records the link capacities,
(2) vector $\boldsymbol{F}$ that records the load induced on each link by primary paths,
(3) vector $\boldsymbol{G}$ that records the load induced on each link by the backup paths, and
(4) vector $\boldsymbol{R}$ that records the residual capacity on each link.



Capacity C = [50, 50, 150, 150, 50, 50, 50, 150]
Primary Load F = [10, 10, 8, 26, 18, 6, 8, 12], max F = 26
Backup Load G = [26, 26, 6, 10, 10, 0, 0, 10]
Residual Capacity R = [14, 14, 136, 114, 22, 44, 42, 128]

Fig. 7. Example of a BLD matrix $\boldsymbol{BM}$

Consider link L_4. Primary paths P_2, P_3, P_4 use this link, and therefore its primary load is $F_{L_4} = 8 + 12 + 6 = 26$ units. The corresponding backup paths are $B_2 = (L_1, L_2)$, $B_3 = (L_1, L_2)$, and $B_4 = (L_1, L_2, L_3)$. As the primary paths are not link disjoint, the backup load on the component links evaluates to $G_{L_1} = G_1 = 26$, $G_{L_2} = G_2 = 26$, $G_{L_3} = G_3 = 6$.

We can now see that out of $F_{L_4} = 26$ units of primary load on L_4, $8 + 12 + 6 = 26$ units are backed up on L_1 and L_2, whereas six units are backed up on L_3. Per the definition of the BLD matrix, this is recorded as $BM_{1,4} = 26, BM_{2,4} = 26, BM_{3,4} = 6$.

Note that for row 2, $\max_{\forall j} BM_{2,j} = 26$ represents the maximum backup load on link L_2 induced by any link in the network. In general, for any row i, $\max_{\forall j} BM_{i,j}$ represents the maximum backup load induced on link i by all other links. Clearly, for any link i, $\max_{\forall j} BM_{i,j} \leq G_i$. Note further that if the entries in row i are sorted in decreasing order, we can identify links that induce successively smaller amounts of backup load on link i. This knowledge helps in answering questions such as (a) which links induce the most backup load on link i, or (b) out of N links, which links induce 50% of backup load on i.

The primary-to-backup link wastage described earlier is avoided by use of the BLD matrix. For the example shown in Figure 6(a), BM_{L_B,L_P} would be 5 as only request $r_1 = 5$ that uses L_p is backed up on L_B, thus avoiding the pessimistic assumption that the entire primary load on L_p may be backed up on L_B.

Similarly, the bandwidth release ambiguity can be eliminated using the BLD matrix. In Figure 6(b), when router a needs to release bandwidth on link L_3, it recalls that when the backup for request r_1 was routed using L_3, 10 units of bandwidth were reserved. It consults the $\boldsymbol{BM}$ row corresponding to link L_3, where each column lists which fraction of the primary path load F on link L_i, $i \neq 3$, is backed

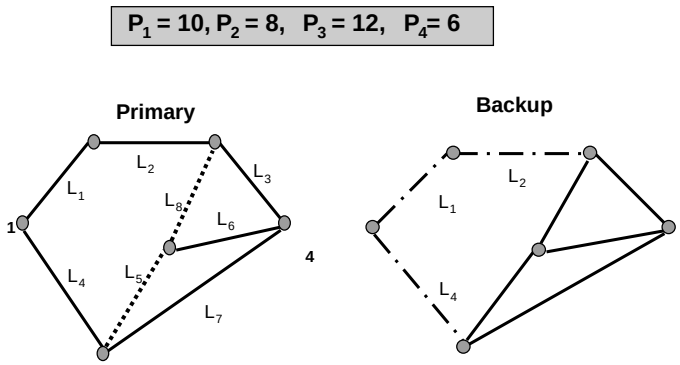


Fig. 8. Free bandwidth on a link available for backup sharing

up on L_3. In our example, $BM_{L_3,L_{13}} = BM_{L_3,L_{12}} = BM_{L_3,L_{11}} = 6$, and $BM_{L_3,L_5} = BM_{L_3,L_6} = BM_{L_3,L_7} = 10$. In this case, router a concludes that primary paths routed through L_{13}, L_{12}, L_{11} use up to 6 units of backup reservation on link L_3. Therefore, even though router a reserved 10 units of backup bandwidth on L_3, it releases only

$$\min \begin{cases} \text{BW reserved on } L_3 \text{ on backup for request } r_1 & \text{(a)} \\ (G_{L_3} - \max_{j \notin (L_{11},L_{12},L_{13})} BM_{L_3,j}) & \text{(b)} \end{cases} \tag{1}$$

which is $\min(10, (10-6)) = 4$ units. In general terms, consider a request r with primary path P, and B such that amount X was reserved on link j in the backup path when B was routed. Then, the bandwidth released on link j when request r is removed is given as

$$\min \begin{cases} X \\ (G_j - \max_{i \notin P} BM_{j,i}) \, . \end{cases} \tag{2}$$

4.2 Freely Shareable Bandwidth

In the following, we introduce the concept of *freely shareable bandwidth on a link* and show how the use of the BLD matrix allows its accurate computation. Consider the example network in Figure 7 with associated BLD matrix $\boldsymbol{BM}$ and the $\boldsymbol{F}$, $\boldsymbol{G}$, and $\boldsymbol{R}$ vectors. Figure 8 shows a snapshot of this network in which, in response to a new LSP request r_{new}, a candidate primary path (L_5, L_8) has been routed but not reserved and (L_4, L_1, L_2) is under consideration as a backup-path candidate. We can see from vector $\boldsymbol{G}$ (Figure 7) that the maximum backup load induced on (L_4, L_1, L_2) is $(10, 26, 26)$.

Let us take a closer look at link L_1. From the BLD matrix, we know that the backup load induced by links in the candidate primary path, namely (L_5, L_8) on L_1, is $(BM_{1,5}, BM_{1,8}) = (18, 12)$. Accordingly, a maximum 18 out of 26 units of backup reserved on L_1 will be required for backing up the primary load on (L_5, L_8) even before the new request r_{new} is admitted. In other words, there are 8 extra units of backup bandwidth reserved for backing up some other links. If the new request requires fewer than 8 units of bandwidth, then no extra bandwidth needs to be reserved on link L_1 in the candidate backup path. We call these 8 units of bandwidth on link L_1 the *freely shareable* bandwidth.

Formally, given a primary path P, the freely shareable (FR) bandwidth available on a candidate backup link L is defined as

$$FR_L = G_l - \max_{i \in P} BM_{L,i} \qquad . \tag{3}$$

In our example, for backup path (L_4, L_1, L_2), $FR_{L_4} = 10$, $FR_{L_1} = 8$, $FR_{L_2} = 6$, and therefore, if request size b_{new} is 6 units or fewer, no bandwidth needs to be reserved on the candidate backup path. As shown, the BLD matrix $\boldsymbol{BM}$ allows a more accurate computation of freely shareable backup bandwidth on a link.

4.3 Modeling the Link Cost

The backup-path computation procedure should favor links that have large freely shareable backup bandwidth. From the perspective of backup routing, every link has two kinds of bandwidth available:

Freely shareable bandwidth (FR), which is completely shareable and requires no extra resource reservation.
Residual bandwidth (R), i.e., 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 consists of two parts: (a) the cost of using the free bandwidth on the link and (b) the cost of using the residual bandwidth on the link. The per-link cost is then as follows:

$$w_l = \begin{cases} C_F(FR_l), & \text{if } b \le FR_l, \\ C_F(FR_l) + C_R(b - FR_l), & \text{if } FR_l < b \le FR_l + R_l, \\ \infty, & \text{if } FR_l + R_l < b \qquad \text{(i.e., } l \text{ is infeasible)}, \end{cases} \tag{4}$$

where C_F and C_R are cost metric functions selected in such a way that links with high residual capacity R_l are preferred. In other words, if $R_{l_1} < R_{l_2}$, then $C_{R_{l_1}} > C_{R_{l_2}}$. One such function is $C_{R_l} = a(1 - \frac{R_l}{R_{\max}})^p$, where $R_{\max} = \max_l R_l$. Similarly, if $F_{\max} = \max_l F$, then $C_{F_l} = c(1 - \frac{F_l}{F_{\max}})^q$, satisfies the constraint that if $FR_{l_1} < FR_{l_2}$, then $C_{F_{l_1}} > C_{F_{l_2}}$.

For primary-path routing, the "free bandwidth" does not play a role as the bandwidth always has to be reserved and no sharing is possible. The cost in this case is therefore only the cost incurred in using the residual bandwidth.

Given this cost function for a link, our routing algorithms attempt to find backup paths with minimum cost, where the cost of the path is the sum of the costs of the component links.

4.4 Implementation Overhead

Whenever a node routes new primary and backup connections, it recomputes the BLD matrix entries. Frequent addition or deletion of paths changes the matrix entries and requires state exchange between network nodes. For a network of fixed size, the size of the BLD matrix and therefore the maximum size of state exchanged between network nodes is fixed and independent of the number of paths. In other words, the BLD matrix captures only the link state induced by paths but no path-specific state. If the state exchange is completely distributed and copies of the BLD matrix at different nodes are inconsistent, two or more nodes may end up selecting paths consisting of links that do not have sufficient capacity to accommodate their requests. In this case, the reservation attempt of some of the nodes will fail and their requests will be rejected. The BLD matrix entries will be consistent again after subsequent state updates have been processed.

Consider the scenario of a distributed global exchange of the BLD matrix among all routers in the networks: If there are M routers and N links, the BLD matrix is N^2 in size. A naive exchange of the $\boldsymbol{BM}$ among M routers will require the exchange of $M(M-1)N^2$ entries. However, note that when a router routes a primary path P of length l links with a backup path B of j links, only the $\boldsymbol{BM}$ entries corresponding to l links in path P change. Therefore, instead of N^2 entries, only entries in l columns can change, at most lN values. In most cases, this can be even simplified to lk, as $l, k \ll N$, the update overhead is reduced to $\approx M(M-1)lk$. Also, it is sufficient to send updates only to the immediate neighbors instead of to all $M-1$ other nodes. If the out-degree of network nodes is limited to a maximum of p nodes, then total BLD-matrix-exchange cost is bounded by $Mplk$. As $p \ll M$, the reduction is significant. In addition, to reduce the frequency of the updates, we can send an update only when there is a significant change to the column-vector entries. In practice, to reduce the size of the updates, we can compress the column vector by only sending entries with non-zero values along with a preamble indicating the links to be updated. Note that, as for other link-state information, we can also adopt the existing policy of triggered updates.

An alternative centralized scheme that can minimize the BLD-matrix-distribution overhead and resulting inconsistencies uses *repository* nodes. The routers dynamically elect one or more among themselves to act as repositories for the BLD matrix state and to serve it to other network nodes. In the event of BLD matrix changes, each node registers its changes with the repository nodes and is also notified of changes made by others. The routers can periodically or upon the arrival of a path setup or teardown request, query and download the BLD matrix.

In the distributed exchange scheme, the well-known link-state routing protocol OSPF [8, 14] can be used to propagate BLD matrix entries. The changes to OSPF are not discussed here, as they are analogous to the descriptions in [2, 8], to which the reader is referred for further details.

5 Routing Algorithms

In this section, we will describe two types of algorithms:

Two-step algorithm: This algorithm first computes a primary path using one of the many available algorithms such as MIRA [11], PBR [12], or WSPF [8]. For this candidate primary path, the algorithm then computes a least-cost backup path.

Iterative or enumeration-based algorithm: This algorithm enumerates pairs of candidate primary and backup paths, and picks the pair with smallest joint cost. It uses the WSPF heuristic and associated data structures, and therefore is less generic.

Both algorithms use $\boldsymbol{F}, \boldsymbol{G}, \boldsymbol{R}$ variables per link and the BLD matrix, and run in a bounded amount of time. Note that both our algorithms can be deployed alongside OSPF for best-effort traffic and WSPF for primary-path QoS routing.

5.1 Generic Two-Step Algorithm

The basic pseudo-code for this algorithm that can be implemented in a route server or in a distributed fashion at each switch is as shown in the algorithm in Table 1.

The first step in this algorithm (line 10) computes the primary path P using an algorithm such as MIRA, PBR, or WSPF. If this step fails, the request is rejected (line 12). Because the backup and primary paths must be link disjoint, all links in P are removed from the graph on which the backup path is routed (line 15). Using the BLD matrix and Eq. 3, the algorithm then computes the FR_l on each link in the graph for the candidate primary path. Next, the algorithm removes all infeasible links from the graph and computes new graph G' (line 16). Using the cost metric defined in Eq. 4, it assigns a cost w_l to each link l and computes the backup path using the shortest-path algorithm on graph G' (lines 17, 18). If no path is found, the path request is rejected (line 19). Otherwise, an attempt is made to reserve the resources for the primary and backup paths using protocols such as RSVP [15] or LDP [16]. If reservation succeeds, the algorithm updates the path-related link-state variables and corresponding $\boldsymbol{BM}$ entries. It then sends state-change packets to the appropriate neighbors (line 24). If the reservation fails, the request is rejected.

We evaluated a specific instantiation of this generic algorithm using the WSPF algorithm for primary-path computation. We call this algorithm the Enhanced Widest Shortest Path First (EWSPF). The pseudo-code for the exact algorithm that uses the pre-computed WSPF data structures is illustrated in Table 2.

Steps 15, 16, and 17 in Table 1 require $O(m)$ time. Step 10 involves computation of a shortest path using Dijkstra's algorithm, taking $O(m \log n)$ time. Therefore, the worst case complexity of this algorithm is $O(km + m \log n) = O(m \log n)$, where n is the number of nodes and m is the number of links or edges in the network graph. Recall that k is the number of different bandwidth levels and is generally a small constant number.

Table 1
Generic two-step algorithm

```
00: var
01:    T:  Tree;                (* Tree data structure *)
02:    G, G':  NetworkGraph;    (* Network Graph data structure *)
03:    P, B:  Path;             (* Path data structure *)
04:    req: Request3Tuple;      (* 3-tuple: (src, dst, bw) *)
05:    cost: Integer;
06: procedure GenericTwoStep(s, d: node; b: integer);
07: begin
08: req.src := s; req.dst := d, req.bw := b;
09: (* Primary path computation *)
10: GetPrimaryPath(G, req, P);  (* Procedure uses preferred *)
                                (* primary path routing scheme*)
11: if P = NIL then begin
12:      writeln(''No Primary Path found'');
13:      exit;
14: end;
    (* Backup path computation *)
15: G' := RemoveLinks(G, P);    (* Remove primary path links from G. *)
                                (* G' contains the resulting graph *)
16: RemoveInfeasibleLinks(G', BLD, P); (* Remove links with *)
                                (* insufficient bandwidth from G' *)
17: AssignCostW(G', BLD, P);    (* Procedure to compute w_l induced)
                                (* by path prm on all links *)
18: B := SPFBackUpPath(G');     (* Procedure to compute backup *)
                                (* using shortest-path-first*)
19: if B = NIL then begin
20:      writeln(''No backup path found, Request Rejected'');
21:      exit;
22: end;
23: UpdateNetworkState(G, prm, bkp); (* Change the network state *)
                                (* after new paths are routed*)
24: end;
```

5.2 Enumeration-Based Algorithm (ENUM-WSPF)

This algorithm enumerates candidate pairs of primary and backup paths using pre-computed data structures in the WSPF implementation and therefore is called ENUM-WSPF. The basic idea in this algorithm is the following: Given a path request $\langle s, d, b \rangle$, find the bandwidth bin the request is quantized to (line 9 in Figure 5). Using the SPF trees stored in the bin *bin*, find the shortest path from s to d (lines 11 and 12). Treat this path as a *hypothetical backup path* and find a primary path that induces the least cost w_l on this path by searching the SPF trees in all other bins. The search is accomplished by the inner *for* loop (lines 14–27). When searching for the primary path, it is likely that, once links for the backup path have been removed, the tree at a given bin may be disconnected for the required s and d pair (line 18). In this case, a more expensive shortest path computation is done on the original graph (lines 19 and 20). Using the BLD matrix, Eq.s 3 and 4, and the cost of primary path, the joint

Table 2
Enhanced WSPF

```
00: var
01:    T: Tree;                  (* Tree data structure *)
02:    G, G': NetworkGraph;      (* Network Graph data structure *)
03:    P, B, BestP, BestB: Path;(* Path data structure *)
04:    req: Request3Tuple;       (* 3-tuple: (src, dst, bw) *)
05:    bin, cost, mincost: integer;
06: procedure EnhancedWSPF(s, d: node; b: integer);
07: begin
08: req.src := s; req.dst := d; req.bw :=  b; mincost := ∞;
09: bin := Quantize(b);          (* Quantize size to find bin *)
                                 (* this request corresponds to *)
10: for lvl := bin to k do       (* Search this and larger-sized bins *)
11:    (* Do primary path computation *)
12:    T := GetSPFTree(lvl, s); (* SPF tree rooted at s at level lvl *)
13:    P := GetPrimaryPath(T, d);(* Get path to d from s in T *)
14:    if P = NIL then continue;(* No luck, try next *)
15:    G' := RemoveLinks(G, P); (* Remove primary path links *)
16:    (* Do Backup path computation *)
17:    AssignCostW(G', BLD, P); (* Assign w_l induced by P on all *)
18:                             (* links l. Use BLD matrix *)
19:    B := SPF(G', s, d);       (* Run SPF on G' to get backup path B *)
20:    if B = NIL then continue;
21:    cost := JointCost(P, B); (* Joint cost of both paths *)
22:    if mincost > cost then
23:    begin
24:       BestP := P;           (* Current best primary path *)
25:       BestB := B;           (* Current best backup path *)
26:    end;
27: end;
28: UpdateLinkState(G, P, B);   (* Update residual bandwidth r_l *)
29:                             (* forward and backward load *)
30: UpdateBLDMatrix(G, BLD);    (* Update BLD matrix *)
31: SendOSPFUpdates();          (* Send OSPF updates if required *)
32: end;
```

cost of the (P, B) pair is computed (lines 22 and 23) and compared to the current best pair (lines 24–27). At the end of the inner *for* loop (line 28), the best primary path for the backup path from *bin* is selected. The process is then repeated for every higher bin (*bin* $\leq$ *lvl* $\leq k$) (outer *for* loop, lines 10–29). Clearly, this approach enumerates pairs of primary and backup paths and selects the pair with least joint cost.

The complexity of this algorithm is $(kmn \log n)$ for pre-computation and $O(k^2)$ for the cost comparison.

Table 3
ENUM-WSPF

```
00: var
01     T: Tree;                  (* Tree data structure *)
02:    G, G': NetworkGraph;      (* Network Graph data structure *)
03:    P, B, BestP, BestP: Path;(* Path data structure *)
04:    req: Request3Tuple;       (* 3-tuple: (src,dst,bw) *)
05:    bin, cost, mincost: integer;
06: procedure ENUM_WSPF(s, d: node; b: integer);
07: begin
08: req.src := s; req.dst := d; req.bw :=  b; mincost := ∞;
09: bin := Quantize(b);          (* Quantize size to find bin *)
                                 (* this request corresponds to *)
10: for lvl := bin to k do begin
11:    T := GetSPFTree(lvl, s); (* SPF tree rooted at s in T *)
12:    B := GetPath(T, d);       (* Candidate backup path in T *)
13:   if B = NIL then continue; (* None possible, try next *)
14:   for j := 1 to min(k, lvl-1) do begin
15:      T := GetSPFTree(j, s); (* SPF tree rooted at s in level j *)
16:      T' := RemoveLinks(T, B);(* Remove links on backup path from T*)
17:      P := GetPrimaryPath(T', d);(* primary path in T'*)
18:      if P = NIL then begin  (* Oops! T' is disconnected. *)
19:         G' := RemoveLinks(G, B);(* Remove backup path links from G*)
20:         P := SPF(G');        (* Find alternate shortest path *)
                                 (* as primary path in G' *)
21:      end;
22:      AssignCostW(B, BLD, P);(* Cost induced by prm on bkp*)
23:      cost := JointCost(P,B);(* Joint cost of primary and backup*)
24:      if mincost > cost then begin
25:         BestP := P;          (* Current best primary path*)
26:         BestB := B;          (* Current best backup path*)
27:       end;
28:    end;
29: end;
30: end;
```

6 Simulation Results

In this section, we describe simulations that characterize the benefits of our proposed schemes. We conducted two sets of experiments:

Experiment Set I (EXPTSET-I) compares three different schemes: EWSPF, ENUM-WSPF, and simple Shortest Path First (SPF). We simulated two different SPF schemes: *SPF-HOP* uses min-hop-count as path metric, whereas *SPF-RES* uses link costs based on the residual capacity and computes the lowest-cost path. Both SPF schemes compute two independent paths (one used as the primary and other as the backup) and do not attempt to share backup paths.

Experiment Set II (EXPTSET-II) compares our EWSPF scheme with Kodialam et al.'s scheme using data sets used for [4].

Table 4
Simulation parameters for EXPTSET-I

Property	Values
Request (REQ) arrival	Poisson at every router
Mean Call holding time (HT)	100 time units, exponentially distributed
REQ Volume (RV)	50,000 to 300,000
Simulation time (STT)	Fixed 50,000 units
Maximum LSP Req. size (LF)	2.5%, 10% of the link capacity
Mean REQ inter-arrival time	Computed using RV and STT
Destination node selection	Randomly distributed

6.1 Simulator Details

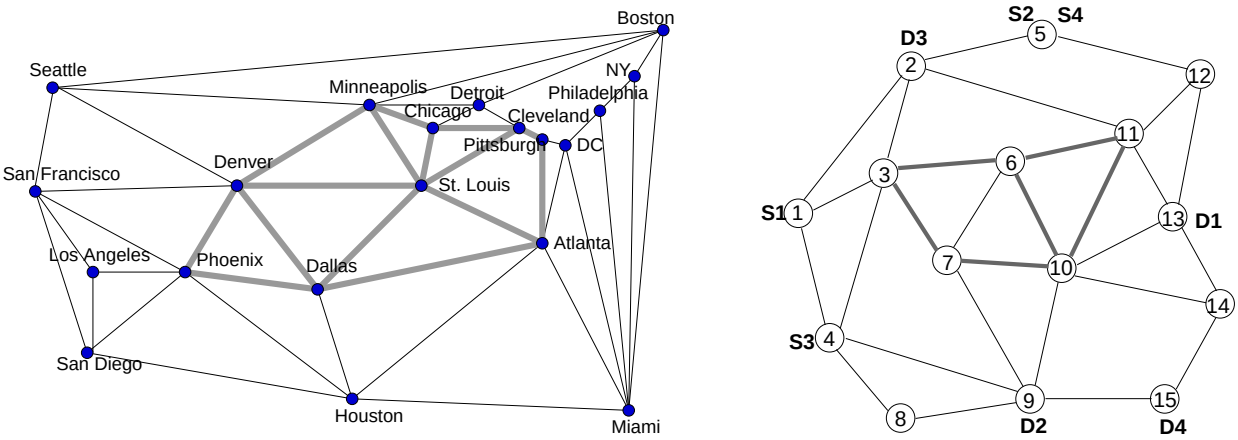
We developed a discrete event simulator in C++ to conduct a detailed simulation study. We simulated only certain aspects of the control path in the network and did not model the data path. Specifically, in the control path, we simulated the arrival and departure of path requests and dissemination of network state information. We did *not* simulate any of the following:

(1) actual data traffic such as TCP/UDP/IP packet flows on the routed primary path LSPs,
(2) the link fault events in response to which backup paths are activated,
(3) signaling protocols that detect and propagate link faults,
(4) or any other operational aspects irrelevant to routing protocol algorithm.

Therefore, our simulator captures the network state using network topology, routed primary, backup paths, per-link F, G, R variables and BLD matrix.

In the following, we describe the network topologies, traffic parameters, and performance metrics used.

LSP Request Load. Table 4 shows the parameters used to run the experiments in EXPTSET-I. We ran the experiments in EXPTSET-I by generating a given volume of requests (50,000 to 300,000) within a fixed simulation time (50,000 time units), effectively varying the LSP request load on the network. LSP requests at each router were modeled as Poisson arrivals, and the mean inter-arrival time was computed based on the total request volume during the simulation time. The call-holding time was exponentially distributed with a mean of 100 time units. The requests were torn down after the appropriate holding time, releasing resources for other new arrivals. The request bandwidth was varied using a uniform random variable with a maximum request size at 10% of the link capacity. We did not simulate the BLD and other state exchanges between the network nodes and therefore, did not measure effects of inconsistent state at the nodes. Note that in reality, the request load at various nodes may not be random and certain node pairs may see disproportionate share of requests. However, no real-life call traffic datasets are currently available in the public domain and no well



(a) Metropolitan Topology

(b) 15-node test topology from [4]

Fig. 9. Experiment Topologies

known methodology exists to generate them synthetically. Given this, we chose to use the LSP request load described earlier.

For the experiments in EXPTSET-II, Kodialam et al. supplied a modified version of the datasets they had used in their paper [4]. Their dataset contains 5 runs each with 100 demands. All demands have infinite call duration: once they are admitted, they do not terminate. The drawbacks of this dataset are (1) the number of demands in the dataset is too small and does not capture the statistical range required to achieve better averaging of performance metrics; (b) also, unlike the dataset in EXPTSET-I, the infinite connection-holding time used in this dataset does not resemble real network conditions, where connections are set up and torn down.

Network Topologies. For EXPTSET-I, we used the topology shown in Figure 9(a) in two configurations. The topology represents the Delaunay triangulation for the 20 largest metropolitan areas in the continental USA. The Delaunay triangulation has the feature that while it minimizes the number of parallel paths between a pair of nodes, it also provides redundant paths for failsafe routing when a link goes down, thus always allowing an alternate path [17, 18]. All routers were randomly selected as potential sources and destinations.

Homogeneous: In this case, all links in the network are of the same capacity (OC-48) and all routers are identical.

Heterogeneous: Here, we simulated a network consisting of a core with fast links that connects with slower links to an access network. Here, the thick links are OC-48 and the thin links are OC-12.

For the experiment set II (EXPTSET-II), to compare our EWSP scheme with Kodialam et al.'s scheme [4], we obtained the network topology (Figure 9(b)) they used in their paper.

Quantizing the Link Bandwidth for WSPF. We used the two bandwidth quantization schemes (Figure 5) in EWSPF and ENUM-WSPF schemes:

(1) Exponential quantization (EXP) used three bandwidth levels of 0.01, 0.1, and 1.0 times the maximum requested bandwidth.
(2) Uniform quantization (UNIFORM) used a more linear set of six levels, which allows to distinguish between 0.05, 0.1, 0.3, 0.5, 0.7, and 1.0 times the maximum requested bandwidth.

6.2 Performance Metrics

We used following performance metrics to compare the various algorithms.

Fraction Rejected (FR) is the fraction of requests that were dropped.

Total Bandwidth Saved Fraction (TBSF) is the fraction of total bandwidth saved when compared with SPF-RES. It is defined as

$$TBSF = \frac{TotalBW_{\text{new scheme}} - TotalBW_{\text{SPF}}}{TotalBW_{\text{SPF}}} \tag{5}$$

Backup Bandwidth Saved Fraction (BBSF) reflects the fraction of backup bandwidth saved for a given backup path by the new scheme compared to the one used by SPF algorithms. It is defined as

$$BBSF = \frac{BackupBW_{\text{SPF}} - BackBW_{\text{new scheme}}}{BackBW_{\text{SPF}}} \tag{6}$$

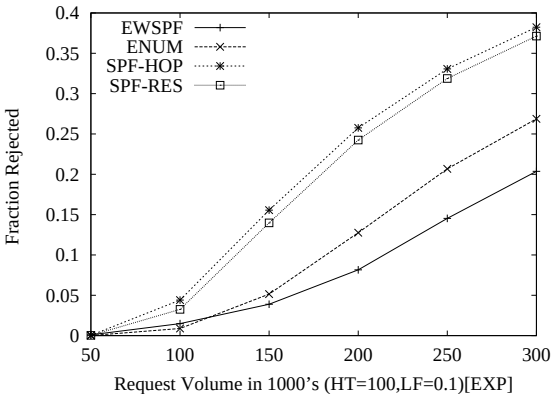
Note that this metric is different from TBSF which compares EWSPF and ENUM-WSPF with SPF. For a given scheme that picks a particular backup path, we hypothetically compute the gain realized by of using the shared bandwidth over using an SPF-like scheme that reserves the entire bandwidth even on the backup path. This metric thus is suitable for comparing EWSPF and ENUM-WSPF.

In EXPTSET-I, we measured both metrics, whereas in EXPTSET-II we measured only the FR metric, as values for other metrics were not available from Kodialam et al. [4].

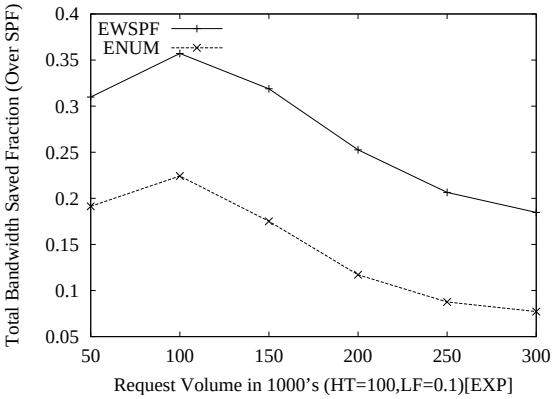
Note that one high-level performance metric that is of interest to the designer of an LSP network is the *path restoration latency* which corresponds to the amount time elapsed from the instance the link fault is detected to the instance the backup path is restored. However, this latency depends on several factors such as design of the signaling protocol for fault detection and propagation, how the control packets are handled in the network, and network load. The amount of time spent in backup path route computation is a small part of the total restoration latency. Since we model neither data path nor faults, we also do not measure path restoration latency metric. By precomputing backup paths and preassigning labels, no such computation is required at fault time and the backup path can be used immediately by the source.

6.3 Experiment Set I (EXPTSET-I)

Each simulation data point was the result of 10 runs with different seed values. The confidence interval was 95%.



(a) Fraction rejected



(b) TBSF

Fig. 10. Performance: Homogeneous Topology

Homogeneous Case. Figure 10 illustrates the FR and TBSF performance metrics for the four routing schemes, namely EWSPF, ENUM-WSPF, SPF-HOP, and SPF-RES.

Fraction Rejected: As expected, the FR increases as the load or RV increase. EWSPF and ENUM-WSPF are significantly better than SPF-HOP and SPF-RES, with up to 66% gain for 150,000 requests. As the load (volume) increases, EWSPF performs better than ENUM-WSPF. At 300,000 requests, EWSPF provides a 20% improvement over ENUM-WSPF and a 50% gain on SPF. ENUM-WSPF performs slightly worse than EWSPF because it uses the pre-computed trees for both primary and backup paths, whereas EWSPF uses the pre-computed trees only for the primary path and recalculates the link weights for the backup path. ENUM-WSPF trades off additional SPF computation and attempts to use the existing trees as much as possible. The main problem with using existing pre-computed information is that the same tree may appear in several bandwidth levels, nullifying the enumeration process and forcing ENUM-WSPF to resort to the shortest path using residual capacity. Hence the performance of ENUM-WSPF, which is still significantly better than SPF-RES, will tend to that of SPF-RES, especially at higher loads. In the remainder of the discussion, we will compare both EWSPF and ENUM-WSPF with SPF-RES, which performs slightly better than SPF-HOP.

TBSF vs. RV: In terms of the overall bandwidth saved compared with SPF-RES, we see that EWSPF saves 33% and ENUM-WSPF saves around 18%. The gain decreases with increasing load as links are saturated and therefore finding free shareable bandwidth becomes increasingly difficult as the number of requests increases.

BBSF: As shown in Figure 11, we see that EWSPF provides a better use of the shared bandwidth over the paths it chooses than ENUM-WSPF does. Effectively, EWSPF saves between 60–80%, whereas ENUM-WSPF saves 40–60%.

Heterogeneous Case.

FR vs. RV: Figure 12 shows FR vs. RV for LF values of 2.5% and 10% of the OC-48 links. As we have the access links at OC-12, an LF of 10% of OC-48 will result in some requests that are

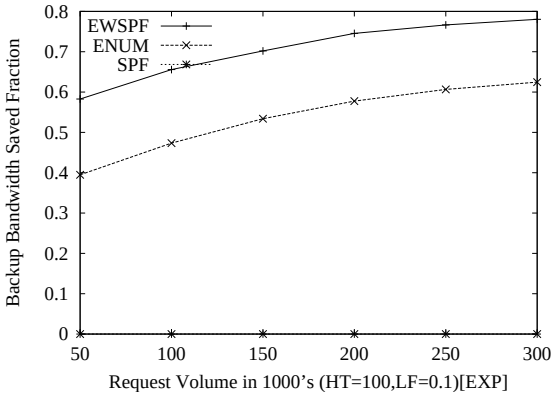
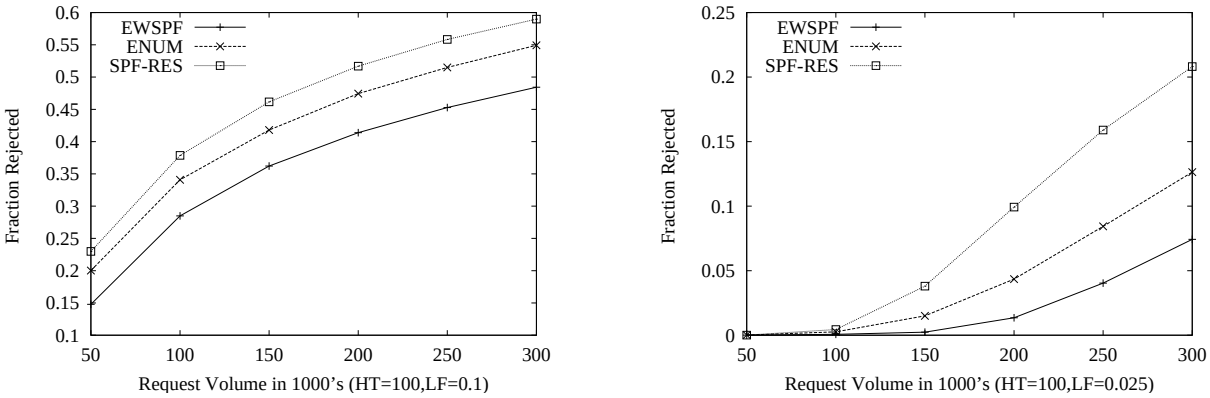


Fig. 11. Performance: BBSF vs. RV (homogeneous configuration)



(a) FR for LF = 0.025

(b) FR for LF = 0.10

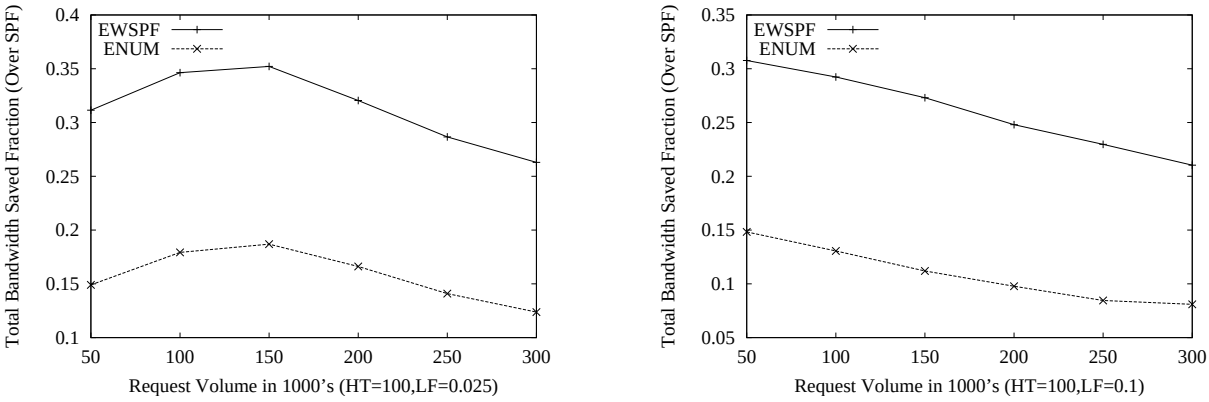
Fig. 12. Fraction rejected performance for heterogeneous configuration

nearly half of the total access link bandwidth. This causes the access links to become saturated very quickly, and leads to higher rejection probabilities for all schemes. The gains of EWSPF/ENUM-WSPF are also less than those of SPF-RES at the higher LF because of the early saturation leading to the dropping of requests.

TBSF vs. RV: We found that gains of EWSPF/ENUM-WSPF are sensitive to LF: the respective are smaller for the higher LF experiment. From Figure 13, we see that EWSPF provides around 28% gain and ENUM-WSPF provides 13% gain for a volume of 150,000 requests. However, once the LF is reduced to 2.5% [4] the gains for EWSPF and ENUM-WSPF improve to 35% and 20% respectively for the same request volume.

BBSF vs. RV: From Figure 14, we see EWSPF saving around 75% and ENUM-WSPF saving 45% at an LF of 10%. The corresponding gains in Figure 14 for an LF of 2.5% are between 60–70% for EWSPF and between 35–50% for ENUM-WSPF. The variation is less at the higher LF value since the links are saturated very early irrespective of the request volume, whereas with a smaller LF, the links take more time and accept more requests before being saturated leading to a larger variation.

[4] Graph for LF=2.5% not shown



(a) TBSF for LF=0.025 (b) TBSF for LF=0.10

Fig. 13. TBSF for heterogeneous case for different LF

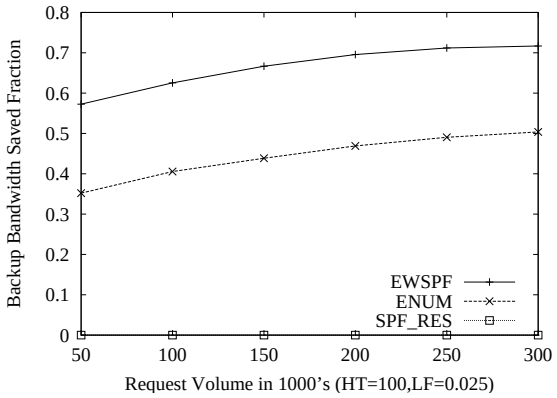


Fig. 14. Backup bandwidth saved for heterogenous case (where LF = 2.5%)

6.4 EXPTSET-II: Comparison with the scheme of Kodialam et al. [4]

From the results of EXPTSET-I, we see that EWSP performs well in all the cases we considered and is very simple to implement. Therefore, we selected EWSPF as a candidate algorithm to compare with the algorithm by Kodialam et al. We modified our simulator to handle the dataset described in Section 6.1. Kodialam et al.'s scheme models the backup path routing as a linear programming problem that uses only three variables, F, G, R. They develop a dual-based algorithm that solves the primal linear program to obtain an upper bound, UB, and it's dual problem to obtain a lower bound, LB. Iteratively running the algorithm reduces (UB-LB) difference and brings the solutions closer to the optimum. Each iteration involves solving many shortest path problems and a large number of iterations, ranging from 100 to 500, may be required to obtain a satisfactory convergence. We call this scheme *Linear Programming Approach* (LPA) for the remainder of this discussion.

We performed two kinds of experiments:

(1) Links have infinite capacity and no request is rejected.
(2) Links have finite capacity and requests are dropped.

Table 5
Comparison of EWSP with LPA

Scheme	Total BW (Exp. 1)	Request Rejection Fraction (Exp. 2)
EWSP	2722	0.062
LPA	2736	0.064

For the former set, we measured the total bandwidth that is reserved by the schemes for all requests. For the latter, we measured the rejection fraction. Our results are summarized in Table 5.

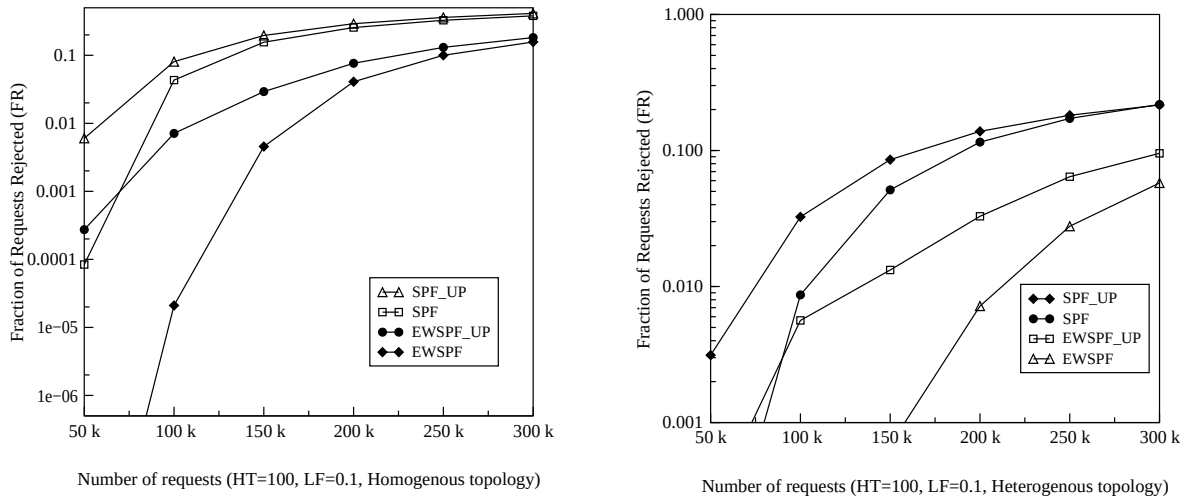
We can see that our scheme realized an improvement in the rejection fraction over LPA. The savings accrue from the use of BLDM to reduce the primary-to-backup link wastage described in Section 3. However, we also noticed a significant standard deviation in the five runs (each with 100 demands). We believe that the limited size of the datasets is the reason for these large deviations, as well as for the relatively small performance gains. Our scheme is very simple as it involves only two shortest-path computations, unlike LPA, which requires tens to hundreds of shortest-path calculations. Moreover, our scheme is easy to deploy because it is directly based on link-state protocols.

6.5 Experiments using Periodic Updates

We performed experiments to determine the impact of the frequency of BLD matrix updates on routing inaccuracies. Specifically, we simulated BLD matrix distribution via column vector updates performed periodically with a frequency of once every 0.30 time units. We assume that the updates are effectively distributed across the entire network. In the following figures, we study the performance of two algorithms, EWSPF and basic SPF. We performed experiments on the homogeneous and heterogeneous topologies described earlier. Each request was uniformly distributed on up to 10% of the link bandwidth on the homogeneous configuration and 2.5% for the heterogeneous configuration (to avoid near-immediate saturation of the access links). We denote by EWSPF(UPD) and SPF(UPD) the EWSPF and SPF versions with BLDM updates, respectively.

Figure 15(a) shows the rejection fraction for the EWSPF and SPF algorithms with and without updates (centralized approach). It is interesting that the performance of the model with updates is quite close to the centralized versions. We note that this is a homogeneous topology without much variation in the routes as all links have the same capacity. Figure 15(b) shows the impact on the more realistic heterogeneous topology, and we see the impact of stale link-state updates on the EWSPF and SPF versions, which perform significantly worse than their centralized counterparts. Note also that the chart is on a logarithmic scale.

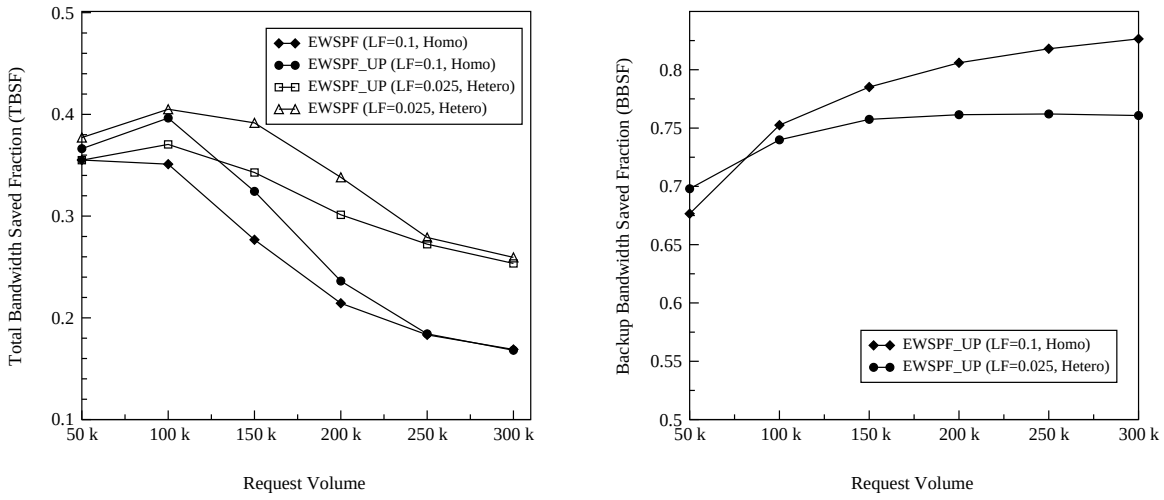
Figure 16(a) shows the fraction of total bandwidth saved for the EWSPF protocols (centralized and update-based models) over the corresponding SPF versions. The EWSPF version provides increased savings in bandwidth than the version with updates, as expected due to the stale link information that guides the EWSPF(UPD) version. It is interesting to note that the performance gap between the two does not change for the heterogeneous topology in Figure 16(a). At higher loads (250,000 requests), the savings of the update version is almost identical to the centralized scheme. Although the rejection fraction results are significantly worse for EWSPF(UPD) than for EWSPF, the overall bandwidth



(a) Homogeneous configuration

(b) Heterogeneous configuration

Fig. 15. Rejection fraction (BM update model; note the log scales)



(a) TBSF

(b) BBSF

Fig. 16. Performance of different configurations (BM update model)

saved is still close to that in the centralized model.

Figure 16(b) shows the fraction of backup bandwidth saved for the homogeneous and heterogeneous topologies. The performance of the EWSPF(UPD) scheme is almost similar to that of the centralized EWSPF scheme shown earlier.

Finally, we show the impact of the update period on the rejection fraction for the SPF and EWSPF protocols for the heterogeneous topology. This was evaluated at a specific request volume of 100,000 requests. As the update period increases, the probability of using stale link state also increases. Recall that the mean call duration is 100 time units. As we progress from 10-time-unit period to a 200-time-unit update period, we see a significant drop in performance (up to a factor of 2.5 for SPF and a factor

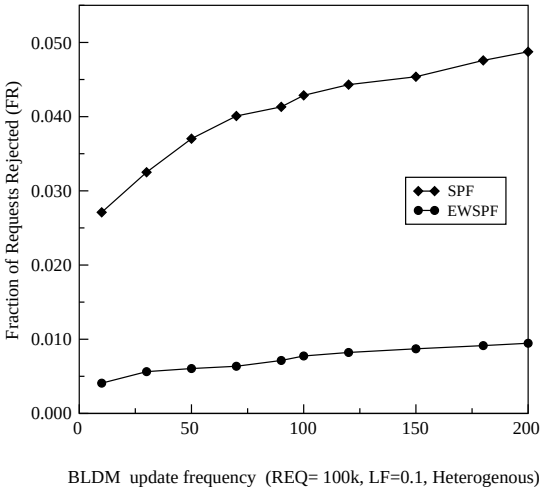


Fig. 17. Impact of update period on rejection fraction for EWSPF ($\boldsymbol{BM}$ update model)

of 1.25 for EWSPF). However, with EWSPF the deterioration of performance is much slower and for small update periods performance is quite acceptable.

7 Conclusions

In this paper we addressed the problem of distributed routing of bandwidth-guaranteed paths in generic label-switched networks with restoration. We showed that approaches to this problem that use only three variables per link l, namely: $\boldsymbol{F}$, the load induced by the primary paths, $\boldsymbol{G}$, the load induced by the backup paths, and $\boldsymbol{R}$, the residual bandwidth, suffer from pessimistic link selection during backup routing and ambiguity in the decision on the amount of bandwidth to release during path termination. We proposed a new form of state information called Backup Load Distribution (BLD) matrix $\boldsymbol{BM}$ that captures the distribution of primary load backed up on other links in the network for each link. For a fixed-sized network, this matrix is of constant size and the overhead incurred in disseminating it does not grow with the number of active paths.

We proposed two new algorithms: *Enhanced Widest Shortest Path First (EWSPF)* and *Enumeration Widest Shortest Path First (ENUM-WSPF)* that use this BLD matrix. Both use pre-computation in conjunction with the Widest Shortest Path First (WSPF) algorithm and run in bounded time. Also, they can be used for any label-switching networks, such as wavelength-switching optical networks as well as packet networks such as MPLS or ATM networks.

Our simulation results for sample topologies show 30–50% reduction in rejected requests and 30–40% savings in total bandwidth used for backup connections. We also show that although the performance of our routing schemes is sensitive to the frequency of BLD matrix updates, performance degradation resulting from stale state information is insignificant for the practical range of update periods.

References

[1] George Apostolopoulos, Roch Guérin, Sanjay Kamat, and Satish K. Tripathi. Quality of service routing: A performance perspective. In *Proceedings of ACM SIGCOMM*, Vancouver, BC, Canada, September 1998.

[2] George Apostolopoulos, Roch Guérin, Sanjay Kamat, Ariel Orda, Tony Przygienda, and Doug Williams. QoS routing mechanisms and OSPF extensions. RFC 2676, Internet Engineering Task Force, August 1999.

[3] Bruce S. Davie and Yakhov Rekhter. *MPLS Technology and Applications*. Morgan Kaufmann, San Francisco, CA, USA, 2000.

[4] Murali Kodialam and T. V. Lakshman. Dynamic routing of bandwidth guaranteed paths with restoration. In *Proceedings of IEEE INFOCOM*, Tel-Aviv, Israel, 2000.

[5] Samphel Norden, Milind M. Buddhikot, Marcel Waldvogel, and Subhash Suri. Routing bandwidth guaranteed paths with restoration in label switched networks. In *Proceedings of IEEE International Conference on Network Protocols (ICNP 2001)*, pages 71–79, Riverside, CA, USA, November 2001.

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

[7] Li Li, Milind M. Buddhikot, Chandra Chekuri, and Katherine Guo. Routing bandwidth guaranteed paths with local restoration in label switched networks. In *Proceedings of IEEE International Conference on Network Protocols (ICNP '02)*, pages 110–120, Paris, France, November 2002.

[8] George Apostolopoulos, Roch Guérin, and Sanjay Kamat. Implementation and performance measurements of QoS routing extensions to OSPF. In *Proceedings of IEEE INFOCOM*, pages 680–688, 1999.

[9] Tsong-Ho Wu. *Fiber Network Service Survivability*. Artech House, Inc., Norwood, MA, USA, 1992.

[10] Dongyun Zhou and Suresh Subramaniam. Survivability in optical networks. *IEEE Network*, 14(6):16–23, November/December 2000.

[11] Murali Kodialam and T. V. Lakshman. Minimum interference routing with applications to MPLS traffic engineering. In *Proceedings of IEEE INFOCOM*, Tel-Aviv, Israel, March 2000.

[12] Subhash Suri, Marcel Waldvogel, Daniel Bauer, and Priyank Ramesh Warkhede. Profile-based routing and traffic engineering. *Computer Communications*, 26(4):351–365, March 2003.

[13] Daniel Bauer. A new minimum-interference routing algorithm based on flow maximization. *IEE Electronics Letters*, 38(8):364–365, April 2002.

[14] John Moy. OSPF version 2. RFC 2328, Internet Engineering Task Force, April 1998.

[15] Daniel O. Awduche, Lou Berger, Der-Hwa Gan, Tony Li, Vijay Srinivasan, and George Swallow. RSVP-TE: Extensions to RSVP for LSP tunnels. RFC 3209, Internet Engineering Task Force, December 2001.

[16] Eric C. Rosen, Arun Viswanathan, and Ross Callon. Multiprotocol label switching architecture. RFC 3031, Internet Engineering Task Force, January 2001.

[17] Mark de Berg, Marc van Kreveld, Mark Overmars, and Otfried Schwarzkopf. *Computational Geometry: Algorithms and Applications*. Springer, Berlin/Heidelberg, Germany, 1997.

[18] Hongzhou Ma, Inderjeet Sing, and Jonathan S. Turner. Constraint based design of ATM networks, an experimental study. Technical Report WUCS-97-15, Department of Computer Science, Washington University in St. Louis, St. Louis, MO, USA, 1997.

Author Biographies

Samphel Norden received the B.S. (1997) from the Indian Institute of Technology, Madras, and the M.S. (1999) and Doctor of Science (D.Sc.) (2002) degrees in computer science from Washington University in St. Louis. He is currently a Member of Technical Staff (MTS) in the Center for Networking Research in Lucent Bell Laboratories. His research interests include mobile networking, denial-of-service detection and prevention, inter-domain QoS routing, overlay networks and wireless security.

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.

Marcel Waldvogel joined IBM Research, Zurich Research Laboratory in 2001 after holding faculty position at Washington University in St. Louis. He graduated from the Swiss Federal Institute of Technology (ETH Zurich), where he received a Diploma degree in computer science and a Ph.D. in electrical engineering. He is a member of the ACM and a senior member of the IEEE and has published over 40 papers on high-speed networking, multimedia communications, network security, overlays, and distributed storage networks.

Subhash Suri is a Professor of Computer Science at the University of California, Santa Barbara. Prior to joining UCSB, he was an Associate Professor at Washington University (1994–2000) and a Member of Technical Staff at Bellcore (1987–1994). He received a Ph.D. in computer science from the Johns Hopkins University in 1987. Professor Suri has published over 80 research papers in design and analysis of algorithms, Internet commerce, computational geometry, computer networking, combinatorial optimization, and graph theory. He serves on the editorial board of *Computational Geometry*, and has served on numerous panels and symposium program committees. He has been awarded several research grants from the National Science Foundation.