A Simple Traffic Independent Scheme for Enabling Restoration Oblivious Routing of Resilient Connections

Murali Kodialam T. V. Lakshman Sudipta Sengupta

Bell Laboratories, Lucent Technologies
101 Crawfords Corner Road
Holmdel, NJ 07733, USA

Abstract— **Fast restoration is an important feature of both MPLS and optical networks. The main mechanism for achieving fast restoration is by locally routing around failures using pre-setup detour paths. Signaling and routing protocol extensions to implement this local bypass ability are currently being standardized. To make use of this ability, dynamic schemes that jointly route primary paths and all link detours for links used by the primary paths have been previously proposed. These schemes also permit sharing of reserved restoration capacity for achieving efficiency. However, this joint computation places a significantly larger computational load on the network elements than that imposed by the shortest path computation variants typically used for unprotected network connection routing. In this paper, we propose a new scheme that is operationally much simpler, shares capacity used for restoration, and permits the network to route the primary paths in a manner that is oblivious to restoration needs. Restoration of all carried traffic is guaranteed by a new link capacity partitioning scheme that maximizes the working capacity of the network without requiring any knowledge of the traffic that will be imposed on the network. Being traffic independent for a priori link capacity partitioning and being oblivious to restoration needs for on-line network routing makes this scheme operationally simple and desirable in the sense of placing no additional routing load on the constrained computing resources at the network nodes. To compute the link capacity partitions, we develop a fast combinatorial algorithm that uses only iterative shortest path computations, and is a fully polynomial time approximation scheme (FPTAS), i.e., it achieves a $(1+\epsilon)$-factor approximation for any $\epsilon > 0$ and runs in time polynomial in the input size and $\frac{1}{\epsilon}$. The approximation scheme also allows link detour paths to be hop constrained if needed so as to bound restoration latency in optical networks.**

I. Introduction

Dynamic provisioning of bandwidth guaranteed paths with fast restoration capability is an important network service feature for both Multi-Protocol Label Switched (MPLS) networks [1] and optical mesh networks [4]. In optical networks, fast restoration has always been been a central requirement since optical transport networks carry a variety of traffic with stringent reliability requirements. Recently, there has been much interest in providing similar fast restoration capabilities in MPLS networks [2] in order to provide the needed reliability for services such as packetized voice, critical VPN traffic, etc.

Link and Path Restoration

A connection in a network can be protected at the path or link level. In link restoration (also often refered to as local restoration or fast restoration), each link of the connection is protected by a set of pre-setup detour paths that exclude the link being protected. Upon failure of the link, traffic on the link is switched to the detour paths. Thus, link restoration provides a local mechanism to route around a failure. In path restoration, the working path of the connection is protected by a diverse backup path from source to destination. Upon failure of any of the resources on the working path, traffic is switched to the backup path by the source node. Note that link restoration can typically restore service much faster than path restoration because restoration is locally activated and unlike for path restoration, there is no need for failure information to propagate to the source.

Two important performance metrics for restoration schemes are low restoration latency and low restoration overhead in capacity usage. Since restoration capacity is not used under normal no-failure conditions (except by low priority pre-emptible traffic), the objective of minimizing restoration capacity overhead in the network translates to higher network utilization. The 50 ms restoration latency and 50% restoration capacity overhead associated with SONET BLSR rings are benchmark figures for these two metrics. Other important metrics are operational simplicity and the additional computational load imposed on the network elements in order to route restorable connections.

<u>*Our Results*</u>

The goal of this paper is to develop a local restoration scheme that is efficient in capacity usage, is operationally simpler than previously proposed schemes, and which does not increase the computational load for online restorable connection routing. We achieve this by using a hybrid approach that combines on-line network routing of primary paths with an a priori allocation of bandwidth on the network links for ensuring guaranteed restoration. Sharing of restoration capacity is intrinsic to this scheme. During network configuration, the scheme partitions the bandwidth of each network link into working and restoration capacity pools. This is done in a traffic independent manner, for operational simplicity, but yet in a manner that maximizes the working capacity. Once link capacities are appropriately partitioned, *online* routing of restorable traffic in the network becomes extremely simple – any demand can be routed in an unprotected manner on the working capacity portion of the network. This greatly simplifies real-time network operation, does not add any restorable route computation overhead to the network elements and permits flexibility in the choice of routing schemes for primary path routing. The network is free to choose any of a number of schemes for online routing of bandwidth guaranteed paths such as simple shortest path routing, widest shortest paths, Minimum Interference Routing [8], etc. Note that schemes such as those presented in [9] jointly route the primary paths and all the necessary link detours in an online fashion.

To compute the link partitions, we formulate the problem of partitioning of the link bandwidth into working and restoration capacity pools as a network design problem and develop a fully polynomial time approximation scheme (FPTAS) for determining the partitioning of each link in the network so as to *maximize the working capacity of the network*, i.e., the sum of the reserved working capacity over all links. Note that in the problem formulation we consider, the point-to-point traffic matrix for the network is unknown and could vary over time. In this paper, we focus on protection against single link failures. Hence, partitioning of link bandwidth usage for working and restoration traffic has to be done in a manner that *guarantees link restoration of traffic affected by any single link failure*.

We mostly use MPLS network terminology in the paper except for the specific instances when the distinction between MPLS and optical networks is necessary. The scheme developed is applicable to both MPLS and optical networks.

The rest of the paper is organized as follows. Section II describes the proposed new scheme and its benefits in detail. Section III presents linear programming formulations of the traffic independent partitioning problem and discusses the special case of equal link partition sizes. Section IV discusses the fast combinatorial algorithm for the general case of arbitrary link partition sizes. Section V presents experimental evaluation of the proposed scheme. Finally, we conclude in Section VI.

II. LINK CAPACITY PARTITIONING SCHEME

In link restoration, as described above, each link of the connection is protected by a detour path that excludes the link being protected. Upon failure of the link, traffic on the link is switched to the detour path. Since we are protecting against single link failures, there is scope for the detour paths for different links to share restoration bandwidth. This can be done using a sophisticated dynamic restoration bandwidth sharing mechanism in an offline or online [9] setting. Such a scheme would require exact knowledge of the demands to be routed and attempt to maximize sharing of resources across detour paths protecting different links.

We propose a fixed capacity reservation scheme that partitions the network into working and restoration capacity. This partitioning of the network into service and restoration capacity is done on a link by link basis. The bandwidth of each link in the network is partitioned into working and restoration capacity so as to:

1) Guarantee that for each link, a set of detour paths exist whose bandwidths sum to the working capacity of the link,
2) Guarantee that for each link, the sum of the working capacity and the shared capacity usage of the detour paths going through it is at most the total capacity of the link, and
3) Maximize the working capacity of the network, i.e., sum of the working capacity of each link.

The above guidelines translate into a network design problem with the objective of maximizing the working capacity of the network under the constraints 1 and 2 above. We develop an FPTAS for determining the partitioning of each link in the network so as to *maximize the working capacity of the network*, i.e., the sum of the reserved working capacity on each link. Note that the above problem formulation does not assume any point-to-point traffic matrix, and hence, is *traffic independent*. Constraints 1 and 2 above ensure that partitioning of link bandwidth usage for working and restoration traffic is done in a manner that *guarantees link restoration of all traffic affected by any single link failure*.

A. Merits of the Link Capacity Partitioning Scheme

Before proceeding, let us outline some of the advantages of the proposed link capacity partitioning scheme:

1) **Traffic Independent:** Much of the work in the network planning literature depends on forecasted traffic patterns for network design. Solutions obtained by such methods can deviate significantly from optimality if actual traffic differs from the forecasted traffic used during the network planning phase. Hence, network design with minimal dependence on traffic predictions is the "holy grail" of any network planning methodology. The partitioning scheme we propose does not use any traffic information and hence operates in a traffic independent fashion.
2) **Restoration Oblivious Routing:** The partitioning scheme makes *online* routing of restorable traffic in the network extremely simple – any demand can be routed in an unprotected fashion on the working capacity network using simple schemes like shortest cost paths, widest shortest paths, etc. This greatly simplifies network operation and also permits prior work in the literature on online routing of bandwidth-guaranteed (unprotected) paths like Minimal Interference Routing [8] to be applied to online routing with *fast restoration*.
3) **Simplified Network Operation:** The partitioning scheme implicitly provides sharing of bandwidth across detour paths protecting different links without maintaining elaborate state information. It is simple to implement and operate and does not place any additional routing load on the constrained routing resources at the network nodes. Available extensions to routing protocols like OSPF [5] can be used to reserve working and restoration bandwidth on each link of the network.
4) **Network Restoration Capacity Overhead Estimation:** The partitioning scheme enables estimation of the restoration capacity overhead of the network without any knowledge of future traffic patterns. In the absence of this scheme, forecasted traffic would have to routed across working and backup paths using sophisticated algorithms in order to minimize and estimate restoration capacity overhead.
5) **Fast and Efficient Link Partitioning Algorithm:** Our fast combinatorial algorithm computes the link partitioning sizes so as to maximize the working capacity of the network up to any given factor of closeness to optimality.

Under the partitioning scheme, the sharing of restoration resources protecting different links could be less than that with a dynamic sharing scheme that depends on exact traffic information. However, the good sharing performance of our scheme combined with the ease of implementation makes it very useful in practice.

B. Illustrative Example

We now provide two illustrations of the link capacity partitioning scheme when *all link capacities are equal*. Any given network can be assumed to be edge biconnected, i.e., the removal of a single link cannot disconnect the network. Hence, for any link $e = (i, j)$, a path B_e exists from i to j that does not include link e. For the first example, suppose that 50% of the capacity of every link is reserved for working traffic. Then, when a link e fails, its working traffic, which is at most 50% of its capacity, can be rerouted on detour B_e. This is because (i) every link on B_e has 50% of its capacity reserved for restoration traffic, and (ii) all link capacities are equal. Hence, for this example, 50% of the network capacity is reserved for restoration, a fraction that is typical of SONET rings.

For the second example, assume that the edge connectivity of the network is 3, i.e., at least 3 edges need to removed to disconnect the network. In this case, for any link $e = (i, j)$, two link disjoint paths B_e and B'_e exist from i to j that do not include link e. Suppose that $\frac{2}{3} \approx 67\%$ of the capacity of every link is reserved for working traffic. Then, when a link e fails, half of its working traffic, which is at most $\frac{1}{3}$ of the link capacity, can be rerouted on detour B_e, and the other half on detour B'_e. This is because (i) every link on B_e and B'_e has $\frac{1}{3}$ of its capacity reserved for restoration traffic, (ii) detours B_e and B'_e are link disjoint, and (iii) all link capacities are equal. Hence, for the second example, 67% of the network capacity is reserved for restoration.

The algorithm presented in this paper seeks to maximize the working capacity of the network under the general scenario when (i) link capacities are not necessary equal, (ii) link partition sizes are possibly different, and (iii) edge connectivity of the network is arbitrary.

C. Dynamic Routing with Link Capacity Partitioning Scheme

In this section, we discuss how traffic demands are routed dynamically in an online arrival scenario. Recall that as a consequence of the link capacity partitioning scheme, a portion of the bandwidth of each link has been reserved for working traffic and a set of detours computed whose bandwidths sum to the reserved working capacity of the link. At any given time, let r_e be the residual working capacity of link e, and let b_e^i be the residual capacity of the i^{th} detour for that link. We have $\sum_i b_e^i = r_e \;\forall\; e \in E$. A demand with bandwidth b can be routed on this link if $b \le r_e$ and $b \le \max_i b_e^i$. Let $b_e^{max} = \max_i b_e^i$. Clearly, $b_e^{max} \le r_e$.

Hence, in order to route demands in a distributed fashion at the ingress, it is sufficient to distribute the value of b_e^{max} for every link e in the network using traffic engineering extensions to a link state routing protocol like OSPF [5]. A demand of bandwidth b can then be routed at the ingress node by (i) computing the path using a simple shortest path algorithm on a subgraph of the network with links having $b_e^{max} \ge b$, and (ii) signaling the path using a signaling protocol like RSVP-TE [6] [3] or CR-LDP [7]. The assignment of a detour to every link of the connection can be done locally at the intermediate nodes during signaling. Also, algorithms for routing bandwidth-guaranteed unprotected paths in an online fashion can be used for path computation to maximize network throughput.

Because link detours are assigned to connections locally during signaling and do not carry traffic under normal no-failure conditions, the assignment of detours to connections for any given link e can be re-optimized locally (and periodically) so as to maximize the value of b_{max}^e for that link. This helps to increase the residual capacity of the link and ultimately network throughput.

In the context of MPLS, there are two ways of allocating labels for link detours. Under the first model, a *single detour* exists for all Label Switched Paths (LSPs) traversing the link, and label stacking [1] is used to nest all affected LSPs into the detour LSP when the link fails. The second model, into which our dynamic routing scheme with link capacity partitioning fits, allows different LSPs traversing a link to have possibly different detours. In this case, label mappings for the detour LSPs are established on a per connection basis for every (primary) LSP traversing the link.

III. Notation and Problem Formulation

We assume that we are given a network with nodes N and edges E, where $|N| = n$ and $|E| = m$. We let (i,j) represent a generic link in the network. The capacity of link (i,j) will be denoted by u_{ij}. To simplify the notation, we will sometimes refer to a link by e instead of (i,j). A portion $0 \le x_{ij} \le u_{ij}$ of the capacity of link (i,j) is reserved for working traffic. The remaining $(u_{ij} - x_{ij})$ portion of the link capacity is used for restoration traffic. The fraction of link (i,j) reserved for restoration traffic is given by $\alpha_{ij} = 1 - x_{ij}/u_{ij}$. The restoration capacity overhead fraction for the entire network is given by

$$\frac{\sum_{(i,j)\in E}(u_{ij} - x_{ij})}{\sum_{(i,j)\in E} u_{ij}}$$

We next present two linear programming formulations for the link partitioning problem – one is link based, and the other is path based. The latter will be used to develop the fast combinatorial algorithm (FPTAS) in Section IV. We conclude this section with a discussion of the special case when all link partition sizes are equal. For the discussion below, we use network flow terminology and concepts for which a good reference is [10].

A. Link Indexed Linear Programming Formulation

We formulate the link partitioning problem as a linear program with network flow type constraints. Let x_{kl} be the working bandwidth reserved on link (i,j). Then, when link (k,l) fails, x_{kl} amount of traffic has to be rerouted along a set of detours for this link. This can be modelled as a network flow of value x_{kl} from node k to node l, using links other than (k,l). Using variables y_{ij}^{kl} to denote this flow, we have the following linear programming formulation:

$$\text{maximize} \sum_{(k,l)\in E} x_{kl}$$

subject to

$$\sum_{j:(i,j)\in E} y_{ij}^{kl} - \sum_{j:(j,i)\in E} y_{ji}^{kl} = \begin{cases} x_{kl} & \text{if } i = k \\ -x_{kl} & \text{if } i = l \\ 0 & \text{otherwise} \end{cases}$$
$$\forall\; i \in N, (k,l) \in E \quad (1)$$

$$y_{kl}^{kl} = 0 \;\forall\; (k,l) \in E \quad (2)$$

$$x_{kl} + y_{kl}^{ij} \le u_{kl} \;\forall\; (i,j),(k,l) \in E, (i,j) \ne (k,l) \quad (3)$$

Constraint (3) ensures that the sum of the working traffic and the restoration traffic that appears on a link due to failure of any other link is at most the capacity of the link. This linear program has polynomial number of variables and constraints and hence, can be solved in polynomial time using standard LP solvers like `cplex`. Using the standard method for decomposing flows into paths [10], the set of detours for link (k,l) can be obtained from the flow variables y_{ij}^{kl}.

However, because LP solvers use standard algorithms for solving general linear programs, they run slower than combinatorial algorithms for the same problem. Moreover, combinatorial algorithms use and provide valuable insight into the structure of the problem. Hence, the motivation for designing a fast combinatorial algorithm for this problem.

B. Path Indexed Linear Programming Formulation

As before, let $0 \leq x_{ij} \leq u_{ij}$ be the working capacity on link (i,j). If link (i,j) fails, then x_{ij} amount of traffic needs to be rerouted through detour paths that originate from node i, end at node j, and are disjoint with link (i,j). Let $\mathcal{P}_{ij}$ denote the set of all paths from node i to node j that do not contain link (i,j). Let $f(P)$ denote the restoration traffic on path P after failure of the link that it protects. Among the paths in the set $\mathcal{P}_{ij}$, those that form the detour paths for link (i,j) must have their $f()$ values sum to x_{ij}. Hence,

$$\sum_{P:P\in\mathcal{P}_{ij}} f(P) = x_{ij}$$

and our objective of maximizing $\sum_{(i,j)\in E} x_{ij}$ is equivalent to maximizing

$$\sum_{(i,j):(i,j)\in E} \sum_{P:P\in\mathcal{P}_{ij}} f(P)$$

Now, let us estimate the total traffic on link $e = (i,j)$ in order to arrive at the capacity constraint for each link. The working traffic on link e is given by $x_{ij} = \sum_{P\in\mathcal{P}_e} f(P)$. Restoration traffic can appear on link e only due to the failure of some other link $f \neq e$. In that case, link e must belong to some path $P \in \mathcal{P}_f$ with $f(P) > 0$. Thus, the total restoration traffic on link e due to failure of link f is given by

$$\sum_{P:P\in\mathcal{P}_f, e\in P} f(P)$$

The total traffic (working and restoration) on each link e must be at most u_e. This allows us to formulate the path indexed linear program (provided below) for determining optimal link partitioning into working and restoration capacities.

In Section IV, we state the dual of this linear program. In general, a network can have an exponential number of paths (in the size of the network). Hence, this (primal) linear program can have possibly exponential number of variables and its dual can have an exponential number of constraints – they are both not suitable for running on medium to large sized networks. However, the usefulness of the primal and dual formulation is in designing a fast (polynomial time) combinatorial algorithm for the problem, as discussed in Section IV.

$$\text{maximize} \sum_{e:e\in E} \sum_{P:P\in\mathcal{P}_e} f(P)$$

subject to

$$\sum_{P:P\in\mathcal{P}_e} f(P) + \sum_{P:P\in\mathcal{P}_f, e\in P} f(P) \leq u_e \qquad \forall\, f \neq e, e, f \in E \tag{4}$$

C. Special Case: Equal Partition Size for all Links

In this section, we consider the case when all link partition sizes (percentage wise) are equal. Let this equal fraction be α. Given that fraction α of every link is reserved for working traffic, the maximum working traffic on link $e = (i,j)$ is αu_e. Let $F(e)$ denote the maximum network flow that can be sent from node i to node j using links other than e and under given link capacities. Since the restoration bandwidth reserved on any link is $(1-\alpha)$ times its original capacity and because of the linearity of maximum network flow, the maximum restoration traffic that can be carried over all possible detours for link e is $(1-\alpha)F(e)$. Clearly, this must be at least the working traffic on link e. Hence, we have

$$(1-\alpha)F(e) \geq \alpha u_e \quad \forall\, e \in E$$

Solving the above inequality for α, we have

$$\alpha \leq \frac{F(e)}{u_e + F(e)} \quad \forall\, e \in E$$

Thus, the maximum possible value of α is given by

$$\alpha = \min_{e\in E} \frac{F(e)}{u_e + F(e)}$$

This value of α can be computed using m maximum flow computations, one for each link e in order to determine $F(e)$, thus providing a simple polynomial time exact algorithm for the special case of equal link partition sizes.

Note that when all *link capacities are equal*, the value of $F(e)/u_e$ for link $e = (i,j)$ is the maximum number of link disjoint paths from i to j not containing e. Thus, the value of α is essentially determined by the link whose endpoints can be disconnected by removing the minimum number of links from the network, and this minimum number is equal to the edge connectivity of the given network.

IV. Fast Combinatorial Algorithm for Link Partitioning

In this section, we design an approximation algorithm that can compute link partitions upto $(1+\epsilon)$-factor of the optimal objective function value (maximum network working capacity) for any $\epsilon > 0$. The value of ϵ can be chosen to provide the desired degree of optimality for the solution. This algorithm is an FPTAS and runs in time polynomial in the input size and $\frac{1}{\epsilon}$. Since the algorithm maintains primal and dual solutions at each step, the optimality gap can be estimated by computing the ratio of the primal and dual objective function values.

The dual formulation of linear program outlined in Section III-B associates a variable $w(e,f)$ with each link pair $e \neq f, e, f \in E$. The dual variable is associated with the constraint that the working traffic on link e plus the restoration traffic that appears on link e due to failure of link $f \neq e$ is at most the capacity u_e of link e. The dual linear program can be written as:

$$\text{minimize} \sum_{e:e\in E} \sum_{f:f\in E, f\neq e} u_e w(e,f)$$

subject to

$$\sum_{f:f\in E, f\neq e} w(e,f) + \sum_{e':e'\in P} w(e',e) \geq 1$$
$$\forall\, P \in \mathcal{P}_e, e \in E \qquad (5)$$

Consider the weights $w(e,f)$ that the dual program assigns. These correspond to combinations of a link e and each other link $f \neq e$ that can fail and possibly cause restoration traffic to appear on link e. For a given link e, let $g(e)$ denote the minimum value of the left-hand-side (LHS) of constraint (5) over all paths $P \in \mathcal{P}_e$.

Given the weights $w(e,f)$, note that $g(e)$ can be computed in polynomial time as follows: Let $e=(i,j)$. Remove e from the graph and compute the shortest path from node i to node j under edge costs $c(e') = w(e',e)\ \forall\, e' \in E$, $e' \neq e$. Then, $g(e)$ is the cost of this shortest path plus the sum of weights $w(e,f)$ over all $f \neq e$.

Given a set of weights $w(e,f)$, it is a feasible solution for the dual program if and only if

$$\min_{e:e\in E} g(e) \geq 1$$

The algorithm works as follows. Start with equal initial weights $w(e,f)=\delta$ (the quantity δ depends on ϵ and is derived later in this section). Repeat the following until the dual feasibility constraints are satisfied:

1) Compute the link $\bar{e}$ for which $g(e)$ is minimum. This identifies a link $\bar{e}$ and a path $P \in \mathcal{P}_{\bar{e}}$.
2) Compute the minimum (original) capacity of each link $\bar{e}$ and each link on P. Call this Δ.
3) Update the weights $w(\bar{e},f)\ \forall\ f \neq \bar{e}$ as $w(\bar{e},f) \leftarrow w(\bar{e},f)(1+\epsilon\Delta/u_{\bar{e}})$.
4) Update the weights $w(e,\bar{e})\ \forall\ e \in P$ as $w(e,\bar{e}) \leftarrow w(e,\bar{e})(1+\epsilon\Delta/u_e)$.
5) Increment the working traffic for link $\bar{e}$ by Δ.
6) For each link $e \in P$, increment the restoration traffic on link e due to failure of link $\bar{e}$ by Δ.

When the above procedure terminates, dual feasibility constraints will be satisfied. However, primal capacity constraints on each link will be violated, since we were working with the original (and not residual) link capacity at each stage. To remedy this, we scale down the working and restoration traffic on each link uniformly so that capacity constraints are obeyed.

The idea of augmenting flows in the primal solution and updating weights in a multiplicative fashion in the dual solution has been used to solve multicommodity flow and fractional packing problems in [11].

In the context of optical mesh networks [4], cross-connects need to setup on the link detour(s) after failure for restoration. This involves end-to-end signaling on the link detour. Hence, in order to bound restoration latency in optical networks, it may be necessary to impose a hop constraint (say, at most h hops) on each link detour. This is easily incorporated into our algorithm by restricting $\mathcal{P}_e$ to contain paths of at most h hops and using the Bellman-Ford algorithm [10] to compute shortest paths bounded by a hop count of h.

The pseudo-code for the above procedure, called Algorithm LINK_PARTITION, is provided in the box below. Arrays $work(e)$ and $bkp(e,f)$ keep track respectively of the working traffic on link e and the restoration traffic that appears on link e due to failure of link f. The variable G is initialized to 0 and remains < 1 as long as the dual constraints remain unsatisfied. After the **while** loop terminates, the factor by which the capacity constraint on each link e gets violated is computed into array $scale(e)$. Finally, the array $work(e)$ is divided by the maximum capacity violation factor and the resulting values are output as the working capacity partition on each link.

The values of ϵ and δ are related, in the following theorem, to the approximation factor guarantee of Algorithm LINK_PARTITION.

Theorem 1: For any given $\epsilon' > 0$, Algorithm LINK_PARTITION computes a solution with objective function value within $(1+\epsilon')$-factor of the optimum for

$$\delta = \frac{1+\epsilon}{[(1+\epsilon)(n+m-2)]^{1/\epsilon}} \text{ and } \epsilon = 1 - \frac{1}{\sqrt{1+\epsilon'}}.$$

```
Algorithm LINK_PARTITION:

$w(e,f) \leftarrow \delta \;\; \forall\, e \neq f, e, f \in E$
$work(e) \leftarrow 0 \;\; \forall\, e \in E$ ;
$bkp(e,f) \leftarrow 0 \;\; \forall\, e \neq f, e, f \in E$
$G \leftarrow 0$ ;

while $G < 1$ do
    for each $e = (i,j) \in E$ do
        Compute shortest path $S(e)$ from $i$ to $j$
        under link costs $c$, where $c(e') = w(e', e)$
        for all $e' \neq e$ and $c(e) = \infty$ ;
        $g(e) \leftarrow \sum_{f: f \in E, f \neq e} w(e,f) + cost(S(e))$ ;
    end for
    $G \leftarrow \min_{e \in E} g(e)$ ;
    if $G \geq 1$ break ;
    Let $\bar{e}$ be the link for which $g(e)$ is minimum ;
    $\Delta \leftarrow \min(u_{\bar{e}}, \min_{e \in S(\bar{e})} u_e)$ ;
    $work(\bar{e}) \leftarrow work(\bar{e}) + \Delta$ ;
    for each $e \in S(\bar{e})$ do
        $bkp(e,\bar{e}) \leftarrow bkp(e,\bar{e}) + \Delta$ ;
    end
    for each $f \neq \bar{e}, f \in E$ do
        $w(\bar{e},f) \leftarrow w(\bar{e},f)(1 + \epsilon\Delta/u_{\bar{e}})$ ;
    end for
    for each $e \in S(\bar{e})$ do
        $w(e,\bar{e}) \leftarrow w(e,\bar{e})(1 + \epsilon\Delta/u_e)$ ;
    end for
end while

for each $e \in E$ do
    $bkp\_max(e) \leftarrow \max_{f: f \neq e, f \in E} bkp(e,f)$ ;
    $scale(e) \leftarrow (work(e) + bkp\_max(e))/u_e$ ;
end for
$scale\_max \leftarrow \max_{e \in E} scale(e)$ ;
for each $e \in E$ do
    $work(e) \leftarrow work(e)/scale\_max$ ;
end for

Output $work(e)$ as the working capacity
on link $e$ ;
```

We end this section with a bound on the running time of Algorithm LINK_PARTITION.

Theorem 2: For any given $\epsilon > 0$ chosen to provide the desired approximation factor guarantee in accordance with Theorem 1, Algorithm LINK_PARTITION runs in time

$$O\left(\frac{m^3}{\epsilon}(m + n\log n)\log_{1+\epsilon}(m+n)\right).$$

Detailed proofs for the above theorems are provide in the Appendix (section VII) towards the end of the paper.

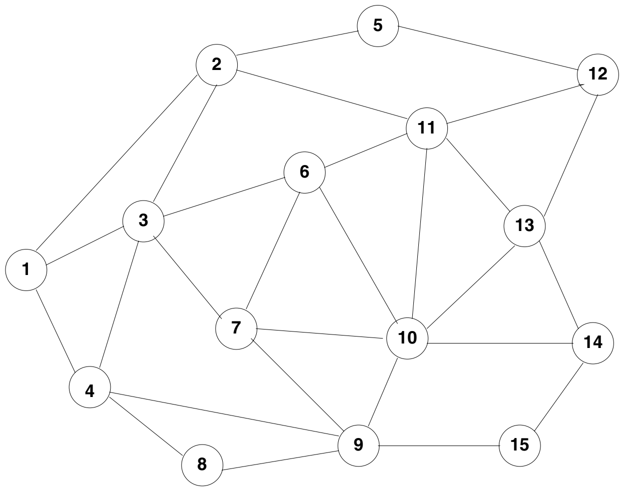

Fig. 1. 15-node, 28-link topology

V. Implementation Results

In this section, we evaluate some of the performance metrics of our link partitioning scheme using an implementation of Algorithm LINK_PARTITION of Section IV. We consider two network topologies – (i) a 15-node network with 28 bidirectional links, each with capacity 100 units, and (ii) a 70-node network with 103 bidirectional links, each with capacity 100 units. These topologies are representative of US carrier backbone networks in their size range. The 15-node network is shown in Figure 1. For the results, we ran Algorithm LINK_PARTITION up to within 5-10% of optimality (except as otherwise stated).

In Tables 1 and 2, we summarize the results for the 15-node and 70-node networks respectively. For the 15-node network, the network working capacity achieved by our scheme with arbitrary link partition sizes is 72.12%, while the same quantity with equal link partition sizes is 50%. Recall that the latter can be computed using the method outlined in Section III-C. Hence, in moving from equal to arbitrary link partition sizes, the scheme achieves a decrease of 44.24% in the restoration capacity overhead of the network. To give some indication of the variability in different link partition sizes, we also mention the maximum and minimum link partition sizes, which are 94.52% and 48.87% respectively.

For the 70-node network, the link partitioning scheme with arbitrary partition sizes achieves a working capacity of 63.61%, while the same quantity with equal link partition sizes is 50%. Thus, in moving from equal to arbitrary link partition sizes, the scheme achieves a decrease of 27.22% in restoration capacity overhead for the 70-node network. The maximum and minimum link partition sizes are 90.8% and 48.18% respectively.

Network working capacity (arbitrary link partition sizes)	72.12%
Maximum link working partition size	94.52%
Minimum link working partition size	48.87%
Network working capacity (equal partition sizes)	50%
Restoration overhead decrease (equal vs. arbitrary link partition sizes)	44.24%

Table 1. Results for 15-node network

Network working capacity (arbitrary link partition sizes)	63.61%
Maximum link working partition size	90.8%
Minimum link working partition size	48.18%
Network working capacity (equal partition sizes)	50%
Restoration overhead decrease (equal vs. arbitrary link partition sizes)	27.22%

Table 2. Results for 70-node network

We now study the impact of bounding the link detour hop count on the network working capacity. As pointed out earlier, it might be necessary to have short link detours in order to bound restoration latency in optical mesh networks. For the case when link detours are unrestricted in hop count, the average link detour hop count is 7.12 for the 15-node network and 28.69 for the 70-node network.

In Figures 2 and 3, we plot the network working capacity as the maximum allowable hop count of any link detour is increased. The hop count axis starts at 3 for the 15-node network and at 7 for the 70-node network, since these are the respective lengths of the longest shortest hop link detours for the two topologies.

We observe that for the 15-node network, the network working capacity does not increase much beyond a link detour hop bound of 6. Similarly, for the 70-node network, a link detour hop bound of 12 achieves a network working capacity sufficiently close to the theoretical maximum. Given the preference for shorter link detours to achieve lower restoration latency, it suffices to restrict link

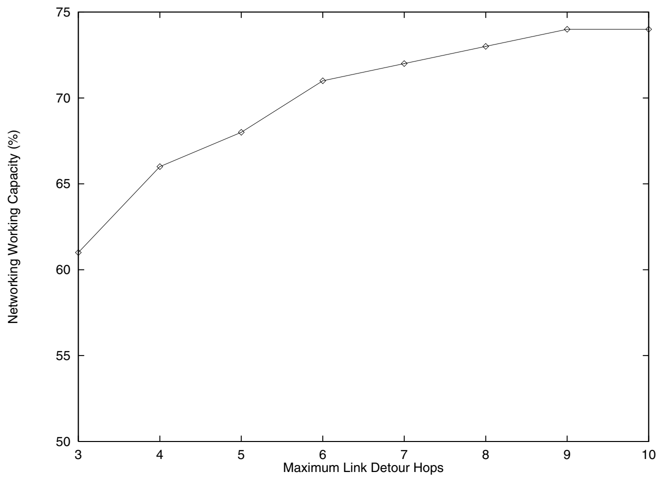

Fig. 2. Network working capacity vs. Maximum link detour hop count for 15-node network

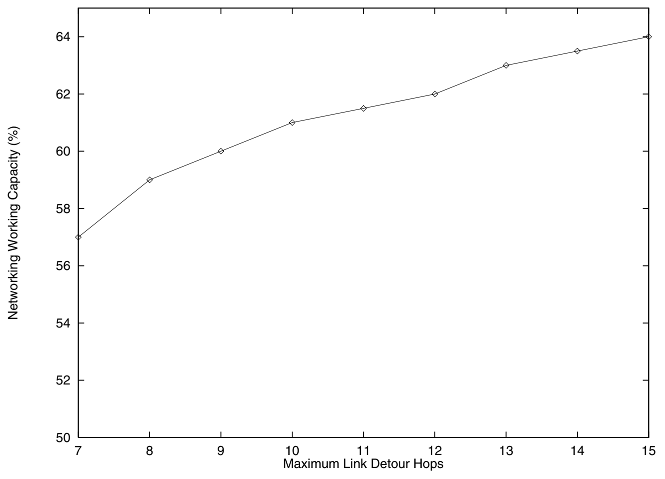

Fig. 3. Network working capacity vs. Maximum link detour hop count for 70-node network

detours to 6 and 12 hops respectively for the 15-node and 70-node networks while achieving near optimal network working capacity.

In Figures 4 and 5, we plot the average number of detours per link as the maximum allowable hop count of any link detour is increased. For the 15-node network, the average number of detours per link increases from 2.64 to 31.34 when the maximum number of hops allowed for a detour increases from 3 to unrestricted. For the 70-node network, the same number increases from 4 to 60.55 when the maximum number of allowable link detour hops is increased from 7 to unrestricted.

Since the traffic associated with a given connection in the network has to be rerouted along a *single* detour after failure, fewer number of detours for a link implies that each detour has, on the average, higher bandwidth and hence, can be used to restore relatively higher bandwidth connections traversing the protected link. Opti-

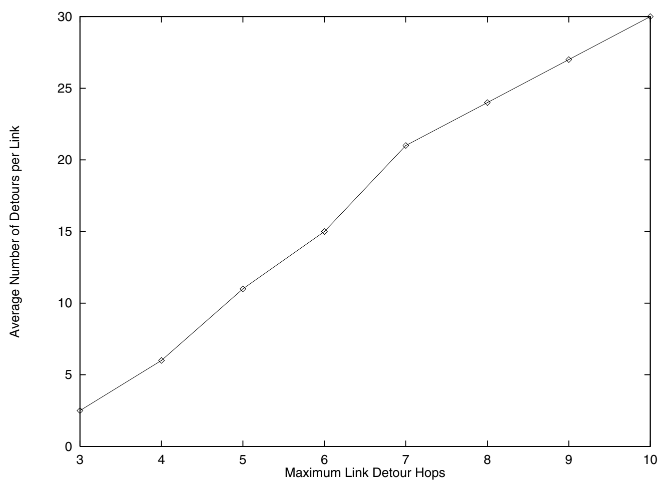

Fig. 4. Average number of detours per link vs. Maximum link detour hop count for 15-node network

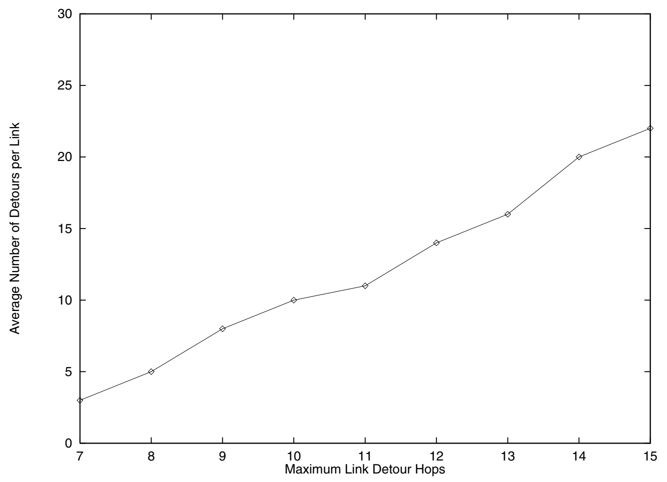

Fig. 5. Average number of detours per link vs. Maximum link detour hop count for 70-node network

cal networks are typically associated with high bandwidth connections, and hence, it may be desirable to have fewer number of detours per link in such networks so as to accommodate high bandwidth connections that need restoration. Thus, the maximum allowable link detour hop count can be used as a knob to adjust the number of detours per link which relates to the average bandwidth associated with link detours.

We conclude this section with a discussion of the behaviour of Algorithm LINK_PARTITION. Theorem 1 guarantees that the algorithm terminates with an approximation guarantee of $1 + \epsilon'$ when the values of δ and ϵ are chosen appropriately. The *actual* approximation factor associated with the solution when the algorithm terminates can be estimated by the primal-dual gap, i.e., the ratio of the dual and primal solutions in this case. We observed that this approximation factor was *appreciably better* than the approximation guarantee provided by our analysis of

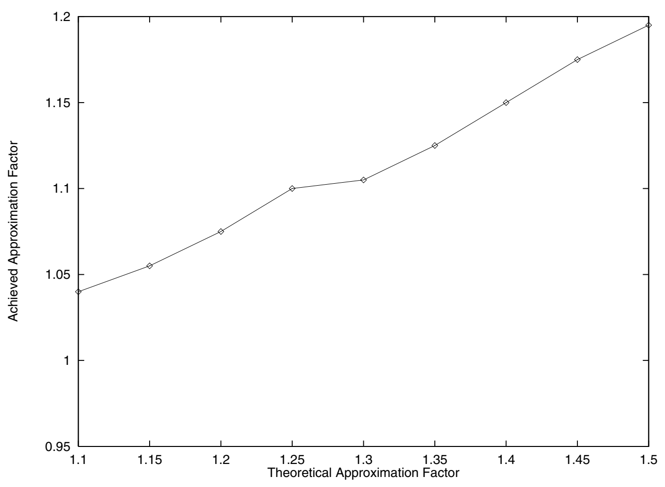

Fig. 6. Achieved approx. factor vs. Theoretical approx. factor for Algorithm LINK_PARTITION on 15-node network

the algorithm. We plot this behaviour of the algorithm in Figure 6 for the 15-node network.

As seen in Figure 6, a value of $\epsilon' = 0.3$ provides a guarantee of 30% closeness to optimality, but actually produces a solution which is within 10% of the optimal solution. A smaller value of $\epsilon' = 0.15$ produces a solution which is within 5% of optimality. Since the running time of the algorithm increases with decreasing ϵ', our observation implies that the algorithm, in practice, converges faster to the optimal solution than what our theoretical analysis predicts. For the network sizes considered in this section, Algorithm LINK_PARTITION takes few tens of seconds to execute on an Intel Pentium III 1000Mhz 256MB RAM machine.

VI. CONCLUSION

With the advent of MPLS networks, and the standardization of Generalized Multi-Protocol Label Switching (GMPLS) protocols, there has been considerable interest in schemes for fast local restoration of connections. Preferably, one would like to have schemes that are operationally simple, achieve low restoration latency while being capacity efficient, and in addition do not impose much additional computational load on the network elements. The new scheme described in the paper meets these requirements. It preserves the network's ability to route connections on-line using any chosen (unprotected) QoS routing algorithm while also guaranteeing restorability under any single link failure. Since the network routing is oblivious to restoration needs, the restoration requirement does not impose any new computational load for connection routing on the network elements. Restorability is ensured using a priori traffic independent partitioning of link capacities in the network into working and protection bandwidth pools. The traffic independent nature of the partitioning makes the scheme operationally simpler since no traffic forecasts are needed. However, network working capacity is still maximized using the developed FPTAS. Sharing of restoration capacity is automatically achieved and a priori estimation of restoration capacity needs can be done.

References

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

[2] V. Sharma, et al., "Framework for Multi-Protocol Label Switching (MPLS)-based Recovery", RFC 3469, February 2003.

[3] Ping Pan, et al., "Fast Reroute Extensions to RSVP-TE for LSP Tunnels", Internet Draft, <draft-ietf-mpls-rsvp-lsp-fastreroute-03.txt>, December 2003.

[4] S. Sengupta and R. Ramamurthy, "From Network Design to Dynamic Provisioning and Restoration in Optical Cross-Connect Mesh Networks: An Architectural and Algorithmic Overview", *IEEE Network Magazine*, vol. 15, No. 4, July/August 2001.

[5] D. Katz, K. Kompella, and D. Yeung, "Traffic Engineering Extensions to OSPF Version 2", RFC 3630, September 2003.

[6] D. Awduche, et al., "RSVP-TE: Extensions to RSVP for LSP Tunnels", RFC 3209, December 2001.

[7] P. Ashwood-Smith, et al., "Constraint-Based LSP Setup using LDP", RFC 3212, January 2002.

[8] M. Kodialam, T. V. Lakshman, "Minimum Inteference Routing of Bandwidth Guaranteed Tunnels with MPLS Traffic Engineering Applications", *IEEE Infocom 2000*, March 2000.

[9] M. Kodialam, T. V. Lakshman, "Dynamic Routing of Locally Restorable Bandwidth Guaranteed Tunnels using Aggregated Link Usage Information", *IEEE Infocom 2001*, April 2001.

[10] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin, *Network Flows: Theory, Algorithms, and Applications*, Prentice Hall, February 1993.

[11] N. Garg and J. Konemann, "Faster and Simpler Algorithms for Multicommodity Flow and other Fractional Packing Problems", *39th Annual Symposium on Foundations of Computer Science (FOCS)*, 1998.

VII. Appendix

In this section, we provide detailed proofs for Theorems 1 and 2 from Section IV for the approximation factor guarantee and running time of Algorithm LINK_PARTITION. We begin with some notation, then state some useful lemmas, and finally conclude with the proofs of the main theorems.

Given a set of dual weights $w(e, f)$, let $V(w)$ denote the dual objective function value and let $\Gamma(w)$ denote the minimum value of the LHS of dual program constraint (5) over all paths $P \in \mathcal{P}_e \; \forall \; e \in E$. Then, solving the dual program is equivalent to finding a set of weights $w(e, f)$ such that $V(w)/\Gamma(w)$ is minimized. Denote the optimal objective function value of the latter by θ, i.e., $\theta = \min_w V(w)/\Gamma(w)$.

We introduce some more notation before stating an important lemma. Let w_{i-1} denote the weight function at the beginning of iteration i of the **while** loop, and let f_{i-1} be the total flow routed up to the end of iteration $i-1$. Let $L = n + m - 2$. Suppose the algorithm terminates after iteration k.

Lemma 1: At the end of every iteration i of Algorithm LINK_PARTITION, $\forall \; 1 \leq i \leq k$, the following holds

$$\Gamma(w_i) \leq \delta L \prod_{j=1}^{i} [1 + \frac{\epsilon}{\theta}(f_j - f_{j-1})]$$

Proof: Let e be the edge and $P \in \mathcal{P}_e$ the corresponding detour along which flow is augmented during iteration i. Recall that the weights are updated as follows:

- $w_i(e, f) = w_{i-1}(e, f)(1 + \epsilon(f_i - f_{i-1})/u_e) \; \forall \; f \neq e$, and
- $w_i(e', e) = w_{i-1}(e', e)(1 + \epsilon(f_i - f_{i-1})/u_{e'}) \; \forall \; e' \in P$.

All other weights remain unchanged. Using this, we have

$$\begin{aligned} V(w_i) &= \sum_{e,f \in E, e \neq f} u_e w_i(e, f) \\ &= \sum_{e,f \in E, e \neq f} u_e w_{i-1}(e, f) + \\ &\quad \epsilon(f_i - f_{i-1})[\sum_{f: f \neq e} w_{i-1}(e, f) + \\ &\quad \sum_{e': e' \in P} w_{i-1}(e', e)] \\ &= V(w_{i-1}) + \epsilon(f_i - f_{i-1})\Gamma(w_{i-1}) \end{aligned}$$

Using this for each iteration down to the first one, we have

$$V(w_i) = V(w_0) + \epsilon \sum_{j=1}^{i} (f_j - f_{j-1})\Gamma(w_{j-1}) \qquad (6)$$

Now consider the weight function $w_i - w_0$. Clearly, $V(w_i - w_0) = V(w_i) - V(w_0)$. Also, the quantity $\Gamma(w)$ is a sum of at most L weights $w(e, f)$, since it is comprised of $m-1$ weights $w(e, f) \; \forall \; f \neq e$ associated with edge e plus weights along the link detour that are at most $n-1$ in number. Hence, we have $\Gamma(w_i - w_0) \geq \Gamma(w_i) - \delta L$. Since θ is the optimal dual objective function value, we have

$$\theta \leq \frac{V(w_i - w_0)}{\Gamma(w_i - w_0)} \leq \frac{V(w_i) - V(w_0)}{\Gamma(w_i) - \delta L}$$

whence,

$$V(w_i) - V(w_0) \geq \theta(\Gamma(w_i) - \delta L)$$

Using this in equation (6), we have

$$\Gamma(w_i) \leq \delta L + \frac{\epsilon}{\theta} \sum_{j=1}^{i} (f_j - f_{j-1})\Gamma(w_{j-1}) \qquad (7)$$

The property claimed in the lemma can now be proved using inequality (7) and mathematical induction on the iteration number. We omit the details here, but point out that the induction basis case (iteration $i = 1$) holds since $w_0(e, f) = \delta \; \forall \; e \neq f$ and $\Gamma(w_0) \leq \delta L$. ■

We now estimate the factor by which the flow value f_k in the primal solution when the algorithm terminates needs to be scaled to ensure that link capacity constraints are not violated.

Lemma 2: When Algorithm LINK_PARTITION terminates, the primal solution needs to be scaled by a factor of at most

$$\log_{1+\epsilon} \frac{1+\epsilon}{\delta}$$

to ensure primal feasibility.

Proof: Consider any link e and associated weight $w(e, f)$ for some $f \neq e$. The value of $w(e, f)$ is updated when flow is augmented on edge e under either of the following circumstances:

- Link e is the primary link being protected by a detour path, in which case the flow is working traffic on this link, or
- Link e is on a detour path protecting link f, in which case the flow appears as restoration traffic on link e under failure of link f.

Let the sequence of flow augmentations on link e that *require update of weight* $w(e, f)$ be $\Delta_1, \Delta_2, \ldots, \Delta_r$, where $r \leq k$. Let $\sum_{i=1}^{r} \Delta_i = \kappa u_e$, i.e, the total flow routed on link e exceeds its capacity by a factor of κ.

Since the algorithm terminates when $\Gamma(w) \geq 1$, and since dual weights are updated by a factor of at most $1 + \epsilon$ after each iteration, we have $\Gamma(w_k) \leq 1 + \epsilon$. Note that just before each augmentation mentioned above, the value $w(e, f)$ is one of the summing components of $\Gamma(w)$. Hence, $w_k(e, f) \leq 1 + \epsilon$. Also, the value of $w_k(e, f)$ is given by

$$w_k(e, f) = \delta \prod_{i=1}^{r} (1 + \frac{\Delta_i}{u_e} \epsilon)$$

Using the fact that $(1 + \beta x) \geq (1 + x)^{\beta} \; \forall \; x \geq 0$ and any $0 \leq \beta \leq 1$ and setting $x = \epsilon$ and $\beta = \frac{\Delta_i}{u_e} \leq 1$, we have

$$\begin{aligned} 1 + \epsilon \; \geq \; w_k(e, f) \quad & \geq \quad \delta \prod_{i=1}^{r} (1 + \epsilon)^{\Delta_i / u_e} \\ & = \quad \delta (1 + \epsilon)^{\sum_{i=1}^{r} \Delta_i / u_e} \\ & = \quad \delta (1 + \epsilon)^{\kappa} \end{aligned}$$

whence,

$$\kappa \leq \log_{1+\epsilon} \frac{1+\epsilon}{\delta}$$

■

Proof of Theorem 1:

Using Lemma 1 and the fact that $1 + x \leq e^x \; \forall \; x > 0$, we have

$$\begin{aligned} \Gamma(w_i) \quad & \leq \quad \delta L \prod_{j=1}^{i} e^{\frac{\epsilon}{\theta}(f_j - f_{j-1})} \\ & = \quad \delta L e^{\epsilon f_i / \theta} \end{aligned}$$

The simplification in the above step uses telescopic cancellation of the sum $(f_j - f_{j-1})$ over j. Since the algorithm terminates after iteration k, we must have $\Gamma(w_k) \geq 1$. Thus,

$$1 \leq \Gamma(w_k) \leq \delta L e^{\epsilon f_i / \theta}$$

whence,

$$\frac{\theta}{f_k} \quad \leq \quad \frac{\epsilon}{\ln(1/\delta L)} \tag{8}$$

From Lemma 2, the objective function value of the feasible primal solution after scaling is at least

$$\frac{f_k}{\log_{1+\epsilon} \frac{1+\epsilon}{\delta}}$$

The approximation factor for the primal solution is at most the (ratio) gap between the primal and dual solution. Using (8), this is given by

$$\begin{aligned} \frac{\theta}{f_k} \quad & \leq \quad \frac{\epsilon \log_{1+\epsilon} \frac{1+\epsilon}{\delta}}{\ln(1/\delta L)} \\ & = \quad \frac{\epsilon}{\ln(1+\epsilon)} \frac{\ln \frac{1+\epsilon}{\delta}}{\ln(1/\delta L)} \end{aligned}$$

The quantity $\ln \frac{1+\epsilon}{\delta} / \ln(1/\delta L)$ equals $1/(1-\epsilon)$ for $\delta = (1+\epsilon)/((1+\epsilon)L)^{1/\epsilon}$. Using this value of δ, the approximation factor is upper bounded by

$$\frac{\epsilon}{\ln(1+\epsilon)} \frac{1}{(1-\epsilon)} \leq \frac{\epsilon}{(\epsilon - \epsilon^2/2)(1-\epsilon)} \leq \frac{1}{(1-\epsilon)^2}$$

Setting $1 + \epsilon' = 1/(1-\epsilon)^2$ and solving for ϵ, we get the value of ϵ stated in the theorem.

■

Proof of Theorem 2:

We first consider the running time of each iteration of the algorithm during which a link e and associated detour path $P \in \mathcal{P}_e$ is chosen to augment flow. Selection of this link and its detour involves a shortest path computation for each link. Shortest path computation per link can be implemented in $O(m + n \log n)$ time using Dijkstra's shortest path algorithm with Fibonacci heaps [10]. All other operations within an iteration are absorbed (up to a constant factor) by the the time taken for these m shortest path computations, leading to a total of $O(m(m + n \log n))$ time per iteration.

We next estimate the number of iterations before the algorithm terminates. Recall that in each iteration, flow is augmented along a link e and associated detour path $P \in \mathcal{P}_e$, the value being equal to the minimum capacity of the links in $P \cup \{e\}$. If this link is e, then weights $w(e, f)\ \forall\ f \neq e$ increase by a factor of $1 + \epsilon$. Otherwise, if this link is some $e' \in P$, then weight $w(e', e)$ increases by a factor of $1 + \epsilon$. Thus, with each iteration, we can associate a weight $w(e, f)$ which increases by a factor of $1 + \epsilon$.

Consider the weight $w(e, f)$ for fixed $e, f \in E$. Since $w_0(e, f) = \delta$ and $w_k(e, f) \leq 1 + \epsilon$, the maximum number of times that this weight can be associated with any iteration is

$$\log_{1+\epsilon} \frac{1+\epsilon}{\delta} = \frac{1}{\epsilon}(1 + \log_{1+\epsilon} L) = O(\frac{1}{\epsilon} \log_{1+\epsilon}(m + n))$$

Since there are a total of $m(m - 1) = O(m^2)$ weights $w(e, f)$, hence the total number of iterations is upper bounded by $O(\frac{m^2}{\epsilon} \log_{1+\epsilon}(m + n))$. Multiplying this by the running time per iteration, we obtain the overall algorithm running time as $O(\frac{m^3}{\epsilon}(m+n \log n) \log_{1+\epsilon}(m+n))$. ■