Failure-Oriented Path Restoration Algorithm for Survivable Networks

William Lau and Sanjay Jha
School of Computer Science and Engineering,
The University of NSW, Kensington Sydney 2052 Australia
wlau,sjha@cse.unsw.edu.au (Telephone: 0293854354, 0293856471)

UNSW-CSE-TR-0323

THE UNIVERSITY OF
NEW SOUTH WALES

SYDNEY · AUSTRALIA

Abstract

Connection-oriented networks such as MPLS and GMPLS offer network providers the mechanisms to deliver a high level of service quality to their clients. One critical factor in determining the service quality is the rate of availability or what some call the *up-time* of the connection. A common approach for provisioning high availability with shorter restoration times is to use pre-calculated backup paths that are used when the normal service paths fail. The challenge in this approach is to allocate the minimal total spare capacity required by the backup paths. One restoration strategy that aims to minimize spare capacity is based on failure-oriented reconfiguration (FORC), where a backup path is calculated for each possible scenario that affects the working service path. Linear and integer programming formulations can be made to find optimal solutions but do not run in polynomial time. An existing heuristic algorithm was proposed to reduce the computation time but it also does not run in polynomial time.

In this paper, a new polynomial-time approximation algorithm called Service Path Local Optimization (SPLO) is proposed. SPLO is shown to perform better than the existing approximations for FORC. SPLO is designed for online computation where only one request is computed at any one time, and the decision making does not depend on future requests. The polynomial-time and online nature of the algorithm make SPLO suitable for use in real-time on-demand path request applications. Further, the potential for SPLO as an algorithm in traffic engineering applications is investigated by looking at the performance impact when source-destination-based traffic aggregation is applied. The results show that spare capacity requirement for SPLO is degraded by up to 5% only.

This paper also introduces a new concept called path intermix where the service path's allocated bandwidth can be used by the backup paths protecting that particular service path. The result shows that path-intermix reduces the lengths of backup paths and can reduce spare capacity by up to 4% for single node failures.

I. INTRODUCTION

High availability of network connections is a key component of any Service Level Agreement (SLA) in the current business of quality-based network services such as leased-line services or Virtual Private Networks (VPNs). Network providers are faced with the challenge of reducing the risk of violating service agreements while at the same time maximizing network efficiency. If the level of availability is not satisfied then a significant loss in revenue and penalty payouts are required. On the other hand, improving network efficiency will increase the number of services that the network infrastructure can support, hence increasing revenue.

Connection-oriented technologies such as MPLS [1], [2], [3] and GMPLS [4], [5], [6] are fast becoming the technologies of choice to provision such services. Traffic engineering [7] can improve network utilization and allow more sophisticated restoration strategies to be deployed. The most basic type of strategy in MPLS to ensure high availability is 1+1 path protection [8] where a backup path is pre-established along with the service path. The backup path is necessarily disjoint [1] from the service path and must have the same bandwidth capacity if full traffic restoration is desired. In 1+1 path protection, both the service path and the backup path are assumed to be used simultaneously for traffic. A common technique is to send redundant traffic on the backup path to reduce the time required for restoring normal traffic operation. An alternative is to use 1:1 protection [8] where the backup path is not used unless the service path is known to have failed. Upon detecting failure of the service path, the traffic is switched to the backup path and then switched back to the service path once recovery is complete.

These two restoration strategies are commonly used to protect from single link or node failures. The unused nature of the backup path bandwidth in the 1:1 strategy leads to many proposals [9], [10], [11], [12], [13], [14] that aim to exploit this property to maximize network utilization. The term spare capacity refers to the total bandwidth allocated for backup paths in the network. For the rest of this paper, the measure for algorithm's performance is the effectiveness of the algorithm in reducing spare capacity unless explicitly specified otherwise.

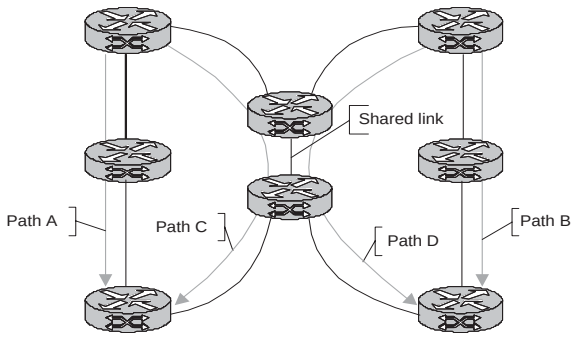

Fig. 1. Backup bandwidth sharing

A simple example that demonstrates the idea of spare capacity sharing is illustrated in Figure 1. Suppose that we have two service paths A and B that are coincidentally link disjointed. Both have corresponding backup paths, C and D respectively. The only requirement is that the service path and backup path pairs are link disjoint, thus it is possible for the two backup paths to go through some common links within the network, as in this case. Because both service paths are disjoint and we only expect to recover fully from a single link failure, the two backup paths are not expected to be used at the same time. Using this extra knowledge, the common links that C and D traverse are not required to reserve the sum of C and D's bandwidth. Only the maximum of C's or D's bandwidth must be reserved. Thus C and D can share overlapping backup bandwidth for common links.

One restoration strategy that exploits this sharing behavior is based on Failure-Oriented Re-Configuration [12] (FORC). FORC is a 1:N path protection scheme that aims to maximize network utilization and can be extended to support multiple links or nodes failures. The general idea is to establish a different backup path for every possible failure scenario that the network is designed to support. These backup paths are not used until a failure occurs, so it is possible to share spare capacity between different backup paths. This paper focuses mainly on single-link and single-node failures.

[1] The path can be link or node disjoint depending on the decision of the network provider.

Linear [12], [9], [15] and integer [10] programming formulations can be made to find optimal solutions but do not run in polynomial time. There is an existing heuristic algorithm (X-FORC) proposed by Xiong et al. [12] that reduces the computation time and finds a sub-optimal solution. However, it does not run in polynomial time. Further, the algorithm is based on fix-routes model where each source-destination requests choose a route from a set of pre-defined routes.

The first contribution of this paper is to modify the algorithm to run with dynamic route computation. The backup path is chosen from the least cost route for the current network resource state. The main purpose for allowing dynamic route is to give the algorithm the flexibility to find better routes and also provide a better basis to compare with other algorithms that dynamically compute routes.

The second contribution of this paper is a new algorithm for FORC backup path computation called Service Path Local Optimization (SPLO). The main advantages of SPLO are:

1) Performs better than X-FORC.
2) Runs in polynomial time.
3) Is an online FORC algorithm.
4) Has small performance impact under source-destination traffic aggregation.

"Online algorithm" indicates that each restoration request to protect a service path is computed on demand and has no prior knowledge of future requests. The decision is based on the current network resource state and the current restoration request. "Polynomial time" means that SPLO does fast request computation in a deterministic bounded-time. Therefore, SPLO is suitable for use in real-time, on-demand path computations. Traffic aggregation helps reduce the number of backup paths required in FORC restoration strategies. We will show that SPLO has a small impact in traffic aggregation situations, further demonstrating its scalability and its high potential for use in traffic engineering applications.

The final contribution of this paper is a new concept called path intermix. Path intermix occurs when the service path's allocated bandwidth can be used by the backup path protecting that particular service path. The result shows that path-intermix reduces the average backup paths length and can improve performance for single node failures by upto 4%.

The rest of this paper is as follows. Section 2 discusses related work on restoration strategies. Section 3 describes the existing heuristic algorithm in detail and introduces the proposed polynomial-time algorithm. In Section 4, path-intermix is introduced along with a discussion of the modifications required. Section 5 explains the simulation experiment setup while Section 6 analyzes the results. A simple benchmark to demonstrate run-time differences between the algorithms are given in Section 7. The paper then concludes in Section 8.

II. Related Work

Kodialam et al. [13] was the first to propose an online restoration strategy but it was based on the state-independent concept. In state-independent strategies, only one backup path is computed for each service path. The backup path must be able to recover from all possible single link (or node) failures. State-independent strategies are only suitable for single-failure recoveries. State-dependent strategies offer more flexibility on the types of failures the network provider can protect the service path from. However, the disadvantage of state-dependent strategies is that either the backup paths must be setup on demand or all the backup paths must be pre-established. On-demand setup of the backup path increases the restoration time, while pre-establishing every backup path can be infeasible because of the limited memory in the switch routers. However, we show in this paper that the proposed polynomial-time algorithm is not significantly sensitive to source-destination pair traffic aggregation. Therefore, the number of service paths can be significantly reduced, which also leads to a significant reduction in the number of backup paths that must be established. The consequence is that the proposed method makes pre-establishment of backup paths more feasible. Note that it is also possible to selectively pre-establish some frequently used backup paths or backup paths of high priority service paths. Another major difference is that Kodialam's approach was based on joint path computation and the heuristic algorithm does not run in polynomial time. The algorithm proposed in this paper is based on separate path computation where the service path is assumed to be pre-computed using Dijkstra's shortest path algorithm. Li et al. [14] proposed a polynomial time algorithm for Kodialam's online model based on separate path computation.

Liu et al. [16] proposed a progressive heuristic algorithm for the state-independent concept. The idea is to incrementally recompute backup paths once the network resource state has changed because of new service path requests.

Although the time complexity for each iteration is polynomial, the algorithm as a whole is non-polynomial and may take an exponential amount of time to converge to a local optima. The algorithm is designed to run in a distributed manner and its iterative nature pushes the network close to the local optimal state over time. However, the distributed algorithm may not converge to a stable state because of possible decisions that cause oscillations in path updates. Proof of convergence and stability was not provided in the paper.

Iraschko et al. [11] proposed an Integer Programming (IP) model that was based on the granularity of links within a span. This IP model assumed restricted fixed routes and was extended to include joint path computation. However, the paper did not propose any heuristic algorithms. The idea of stub release was also proposed in this paper where the bandwidth from the failed service path is released for use by the backup paths. Although performance is improved, the practicability of stub release is questionable. Stub release requires all the affected routers to know how much bandwidth to shift from service bandwidth to backup bandwidth in any given failure scenario. It also requires the network equipment to allow shifting of bandwidth on the fly. Another drawback is that it can complicate the recovery and reversion process. During recovery, it takes time for the service paths' traffic to be rerouted to the corresponding backup paths. During the rerouting time, the bandwidth stub released may still be used by the service path and can potentially delay the time for the traffic to flow back to normal. The reversion process is the rerouting of traffic back to the service path after the failure is fixed. If the service path's bandwidth is released, there is no guarantee that traffic will revert to normal behavior until all the traffic has been flushed from the backup paths that are used during the failure.

There are many other studies [15], [10], [9] that are based on the state-dependent concept but differ in the linear/integer programming optimization formulation and assumptions made. None of these works provide a polynomial-time or heuristic algorithms.

This paper extends the work on FORC from the proposal by Xiong et al. [17], [12]. FORC is based on the state-dependent concept where the state of the network determines the backup paths to use. Each failure scenario causes the network to move into a certain state that may include a single link failure, or a single node failure, or a mix of multiple failures. Given a predefined set of failures, FORC computes a set of backup paths and the minimal spare capacity required to fully recover from any of the failures specified. Xiong et al., formulated the problem into a Linear Programming (LP) model based on a set of fixed routes and also allowed bifurcation of flows. Backup paths were chosen as a set of predefined routes between source-destination pairs. This is a trade-off between optimality and the complexity of the LP model. Bifurcation occurs when a flow splits and the sub-flows follow different paths. The applicability of this assumption is questionable, but this LP model does provide a lower bound on performance for their heuristic algorithm, which is also based on fixed routes. Their heuristic algorithm also requires a post-processing step that involves a non-polynomial-time iteration loop. The new algorithm (SPLO) proposed in this paper does not put limitations on routes. It runs in polynomial time, and is also the first state-dependent algorithm designed for online computation.

In this paper, a new concept called *path intermix* is proposed which is similar to stub release except that the bandwidth is not released to the general backup pool, rather, only the backup paths protecting the service path can exploit that service path's allocated bandwidth. A similar technique was used implicitly by Bejerano et al [18] for path bridging. Their restoration strategy is to build bridges that connect subsections of the path hence they forced the reuse of the service path and puts restrictive constraints on the bridges. They focus mainly on the delay constraint problem and not the global network restoration problem. Thus sharing is only available between bridges belonging to a service path and not available between bridges of different service path. The path intermix proposed here is not to build bridges but to encourage use of freely available service path bandwidth if it is found to be useful. Herberg et al [10] proposed a restoration strategy based on line restoration where the objective is to restore connection around the failure using path segment. Path segment refer to a partial path that starts at the service path on the node preceding the failure and joins back the service path at the node succeeding the failure. However, Iraschko et al [11] and Xiong et al [17] observed that line restoration is inferior to path restoration because it does not distribute spare capacity as much as path restoration over the network. The result for path-intermix shows that a soft approach that encourages reuse of service path bandwidth reduces the average backup paths length and can reduce spare capacity by up to 4% for single node failures.

III. FORC ALGORITHMS

The heuristic algorithm proposed by Xiong et al. [12] is described in Algorithm 1. Firstly, we assume that the set of service paths for the set of requests is pre-computed and is an input to the algorithm. The network graph $G(V, E)$ is another input that contains the set of vertexes V and the set of links E. Each link is assumed to keep track of the resource state which includes the available bandwidth A_l, and total spare capacity allocated for restoration C_l. Another important input to the algorithm is the set of failure scenarios FS. If the network providers want to protect from single link failure then each possible single link failure will be represented by one failure scenario. Single node failure is similar except multiple link failures are associated with each scenario. Each failure scenario must also keep track of the spare capacity it uses on each link in the network (C_l^s). Given these inputs, the algorithm's objective is to compute a set of backup paths protecting the service paths from the defined failure scenarios while at the same minimize the spare capacity requirement. The output of the algorithm is the set of backup paths and the network graph containing the updated resource state.

Algorithm 1 consists of two sections. In the first section, a loop iterates through each failure scenario and computes backup paths for the service paths affected by the failure. The second section consists of an iterative loop where for each link in the network, the failure that causes the worst-case spare capacity usage is selected. For this failure, the backup paths that go through the link in question are recomputed with the link removed from the network. If the new backup path costs less[2] than the original backup path then this new backup path is used and the spare capacity for the affected links is updated. This continues for all backup paths in the failure scenario, and for all worst-case failure scenarios on each link. If at least one backup path is updated during an iteration, then another iteration is made. Otherwise, the algorithm ends. We call the second section the post-processing step. Note that the post-processing part of algorithm presented here is simplified to assume that only one failure scenario cause the worst case. The algorithm used in the experiments do not make this assumption.

The terms route and path are used interchangeably with the same meaning in this paper. In Xiong's original algorithm (X-FORC), a set of fixed routes (R_π^s where π denotes the service path and s denotes the failure scenario) is given that the backup paths can take between a source-destination pair in a given failure scenario. The set of requests is denoted by Π and the bandwidth requested is given by b_p. The cost of taking a given route r in a failure s is calculated using the equation below.

$$cost(r) = \sum_{l \in E} \delta_{rl} w_l \max(b_p - Q_l^s) \tag{1}$$

If link l is used in route r then δ_{rl} is equal to one, otherwise it is equal to zero. The weight for using link l is denoted by w_l. The idea is to share the current spare (backup) capacity (C_l) on each link between different failure scenarios because they are assumed to be independent of each other. That is, only one failure is assumed to occur at any given time. The backup paths in the same failure scenario cannot be shared between themselves because they are using the spare capacity at the same time. Therefore the spare capacity available for use by the current backup path to be computed is $Q_l^s = \max(C_l - C_l^s, 0)$. The cost for using a given route r will be the bandwidth required by the service path (b_p) less the spare capacity that is still available. The route with the minimum cost is taken as the backup path.

The first section can be completed in polynomial-time $O(|FS||P||R_\pi^s|)$. The value $|R_\pi^s|$ is the number of fixed routes for a source-destination pair under a single-failure scenario. The second section is an iterative loop that does not have a polynomial-time bound, and each iteration requires $O(|E||BP|(|R_\pi^s| + |FS|))$.

As the first contribution of this paper, this algorithm is reformulated so that it will run without the fixed-route limitation. This is done through manipulation of the link cost in the network graph. Algorithm 2 describes the new, modified X-FORC algorithm. The main difference is the way we find the backup paths in both sections of the algorithm. To find a backup path, the failed links in the current failure scenario are removed from the graph first. Then the cost for all links in the graph must be recomputed using τ_l^s:

[2]We do not consider the case of equality as it can cause oscillations.

Data : Network Graph: $G(V, E)$; Failure scenario set: FS; Request set: Π; set of fixed routes: R; Service Path set: P
; **Result** : New Network Graph: $G(V, E)$; Backup Path set: BP
; Initialize $BP = \emptyset$;
for *each* $s \in FS$ **do**
  Find the set of affected service paths: P_s;
  Initialize $C_l^s = 0, \forall l \in E$;
  for *each* $p \in P_s$ **do**
    Find the set of routes that a backup path can use: R_π^s;
    Find the route $r \in R_\pi^s$ with the least cost;
    Update $G(V, E)$ and C_l^s to use r as the backup path;
    Add r to BP;
  end
end
Initialize $improve = true$;
while $improve$ **do**
  $improve = false$;
  for *each* $l \in E$ **do**
    Find $s \in FS$ that causes the worst-case state;
    Find the set of backup paths used in s: BP_s;
    for *each* $bp \in BP_s$ *that uses link* l **do**
      Reverse the cost state of $G(V, E)$ when bp is removed from the network;
      Find the set of other fixed routes that the backup path can use: $R_\pi^s - \{r\}$;
      Find the route r with the least cost;
      if *cost of* $r <$ *cost of* bp **then**
        Update $G(V, E)$ and C_l^s to use r as the backup path;
        $BP = BP - \{bp\} + \{r\}$;
        $improve = true$;
      else
        Update $G(V, E)$ and C_l^s to use bp as the backup path;
      end
    end
  end
end

Algorithm 1: X-FORC Algorithm

$$\tau_l^s = \begin{cases} 0, & \text{if } C_l \geq C_l^s + b_p \\ w_l(b_p - C_l + C_l^s), & \text{if } C_l > C_l^s \text{ and } C_l - C_l^s < b_p \\ & \text{and } A_l \geq b_p - C_l + C_l^s \\ w_l b_p, & \text{if } C_l \leq C_l^s \text{ and } A_l \geq b_p \\ \infty, & \text{otherwise.} \end{cases}$$

The first case occurs when a link can be used for free because there is enough spare capacity already allocated on the link to accommodate the backup load in the failure scenario. The second case occurs when there is enough spare capacity on the link to accommodate the failure scenario's bandwidth requirement without the including the current request's bandwidth. However, there is some residual spare capacity available that can be used by the current request, so the cost for using this link is the difference between the requested bandwidth and the residual spare capacity. The third case is when no residual spare capacity exists, when the cost for using this link is the current request bandwidth requirement. The final case occurs if the link being considered is down or there is not enough available bandwidth (A_l) on this link to satisfy the request.

Once we have the new network graph, a shortest path algorithm is applied to find the backup path. The network resource state will require updating for the resources used by the backup path. This is done by going through each link used by the backup path with the following:

$$C_l^s = C_l^s + b_p \tag{2}$$
$$C_l = C_l + \tau_l^s \tag{3}$$

Data : Network Graph: $G(V, E)$; Failure scenario set: FS; Service Path set: P
; **Result** : New Network Graph: $G(V, E)$; Backup Path set: BP
; Initialize $BP = \emptyset$;
for *each* $s \in FS$ **do**
  Find the set of failed links E_s in the current failure scenario;
  Remove all failed links ($l \in E_s$) from $G(V, E)$;
  Initialize $C_l^s = 0, \forall l \in E$;
  Find the set of affected service paths P_s;
  for *each* $p \in P_s$ **do**
    Recompute the network links cost τ_l^s;
    Find the shortest path bp using the new graph;
    Update $G(V, E)$ and C_l^s to use bp as the backup path;
    Add bp to BP;
  end
  Insert links in E_s back into $G(V, E)$;
end
Initialize $improve = true$;
while $improve$ **do**
  $improve = false$;
  for *each* $l \in E$ **do**
    Find $s \in FS$ that causes the worst-case state;
    Find the set of backup paths used in s: BP_s;
    Remove all failed links ($l \in E_s$) from $G(V, E)$;
    for *each* $bp \in BP_s$ *that uses link* l **do**
      Reverse the cost state of $G(V, E)$ when bp is removed from network;
      Recompute the network links cost τ_l^s;
      Find the shortest path bp_{new} using the new graph;
      if *cost of* bp_{new} $<$ *cost of* bp **then**
        Update $G(V, E)$ and C_l^s to use bp_{new} as the backup path;
        $BP = BP - \{bp\} + \{bp_{new}\}$;
        $improve = true$;
      else
        Update $G(V, E)$ and C_l^s to use bp as the backup path;
      end
    end
    Insert failed links back into $G(V, E)$;
  end
end

Algorithm 2: New X-FORC Algorithm

X-FORC is an offline algorithm, so a restoration request consists of finding backup paths for a set of service paths. If X-FORC fails to find a backup path then it exits the computation. This is because it fails to find a feasible solution for the restoration request as a whole.

The complexity of the new X-FORC is now $O(|FS||P|(|E|+|V|\log|V|))$ for the first section, and each iteration of the second section runs in $O(|E||BP|(|E|^2+|V|\log|V|))$. The increase in time complexity comes from the need to apply the shortest path algorithm ($O(|V|\log|V|)$) and the need to remove the backup path from the network ($O(|E|^2)$). However, the modified version of the algorithm should reduce the spare capacity requirement more than the original X-FORC algorithm because it takes all possible routes into consideration.

The loop for computing the backup paths takes the failure scenario into consideration first and the best utilization of spare capacity is made between backup paths in the same failure scenario. This paper proposes a new approximation algorithm called Service Path Local Optimization (SPLO). In SPLO, we consider each service path separately and find the best backup paths to protect the whole service path. The nature of this algorithm makes it an online algorithm suitable for real-time on-demand path protection applications. The idea is that we try to optimize one service path at a time given the current network resource state. The SPLO algorithm is shown in Algorithm 3. We assume that a shortest path algorithm is used to compute the service path and then Algorithm 3 is executed to find the set of backup paths that cover the service path for any defined failure scenario.

The inputs to this algorithm is a service path (p) to protect, the current network graph and resource state ($G(V, E)$), and the set of failure scenarios to protect service path from. The output is then a set of backup paths that protects the service path, the updated network resource state, and the set of failure scenarios which also keeps track of G_l^s.

Data : Network Graph: $G(V, E)$; Failure scenario set: FS; Service Path: p
; **Result** : New Network Graph: $G(V, E)$; Backup Path set: BP; Failure scenario set: FS
; Initialize $BP = \emptyset$;
Find the set of failures FS_p affecting the Service Path;
for *each* $s \in FS_p$ **do**
    Find the set of failed links E_s in the current failure scenario;
    Remove all failed links $l \in E_s$ from $G(V, E)$;
    Recompute the network links cost τ_l^s.;
    Find the shortest path using the new network graph.;
    Update $G(V, E)$ and C_l^s to use this shortest path as the backup path;
    Add the backup path to BP;
    Insert failed links back into $G(V, E)$;
end

Algorithm 3: Online Approximation Algorithm

The logic behind Algorithm 3 is to find all the failure scenarios that will affect the service path. The next step is to compute a backup path for each affected failure scenario. The methods used for computing the network cost and updating the network graph after finding the backup path are the same as those described previously for Algorithm 2. The time complexity for SPLO is $O(|FS|(|E| + |V| \log |V|))$ per request. This complexity must be multiplied by a factor of $|P|$ to include all requests.

IV. Path Intermix

Path intermix is a new concept proposed in this paper. Path intermix encourages reuse of the bandwidth allocated to the service path. A particular service path's bandwidth cannot be used by backup paths belonging to other service paths, only the backup paths protecting that service path can exploit the service path's bandwidth. The word encourage is stressed here because there is no notion of forcing backup paths to reuse the bandwidth as seen in other works [10], [18]. The logic is to take into account this service bandwidth when recomputing the link cost for the backup path. To incorporate path intermix into X-FORC and SPLO, the formula for τ_l^s is modified.

$$\tau_l^s = \begin{cases} 0, & \text{if } l \in E_p \\ 0, & \text{if } C_l \geq C_l^s + b_p \\ w_l(b_p - C_l + C_l^s), & \text{if } C_l > C_l^s \text{ and } C_l - C_l^s < b_p \\ & \text{and } A_l \geq b_p - C_l + C_l^s \\ w_l b_p, & \text{if } C_l \leq C_l^s \text{ and } A_l \geq b_p \\ \infty, & \text{otherwise.} \end{cases}$$

An extra condition is added into τ_l^s, which states that the cost for using a link is zero if the link is part of the service path.

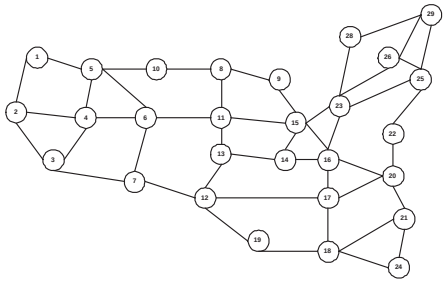

Fig. 2. Network1: US Long-Distance

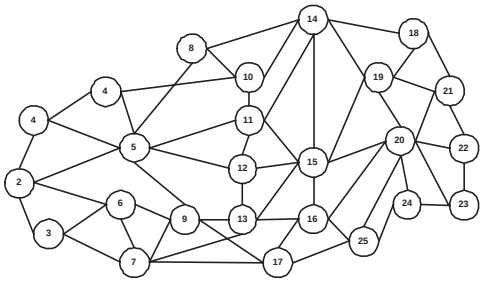

Fig. 3. Network2: Toronto Metropolitan

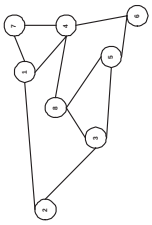

Fig. 4. Network3: Random Sparse Network

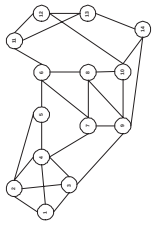

Fig. 5. Network4: Random Dense Network

After finding the backup path, the network must be updated. This update process must also be modified. For each link on the backup path, the resource state is updated by the following.

$$\text{If}(l \in E_p)\{C_l^s = C_l^s + b_p\} \tag{4}$$

$$C_l = C_l + \tau_l^s \tag{5}$$

The extra conditional clause is to make sure that the use of this link by the backup path will not impose any additional burden for spare capacity in this failure scenario. This small modification to the algorithms has minimal impact on the computation cost and does not change the time complexity of the algorithm.

To support path intermix, RSVP-TE [19] explicit route setup must be extended to include information on which type of bandwidth to use in each of the routers, service capacity or spare capacity.

V. Simulation Experiment Setup

Four network topologies were chosen to compare X-FORC and SPLO. The networks are shown in Figures 2, 3, 4, 5. Network1 and Network3 are sparse networks of different sizes. Network2 and Network4 are dense networks of different sizes. Network1 and Network2 are taken from the network topologies used by Xiong et al. [17] to allow comparisons of performance of the algorithms under similar conditions. The denser networks resemble metropolitan style networks while the sparse networks resemble backbone networks. Without loss of generality in all networks, all links are bi-directional and the weights are set to 1. Request pairs are uniformly distributed and have equal bandwidth demand of one unit each. For all experiments, the result is taken from the average of results from 7 different requests sets. Each requests set is fixed with 1000 requests. Each link is also given unlimited capacity.

As well as comparing X-FORC and SPLO, we further investigate two variants of the algorithms. The first one is pX-FORC, which consists of only the polynomial-time section of X-FORC. This is to show the difference between SPLO and an alternative polynomial-time algorithm. The second variant is nSPLO, which consists of SPLO with a post-processing enhancement. The post-processing step is exactly the same as the one used in the second section of X-FORC, thus nSPLO runs in non-polynomial time. Another algorithm that will also be shown in the experiments is the non-sharing shortest path algorithm. This algorithm simply finds a shortest path that is totally disjoint from the service path. The bandwidth used in the backup path is not shared with other backup paths. The non-sharing algorithm reflects the 1+1 restoration strategy and is used as a base performance metric.

All algorithms use the same shortest path algorithm to compute the service path, so the service path will be the same for all the algorithms. The differences lies in the backup paths computed. The characteristics of the algorithms are summarized in Table 1.

Algorithm	Computation-Type	Time-Complexity
Non-sharing	Online	Polynomial
pX-FORC	Offline	Polynomial
X-FORC	Offline	Non-Polynomial
SPLO	Online	Polynomial
nSPLO	Offline	Non-Polynomial

TABLE I

ALGORITHM COMPUTATION TYPE

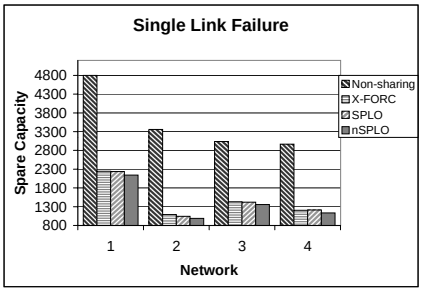

Fig. 6. Total Spare Capacity vs Network (single-link failure)

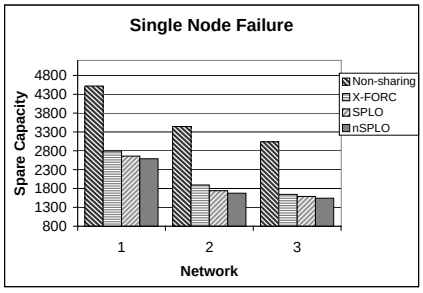

Fig. 7. Total Spare Capacity vs Network (single-node failure)

The main metric used for measuring performance is the total bandwidth used by the backup paths. That is, the total spare capacity required for 100% recovery from the defined failure scenarios. The simulations consider single link failure and single node failure separately.

VI. SIMULATION RESULTS

The first experimental results illustrate the significance of the restoration algorithm in reducing the total spare capacity requirement. The heuristic algorithms are compared with the non-sharing algorithm in single-link (Figure 6) and single-node failure scenarios (Figure 7). The results show that for single failures, the heuristic algorithms reduce the spare capacity requirement by 48–70%. The spare capacity requirement for the non-sharing algorithm for single node failures is not significantly larger than for single link failures. The extra capacity derives from the slightly longer length of the backup paths to communicate around the node failure. The heuristic algorithms suffer a larger performance reduction because of the need to protect from multiple link failures caused by the single node failure. However, the spare capacity reduction in range of 26–50% is still very substantial. Note that the results for the non-sharing algorithm for Network1 are distorted because of blocked requests in the single-node failure scenario. Blocking occurs because the algorithm fails to find a path that is node disjoint from the service path. In Network1, the blocking is because of trapping [20], where the decision made for the service path traps the network into the blocked state. The approximation algorithms do not have this problem because they are based on FORC, which uses one backup path per link/node failure. This highlights the flexibility of the FORC restoration strategy to avoid traps in network topologies. There is little meaning in analyzing offline algorithms when blocked requests exist, thus the blocking requests are removed from the request set in Network1 for single-node failures.

We focus more closely on the results for the approximation algorithms in Figures 8 and 9. Figure 8 displays the results for these algorithms for single-link failures. We can observe that SPLO performs the same and in most cases slightly better (up to 5%) than X-FORC across the network topologies. SPLO outperforms the polynomial-time pX-FORC by 8–15%. The best performer is the nSPLO algorithm, which does better than SPLO by up to 7%. For single failures, as shown in Figure 9, SPLO also does better than X-FORC and by a larger margin of up to 8% and 10–17% over pX-FORC. The non-polynomial time nSPLO still outperforms X-FORC by 5–10% and SPLO by up to 3%.

To analyze these performance differences between the algorithms, some additional results are required and are shown in Figures 10 and 11. The graphs show the average backup path length for different failure scenarios. Comparing these graphs with the corresponding graphs in Figures 8 and 9 shows the relationship between backup path length[3] and spare capacity. The algorithms that perform better have longer backup paths. This is because of the need to move away from

[3]Length is in hop counts.

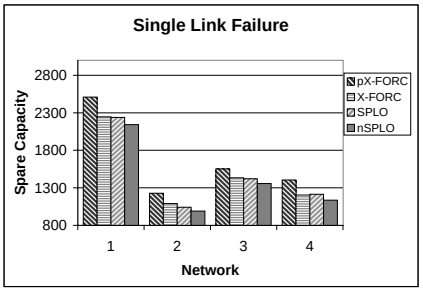

Fig. 8. Total Spare Capacity vs Network (single-link failure)

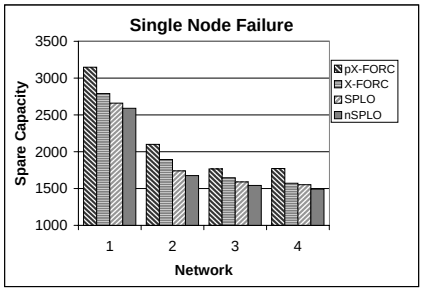

Fig. 9. Total Spare Capacity vs Network (single-node failure)

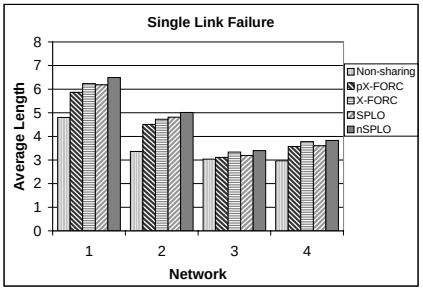

Fig. 10. Average Backup Path Length vs Network (single-link failure)

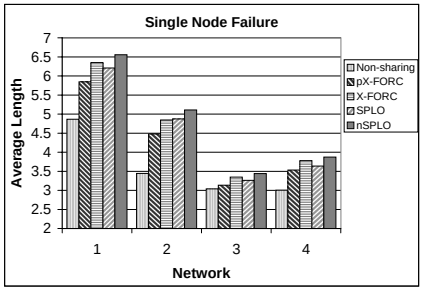

Fig. 11. Average Backup Path Length vs Network (single-node failure)

the service path to find opportunities for finding spare capacity it can utilize. This indicates that the more distributed the backup paths are across links, the better the performance. The non-sharing algorithm produces the shortest backup paths, and they are significantly shorter than the backup paths from the approximation algorithms.

Using this relationship as the base argument, we can go deeper into the algorithms and find the logic that differentiates their performance. The pX-FORC algorithm uses the failure scenario loop where it computes all the backup paths in a given scenario. This puts priority on better utilization between backup paths in the same failure scenario. However, all the backup paths must recover from the same failed link so the backup paths will tend to be packed around the area near the failed link. Therefore maximum sharing occurs within that small area. The SPLO algorithm uses the service path link loop where it computes all the backup paths for a service path only. In this case, priority is given to best utilizing spare capacity between backup paths belonging to the same service path. The spare capacity is thus spread over a large area surrounding the service path. Therefore, SPLO naturally distributes spare capacity in a wider area of the network than pX-FORC. From data not shown here, the spare capacity used in each failure scenario (G_l^s) tends to be lower for more links when running the SPLO algorithm compared with pX-FORC. The iterative post-processing loop allows X-FORC to redo some decision made by pX-FORC. The result is that it spreads the backup paths further out and so increases the backup path length. However, the decisions made by pX-FORC force X-FORC into inferior local optima compared to those that SPLO can reach. This is reflected by the consistent best performance of nSPLO. Although SPLO does not reach the local optima, it goes very close (nSPLO), and still does better than X-FORC because of better distribution of spare capacity.

Another implication of this relationship is that there is a trade-off between spare capacity utilisation and delay constraint on backup paths. Delay constraint normally restricts the choice of the backup path and pull it closer to the shortest backup path. This means that the backup path can only utilise spare capacity over a small region and that spare capacity will be distributed over a smaller region. The end result will be higher total spare capacity requirement in the network.

We further investigate the suitability of using SPLO in traffic aggregation situations. Traffic aggregation is frequently used by network providers to reduce the number of Label Switched Paths (LSPs) in the network. Reducing LSPs improves the scalability and reduces complexity of the network. To show the effect of aggregation, source-destination-based traffic aggregation was used on the path requests before they were passed to the algorithms for computation. The results of this experiment are shown in Figure 12. The observation made is that all three (X-FORC, SPLO, nSPLO) algorithms increase spare capacity usage by up to 5% only. This implies that spare capacity sharing between backup paths is still very significant under traffic aggregation and SPLO can be used for online traffic engineering applications.

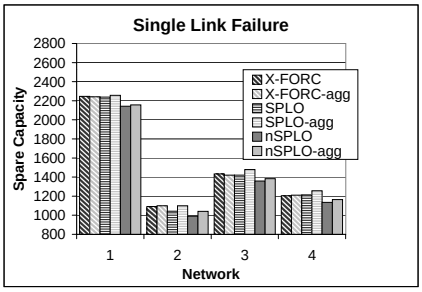

Fig. 12. Total Spare Capacity vs Network (single-link failure)

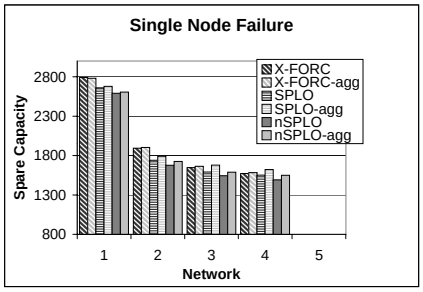

Fig. 13. Total Spare Capacity vs Network (single-node failure)

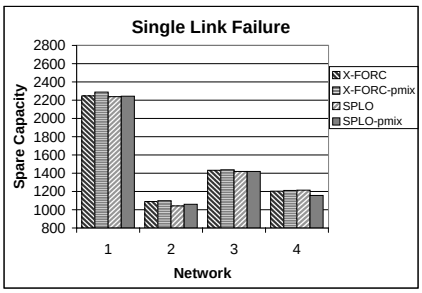

Fig. 14. Total Spare Capacity vs Network (single-link failure)

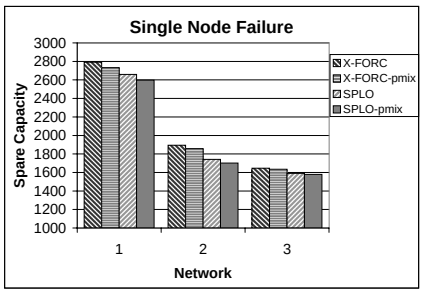

Fig. 15. Total Spare Capacity vs Network (single-node failure)

Finally, we investigate the use of path-intermix as an enhancement to SPLO and X-FORC. The results are shown in Figures 14 and 15. For single-link failures, the performances observed are improvement or degradation of performance by 1%. For single-node failures, path-intermix improves performance for both algorithms by up to 4%.

From the argument we made earlier, the better performance algorithms use longer backup paths and spread the backup paths more. Path-intermix contradicts this argument and acts to force backup paths closer to the service paths. The consequence of this is that the net gain made by exploiting the service path bandwidth is offset by the lost opportunity for better sharing through wider distribution of the backup path. Average backup path length results (not included in this paper) show that the algorithms under path-intermix give shorter path lengths. For single-link failures, the gains made by path-intermix seem to be about the same as the offset, thus the resulting performances fluctuate. For single-node failures, path-intermix seems to perform a little better. This is because node failure restricts the distribution of the backup paths therefore increasing the weight on the gains made from reusing service bandwidth.

VII. Run-time Performance Benchmark

To illustrate important difference in computation-time of the heuristic algorithms, a simple run-time benchmark experiment is demonstrated. Using non-optimized implementations of SPLO, X-FORC, and nSPLO, the processing times to run the experiments are shown in Table 2. The experimental runs were made on a PIII 700 Mhz workstation with 512 MB of RAM. The results show that the iterative step is extremely expensive and is not suitable for online applications. The execution time for X-FORC and nSPLO depends strongly on the number of iterations and the number of iterations in turn is dependent on the requests. Therefore, the figures should not be used to compare X-FORC and nSPLO and should only be used to give an idea of the computation difference between the polynomial-time algorithms and their non-polynomial time counterparts.

Network	pX-FORC	X-FORC	SPLO	nSPLO
1	4.4s	362s	4.5s	500s
2	1s	100s	1.2s	150s
3	0.69s	11.5s	0.7s	12.8s
4	1.3s	100.3s	1.25s	100s

TABLE II

Algorithm Run-times

On average, the post-processing takes about 3–4 iterations for both nSPLO and X-FORC. Post-processing is only suitable for offline applications or periodic idle time improvements in online applications. Post-processing requires a final iteration that does not improve the result before the iteration stops. One heuristic to limit processing time is to limit the number of iterations. The limit can be set to the average iterations measured minus one. The potential gains from the iterations truncated in this way are minor, because most improvements are usually obtained in the earlier iterations.

VIII. Conclusion

Deploying a restoration strategy that allows sharing of spare capacity shows strong economic advantages while reducing the risk of validating SLAs. In this paper, we focused only on failure-oriented restoration strategies and devised a new polynomial time algorithm (SPLO) that performs better than the existing non-polynomial algorithm (X-FORC). An optional post-processing enhancement to SPLO was shown to improve performance by a further 2–7%. However, the post-processing enhancement is very computationally expensive and is not guaranteed to run in polynomial-time.

SPLO naturally works for online computational applications and its polynomial-time nature improves the processing scalability of the online system. The impact of source-destination traffic aggregation on SPLO is relatively small (up to 5%) thus making the algorithm suitable for use in traffic engineering applications.

The results in this paper capture the potential for using SPLO but further diverse experiments should be made to see how it performs under different network environments and applications, for example, different network topologies and different request distributions.

We also introduced and investigated the path-intermix concept. Results showed that path-intermix can reduce the average backup path length, but the performance advantages in terms of spare capacity are only significant for single-node failures.

References

[1] E. Rosen, A. Viswanathan, and R. Callon, "Multiprotocol Label Switching Architecture," *IETF RFC 3031*, 2001.

[2] J. Boyle, V. Gill, A. Hannan, D. Cooper, D. Awduche, B. Christian, and W.S. Lai, "Applicability statement for traffic engineering with MPLS," *IETF RFC 3346*, 2002.

[3] D. Awduche, J. Malcolm, J. Agogbua, M. O'Dell, and J. McManus, "Requirements for Traffic Engineering Over MPLS," *IETF RFC 2702*, 1999.

[4] L. Berger, "Generalized multi-protocol label switching (GMPLS) signaling functional description," *IETF RFC 3471*, 2003.

[5] L. Berger, "Generalized multi-protocol label switching (GMPLS) resource reservation protocol-traffic engineering (RSVP-TE) extensions," *IETF RFC 3473*, 2003.

[6] P. Ashwood-Smith and L. Berger, "Generalized multi-protocol label switching (GMPLS) signaling constraint-based routed label distribution protocol (CR-LDP) extensions," *IETF RFC 3472*, 2003.

[7] G. Swallow, "MPLS advantages for traffic engineering," *IEEE Communications 37(12)*, pp. 54–57, 1999.

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

[9] H. Sakauchi, Y. Nishimura, and S. Hasegawa, "A self-healing network with an economical spare-channel assignment," in *Globecom*, 1990.

[10] M. Herzberg, S. Bye, and A. Utano, "The hop-limit approach for spare-capacity assignment in survivable networks," *IEEE/ACM Transactions on Networking*, vol. 3, December 1995.

[11] R. Iraschko, M. MacGregor, and W. Grover, "Optimal capacity placement for path restoration in mesh survivable networks," in *ICC*, 1996.

[12] Y. Xiong and L. Mason, "Restoration strategies and spare capacity requirements in self-healing ATM networks," *IEEE/ACM Transactions on Networking*, vol. 7, February 1999.

[13] M. Kodialam and T.V. Lakshman, "Dynamic routing of bandwidth guaranteed tunnels with restoration," in *Infocom*, 2000.

[14] G. Li, D. Wang, C. Kalmanek, and R. Doverspike, "Efficient distributed path selection for shared restoration connections," in *Infocom*, 2002.

[15] K. Murakami and H. Kim, "Joint optimization of capacity and flow assignment for self-healing atm networks," in *ICC*, 1995.

[16] Y. Liu, D. Tipper, and P. Siripingwutikorn, "Approximating optimal spare capacity allocation by successive survivable routing," in *Infocom*, 2001.

[17] Y. Xiong and L. Mason, "Restoration strategies and spare capacity requirements in self-healing ATM networks," in *Infocom*, 1997.

[18] Y. Bejerano, Y. Breitbart, A. Orda, R. Rastogi, and A. Sprintson, "Algorithms for computing qos paths with restoration," in *Infocom*, 2003.

[19] D. Awduche, L. Berger, D. Gan, T. Li, V. Srinivasan, and G. Swallow, "RSVP-TE: Extensions to RSVP for LSP Tunnels," *IETF RFC 3209*, 2001.

[20] D. Dunn, W. Grover, and M. MacGregor, "Comparison of k-shortest paths and maximum flow routing for network facility restoration," *IEEE JSAC*, vol. 2, 1994.