Design of Shared Mesh Restoration Schemes with Traffic Load Balancing Constraint

Hassan Naser  
Department of Software Engineering  
Lakehead University  
Thunder Bay, Ontario, Canada P7B 5E1  
hnaser@lakeheadu.ca

Ming Gong  
Department of Electrical and Computer Engineering  
University of Waterloo  
Waterloo, Ontario, Canada N2L 3G1  
mjong@uwaterloo.ca

Abstract—This paper presents two novel shared mesh restoration schemes for survivable transport networks. First, we present a two-step Load Balancing Pool Sharing (LBPS) scheme and show how this scheme can be used to balance the loads on the network links while still minimizing the reserved working and backup capacity in the network. Then, we propose an Iterative Load Balancing Pool Sharing (ILBPS) algorithm, designed to eliminate the trap-topology problem associated with all two-step mesh restoration algorithms (including LBPS) while still achieving the LBPS algorithm's goals. We compare the capacity-usage, load balancing, and computation complexity performances of these algorithms with two representative algorithms. We show that with the proposed schemes, the network traffic can similarly, or more evenly, be distributed among network links than with the other schemes, at lower computation cost.

Keywords: routing, quality of service, survivable network, network design and planning

I. INTRODUCTION

Survivable networks have the capability to survive from the events of network components failures. The resilience mechanisms in these networks protect and restore the impaired communication paths by using spare capacity. Many powerful dynamic protection and restoration algorithms have been developed for networks with different topology configurations, such as mesh and ring [1][2]. Mesh restoration refers to a class of restoration techniques that the source and destination nodes of the traffic traversing a failed link will reroute the traffic over one or a set of replacement paths (backup paths) between these nodes. In shared mesh restoration, the spare (backup) capacity is shared among different connections. The primary paths of these connections must be failure disjoint, so that no single failure can put out of service more than one connection at one time. Previous research studies have shown that shared mesh restoration is the most promising technique for saving the spare capacity while still achieving full restoration for any single network link failure [2].

In addition to the high usefulness of achieving this capacity goal, it should be emphasized that achieving traffic load balancing is of fundamental importance in communications networks, given that ensuring even workload distribution helps to eliminate congestions on network links. In this paper, load balancing refers to the process of distributing the network traffic evenly amongst network links so that no single link is overwhelmed.

In recent years, a plethora of shared mesh restoration algorithms have been proposed to provide low-cost and readily available restoration alternatives to SONET/SDH ring and linear protection. However, selecting restoration paths to maximize capacity sharing typically increases restoration path lengths, leading to longer restoration times [3]. It can also cause some links to be heavily shared (loaded) while other links are not. This contradicts the principle of load balancing which has as its ultimate goal to distribute the network load evenly amongst network links for congestion control. Therefore, a major challenge is to search for mesh restoration algorithms that can distribute capacity on network links according to load balancing specifications and allocate spare capacity on links according to survivability requirements. Consequently, development of novel integrated strategies that facilitate load balancing and survivability algorithms is of significant practical interest and is the primary focus of this study.

In this paper, we first present a novel integrated algorithm for load balancing and shared restoration, referred to as Load Balancing Pool Sharing (LBPS). The LBPS scheme aims to achieve the following two goals simultaneously: (a) minimize the total reserved capacity in the network by allowing backup paths for multiple connections (demands) share common spare capacities on backup links, and (b) distribute the network traffic evenly amongst network links.

The LBPS algorithm belongs to a class of mesh restoration algorithms known as two-step heuristic algorithms. With the two-step routing algorithms, the working and backup paths for every demand are computed independently in two steps. In step 1, the working path is computed, whereas in step 2 the selected working path is used to compute a link-disjoint backup path. One major drawback of all two-step routing algorithms (including LBPS) is that these algorithms suffer from a problem known as trap-topology, where the algorithm cannot find a pair of link-disjoint paths between two nodes even though diverse paths between these nodes actually exist on the topology [1]. For example, in the network topology shown in Fig. 1.(b), the LBPS scheme can potentially compute the working path A-B-C-Z for a demand between nodes A and Z. This pre-selected working path will not have a diverse backup path.
even though two diverse paths A-D-E-C-Z and A-B-F-G-Z exist between nodes A and Z in the network.

In order to address the trap topology problem associated with the LBPS scheme, we will introduce a novel iterative algorithm called Iterative Load Balancing Pool Sharing (ILBPS), in this paper. The ILBPS scheme adopts a simultaneous diverse paths computation strategy, similar to the one used in Suurballe’s algorithm [1]. However, the main difference is that the Suurballe’s algorithm uses the same link-cost function to compute the first and second paths, whereas the ILBPS does not. By selecting different link costs for the second path (than those used for the first path), the ILBPS scheme can achieve its backup bandwidth sharing objective.

Using simulations, we study the proposed LBPS and ILBPS algorithms on an existing North-American transport network and compare their capacity, load-balancing, and complexity performances with two representative shared mesh restoration algorithms. These are called: Routing with Load Balancing Heuristics (RLBH) which is a load-balancing shared mesh restoration algorithm [8]; and Iterative Two-Step Approach (ITSA) which has been reported as an “optimal” algorithm in the literature [5]. We will show that the LBPS and ILBPS algorithms outperform the RLBH algorithm in terms of the capacity and load-balancing performances. We will also show that the proposed algorithms achieve the same capacity and load-balancing performances as the ITSA scheme, however at much lower computation cost.

Section II provides a survey of related research on shared mesh restoration schemes and load balancing techniques. Sections III and IV introduce the LBPS and ILBPS algorithms, respectively. Section V presents the simulation results, and Section VI summarizes this paper.

II. RELATED RESEARCH

In [5], an Iterative Two-Step Approach (ITSA) is introduced, which implements an iterative rule to compute a pair of link-disjoint working and backup paths for a given demand. In each iteration, a two-step path computation approach is used to compute a new pair of working and backup paths. The sum of the costs of the two paths is used as the criterion to select the optimal pair of paths. The ITSA algorithm executes in $K$ iterations. It uses Yen's algorithm to compute $K$ shortest paths between the source and destination nodes of the given demand. Yen's algorithm is a classical algorithm for ranking the $K$ shortest loopless paths between a pair of nodes in a network [5]. In each iteration $k$, the $k$th shortest path is used as the potential working path, and a link-disjoint backup path is computed for the working path. The sum of the costs of the two paths found in this iteration is then compared with the sum found in the previous iteration. If the new sum is less than the old, the new pair of paths is accepted and the old pair is discarded. At termination, the ITSA algorithm yields the pair of link-disjoint paths with the least total cost.

Reference [6] has proposed a load balancing routing algorithm that can be used to compute a path between any given pair of nodes in the network. This algorithm cannot however be used in mesh restoration networks where a pair of link-disjoint paths must be computed between any two nodes. In [7], a load balancing routing algorithm has been introduced which computes a pair of link-disjoint paths between a given pair of nodes. However, the algorithm does not incorporate backup bandwidth sharing, and therefore is suitable for dedicated mesh restoration only.

In [8], a two-step shared mesh restoration algorithm called Routing with Load Balancing Heuristics (RLBH) was presented. The algorithm employs a threshold-based load balancing mechanism to select candidate links and to avoid using heavily loaded links during the working and backup path computations. For working path computation, if the amount of free bandwidth in a link is less than a threshold, the link becomes critical and the cost of the link is set to a large number. For backup path computation, the cost of a link is set to a large number if the link is heavily loaded or if it does not have high chance of containing a sharable backup bandwidth to restore the new demand. To measure the later quantity, the number of working paths that already supported by the shared backup bandwidth on this link is compared against another threshold (called venture index). A backup link with supported ratio larger than the venture index is considered not able to support any more working paths; therefore such a link is not considered as a backup link candidate for the new demand.

III. LOAD BALANCING POOL SHARING (LBPS) SCHEME

The LBPS scheme is a non-iterative shared mesh restoration algorithm that computes a pair of link-disjoint working and backup paths for a demand $r$ in two steps. In step 1, the working path is computed, whereas in step 2 a link-disjoint backup path is computed. The LBPS scheme adopts a cost assignment strategy based on the load balancing technique described below to assign link-costs for the working and backup path computations.

Let us denote $b$ the amount of bandwidth requested by the newly arrived demand $r$, $M_{i}$ the total capacity on link $i$, $A_{i}$ the available capacity on link $i$, $W_{i}$ the total reserved working bandwidth on link $i$, and $B_{i}$ the total reserved backup bandwidth on link $i$ ($M_{i} = W_{i} + B_{i} + A_{i}$).

In step 1, the LBPS algorithm computes a working path for demand $r$. The link cost function for working path computation is designed to achieve two goals: 1) to seek for the shortest path that traverses through links with sufficient bandwidth for demand $r$, and 2) to minimize the likelihood of using links that are heavily loaded. To achieve the second goal, we define a measure of load balance on link $i$ as:

$$l_{i} = \frac{\alpha_{1}W_{i} + \alpha_{2}B_{i}}{M_{i}}$$ (1)

Weights $\alpha_{1}$ and $\alpha_{2}$ are small non-negative tunable parameters.
Essentially, $l_i$ measures the ratio between the weighted sum of the allocated working and backup capacities and the total capacity on link $i$. Some networks use the backup bandwidth to carry low priority traffic, or extra traffic. The low-priority traffic is preempted when the protection switching occurs. In these networks, different values for $\alpha_1$ and $\alpha_2$ can be used to control the effect of the allocated working and backup capacities on the load factor $l_i$. However, when a value of 1 is used for $\alpha_1$ and $\alpha_2$, $l_i$ is simply the fraction of the total capacity used on link $i$.

We denote $C_W(i)$ the cost of link $i$ for the purpose of working path computation. The value for $C_W(i)$ can be obtained as:

$$C_W(i) = \begin{cases} \infty & b > A_i \\ 1 + l_i & b \leq A_i \end{cases}$$

(2)

$C_W(i)$ is set to $(1 + l_i)$ if link $i$ has enough available capacity for demand $r$, otherwise, it is set to infinite. Condition 2 in the above equation ensures that if two candidate links $i$ and $j$ have enough available capacity to accommodate demand $r$ (i.e. if $b \leq A_i$ and $b \leq A_j$), the link is favored by the path computation algorithm that has the lower load factor. A least-cost algorithm (e.g. Dijkstra’s algorithm) is used with the above cost assignment in order to compute a working path for demand $r$.

In step 2, the LBPS algorithm computes a link-disjoint backup path for the working path found in step 1. The LBPS scheme uses a pool sharing process that places the backup bandwidth on each link into a common pool. Different demands can share the backup bandwidth in the pool if their working paths are link-disjoint. This ensures that the failed connections from any single link failure can be fully restored on the backup link. The backup bandwidth reserved on each link is recorded in the following matrix [4]:

$$\Omega = \begin{bmatrix}
0 & k_{12} & k_{13} & \ldots & k_{1J} \\
k_{21} & 0 & k_{23} & \ldots & k_{2J} \\
k_{31} & k_{32} & 0 & \ldots & k_{3J} \\
\vdots & \vdots & \vdots & \ddots & \vdots \\
k_{J1} & k_{J2} & k_{J3} & \ldots & 0
\end{bmatrix}$$

(3)

Element $k_{ij}$ is the amount of backup bandwidth needed on link $j$ if link $i$ fails ($1 \leq i, j \leq J$), where $J$ is the number of links in the network. The total amount of backup bandwidth needed (reserved) on link $j$ ($B_j$) is indeed the maximum of all elements in column $j$:

$$B_j = \max_{i \in \Omega} k_{ij}$$

(4)

We denote $S_W(r)$ the set of links along the working path of demand $r$, and $T_j$ the maximum amount of backup bandwidth required on link $j$ if a link in $S_W(r)$ fails. $T_j$ can be computed from the following formula:

$$T_j = b + \max_{i \in S_W(r)} [k_{ij}]$$

(5)

In backup path computation, the LBPS scheme pursues the following two goals: 1) avoid using heavily loaded links as in working path computation; and 2) use links that have high enough level of sharable backup bandwidth. Both goals can be achieved by using a properly designed link cost function that allows the algorithm to use the sharable backup bandwidth ($B_j$) and the current load ($l_j$) on every link $j$ to guide the backup path computation. The link cost $C_B(j)$ for backup path computation is thus defined as:

$$C_B(j) = \begin{cases} \infty & j \in S_W(r) \\ \frac{T_j - B_j}{b} + l_j & 0 < T_j - B_j \leq A_j \\ \infty & \text{otherwise} \end{cases}$$

(6)

Condition 1 ensures that the backup path will be link-disjoint from the corresponding working path computed in step 1. Condition 2 ensures that if two links $j$ and $k$ have adequate shared backup bandwidth, the link with less load factor is favored by the path computation algorithm over the other link. With condition 3, the cost of link $j$ is set to $l_j + (T_j - B_j) / b$, where $(T_j - B_j)$ is the amount of additional backup bandwidth that must be reserved to restore demand $r$, and if this additional bandwidth is available (i.e. if $0 < T_j - B_j \leq A_j$). The term $l_j$ has been added to the third condition for similar reason as in condition 2: it allows to some degree a load-based selection of a backup link if on this link $T_j - B_j$ units of bandwidth must be reserved to restore demand $r$. If the additional bandwidth is not available, the cost of link $j$ is set to infinite. Dijkstra’s algorithm is used with the cost assignment (6) in order to compute a link-disjoint backup path for the working path computed in step 1. We denote $S_B(r)$ the set of links along the backup path of demand $r$.

IV. ITERATIVE LOAD BALANCING POOL SHARING (ILBPS)

The Iterative Load Balancing Pool Sharing (ILBPS) algorithm is intended to achieve goals (a) and (b) set for the LBPS scheme in Section 1. However, it achieves these goals by adopting a different mechanism based on an iterative process of searching for link-disjoint paths for a demand. The iterative process is designed in order to eliminate the trap topology associated with the non-iterative two-step path computation algorithms, such as the LBPS scheme.

In each iteration, the ILBPS scheme attempts to find the shortest pair of link-disjoint working and shared backup paths between the given source and destination nodes of the demand. If the pair of the paths is found the algorithm terminates successfully. If the pair is not found, it could be due to a link (or links) causing the trap-topology between the source and destination nodes. We call such a link as “trap-link”. One example of the trap-link is the link BC for the demand between
nodes A and Z in the topology shown in Fig. 1.(b).

The ILBPS algorithm then employs a strategy, similar to the one used in Suurballe’s algorithm to identify the link (or links) that has caused the trap topology in the current iteration. If a trap-link is found, it will be eliminated from the topology in the next iteration of the optimization process. The formal description of the ILBPS algorithm is given below.

Let us denote $T$ as the set of trap-links that are currently known. One iteration of the ILBPS algorithm adds one or more new trap-links to $T$. The algorithm has four steps; steps 2 to 4 are repeated until a pair of link-disjoint paths has been found:

1) **[Initialization]**: $T = \emptyset$.
2) **[Compute Working Path]**: Run Dijkstra’s algorithm to compute a working path $S_W(r)$ by using the following cost function for every link $i$:

$$C_w(i) = \begin{cases} \infty & i \in T \\ b & b > A_i \\ 1 + l_i & b \leq A_i \end{cases}$$  

(7)

If the working path is not found, the demand is blocked; otherwise a backup path is computed by going to step 3.

3) **[Compute Backup Path]**: Run Dijkstra’s algorithm again to find a link-disjoint shared backup path $S_B(r)$ by using the link cost function (6). If the backup path is found, the algorithm terminates successfully and returns the pair of paths. Otherwise, the algorithm proceeds to the next step to detect any trap-link that might have blocked the computation of the backup path.

4) **[Find Trap-links]**: Run Dijkstra’s algorithm to find a temporary backup path $S'_B(r)$ which can possibly have overlaps with the working path $S_W(r)$, by using the following cost function:

$$C'_b(j) = \begin{cases} Z_j & j \in S_W(r) \\ l_j & T_j \leq B_j \\ \frac{T_j - B_j}{b} + l_j & 0 < T_j - B_j \leq A_j \\ \infty & \text{otherwise} \end{cases}$$  

(8)

where,

$$Z_j = \begin{cases} \infty & \text{To_Destination} \\ -\delta & \text{To_Source} \end{cases}$$  

(9)

If $S'_B(r)$ is found, the trap-links will be the interlacing links of the sets $S_W(r)$ and $S'_B(r)$. Add these links to $T$, and go back to step 2. If $S'_B(r)$ is not found, the demand is blocked.

Note that condition 1 of the cost function (7) is designed to avoid using the trap-links found in the previous iteration. These links are indeed the links that have met condition 1 of the cost function (8). In (9), every link along the path $S_W(r)$ is replaced by two unidirectional links (arcs); one directed toward the destination (To_Destination), and the other toward the source (To_Source). The cost of the arc toward the destination is set to infinite ($Z_j = \infty$), whereas the cost of the arc toward the source is set to a very small negative number ($Z_j = -\delta$). The costs of all other links not in $S_W(r)$ are set to either $l_j$ or $l_j + (T_j - B_j) / b$, depending on how much shared backup bandwidth already reserved on these links.

V. SIMULATION RESULTS

We have used simulation technique in C++ to test the LBPS, and ILBPS schemes and also to compare their performances with the RLBH and ITSA restoration schemes. We define the load on link $k$ (denoted by $L_k$) to be the sum of the allocated working and backup capacities ($W_k + B_k$) divided by the total capacity of the link ($M_k$). We use the standard deviation of the sample $L_k$ (denoted by $S$) as a measure of load distribution (balancing) amongst network links. The standard deviation $S$ is calculated by taking the square root of the sum of $(L_k - L)^2 / J$ over all possible values of $k$ (1…J), where $L'$ is the mean of the sample $L_k$. In terms of load balancing, a diverse paths computation scheme is said to outperform other candidate schemes if it achieves the smallest $S$.

We use the average reserved capacity per accepted demand ($V$) in order to evaluate the capacity performance of the mesh restoration schemes presented in this paper. We compute $V$ as the sum of $(W_k + B_k)$ over all values of $k$ divided by $n_d$, where $n_d$ is the number of accepted demands in the network.

We used the National Science Foundation (NSF) network, which is a representative North American backbone network, to test the criteria defined above. The topology of the NSF network is illustrated in Fig. 1.(a) with 16 nodes interconnected by 25 bidirectional links in mesh form.

In this paper, we assume that all links in the network have equal capacity of $M_k = 2.4$ Gbps. We generated six demand matrices for the NSF network; with the total number of demands in each matrix being 50 / 80 / 100 / 150 / 200 / 250. For every demand, the source and destination nodes were generated randomly. In our simulations, all demands requested identical amount of bandwidth $b = 100$ Mbps from the network.

We set the values of the parameters ($\alpha_1, \alpha_2, \delta$) to (0.1, 0.01, 0.001), respectively. These values are the best case scenarios. We studied the effect of the above parameters on $S$ and $V$ by varying their values used in the experiments. The results were very similar to the results shown in this paper. The maximum number of iterations for the ITSA algorithm is $K = 50$, which has been shown to be an optimal number in [5].

The standard deviation of link load ($S$) for all the four restoration schemes is shown in Fig. 2, as a function of the number of accepted demands in the network. The figure shows that the standard deviation $S$ for the LBPS and ILBPS schemes are comparable to the corresponding parameter for the ITSA scheme. They are also lower than the corresponding parameter for the RLBH scheme when the load is not high. This means that with the LBPS and ILBPS schemes, the load assignment on every link is more balanced than the RLBH scheme at low to medium network load.
The demand-blocking ratio is less than that of the RLBH algorithm. This extension indicates that by using the LBPS or ILBPS algorithms, the ranges of the plots using the RLBH scheme are extended about 160 accepted demands, which is wider than the spreading more toward the high accepted-demand side, and spread over insufficient capacity on network links. The tail of the plots corresponding to the LBPS and ILBPS schemes are extended (in steps 2, 3, 4). We have found that the number of iterations of the ILBPS algorithm is not more than 3 in our simulation experiments. From these results, it can be concluded that the LBPH and ILBPH schemes yield the same capacity and load balancing performances as the ITSA scheme, but at significantly lower computation cost.

VI. CONCLUSION

We have introduced two novel shared mesh restoration algorithms, called Load Balancing Pool Sharing (LBPS), and Iterative Load Balancing Pool Sharing (ILBPS) which avoids the trap topology. We compared their performances in terms of traffic load balancing, total reserved capacity, and computation complexity with two existing shared mesh restoration schemes: RLBH and ITSA algorithms. We have shown that the LBPS and ILBPS schemes are more optimized than the RLBH scheme in terms of capacity usage and load balancing (at low to medium load). Compared with the ITSA algorithm, the LBPS and ILBPS schemes yield similar capacity performances at much lower computation cost.

REFERENCE