Optimizing Restoration with Segment Routing

Fang Hao, Murali Kodialam, T.V. Lakshman
Bell Laboratories, Alcatel-Lucent
Crawford Hill, NJ 07733
{firstname.lastname}@bell-labs.com

I. Abstract

Segment routing is a new proposed routing mechanism for simplified and flexible path control in IP/MPLS networks. It builds on existing network routing and connection management protocols and one of its important features is the automatic rerouting of connections upon failure. Re-routing can be done with available restoration mechanisms including IGP-based rerouting and fast reroute with loop-free alternates. This is particularly attractive for use in Software Defined Networks (SDN) because the central controller need only be involved at connection set-up time and failures are handled automatically in a distributed manner. A significant challenge in restoration optimization in segment routed networks is the centralized determination of connections primary paths so as to enable the best sharing of restoration bandwidth over non-simultaneous network failures. We formulate this problem as a linear programming problem and develop an efficient primal-dual algorithm for the solution. We also develop a simple randomized rounding scheme for cases when there are additional constraints on segment routing. We demonstrate the significant capacity benefits achievable from this optimized restoration with segment routing.

II. Introduction

Segment Routing [6], [7] is envisaged to make possible simplified flexible connection routing in IP/MPLS networks building largely on features of existing network protocols. The main idea in segment routing is to use a sequence of segments to compose the desired end-to-end connection path. The path between each segments end points is determined by a conventional routing protocol like OSPF. The segment labels are carried in the packet header and so per-flow state is maintained only at the ingress node. A segment label is like an MPLS label and traditional push, pop, swap actions can be applied to it by the routers on the segment path [7]. Segment routing permits finer control of the routing paths and so can be used to distribute traffic for better network utilization. A central controller can exploit the full potential of segment routing by choosing segments based on the traffic pattern to judiciously distribute traffic in the network and avoid local hot-spots. This central control element can be done by a path computation element or in the case of a Software Defined Network (SDN) [11] the SDN controller. There has been some recent work on determining the optimal segment routed paths for improving network bandwidth utilization [2]. While the SDN controller can set up the segments based on measured or predicted traffic, it is not necessarily desirable to involve the controller when there are network failures. One of the key feature that segment routing offers is that each segment is routed by the IGP routing protocol and the failure recovery mechanisms of the IGP routing protocol can be used to recover from failures in a distributed manner. Thus an SDN controlled segment routed network can combine the efficiency of centralized control with the fast scalable response to failures that a distributed routing mechanism offers. This distributed restoration assumes that there are sufficient resources in the network to route around network failures. An alternative to an SDN controller is a centralized Path Computation Element (PCE) [4] that plays the same role as an SDN controller. The problem that we address in this paper is how to configure the initial segments such that there are sufficient network resources available for rerouting traffic if there are network failures. We first address the most common practical system when routing is on a single shortest path and the network has to recover from single link failures. We show how to generalize the approach in the paper to handle the case where routing is along Equal Cost Multipaths (ECMP) and the network has to recover from Shared Risk Link Group (SRLG) failures where multiple links can fail at the same time. The key to restoration planning is to share the restoration bandwidth efficiently among independent failures. The main contributions of the paper are the following:

- The first study of restoration planning in segment routed networks.
- Development of a fast Fully Polynomial Time Approximation Scheme (FPTAS) for the restoration planning problem for single link failures.
- Fast algorithms for restoration planning with ECMP and SRLG failures.

With the current strong industry interest in segment routing and related standardization efforts, we believed that the restoration optimization addressed in the paper is timely and of importance.

III. Outline of Segment Routing

The key idea in segment routing is to split the routing path into multiple segments. The segment identifiers are coded into the packet header. Routing within each segment is done by the IGP routing protocol. Figure 1 shows a network with bi-directional links. The number next to each link is its IGP link weight. Consider a connection that has to be established

978-1-4673-9953-1/16/$31.00 ©2016 IEEE

between nodes A and Z. If OSPF is used, packets belonging to this flow will be routed on the shortest path $A-N-Q-P-Z$. Assume that link $N-Q$ and $C-D$ are congested and we want to route the packet on the path $A-B-C-Q-P-Z$. This is done by breaking the path into two segments $A-B-C$ and $C-Q-P-Z$. In addition to the destination address Z, the segment labels C and Q are added to the header. The packet is routed from A to C along the shortest path $A-B-C$ and at node C, the top label is popped and the packet is now routed to Q. At node Q, the second label is popped the packet is routed to Z along the shortest path. Note that there is no per-flow state at any of the intermediate nodes in the network. If there are no segment identifiers, then packets

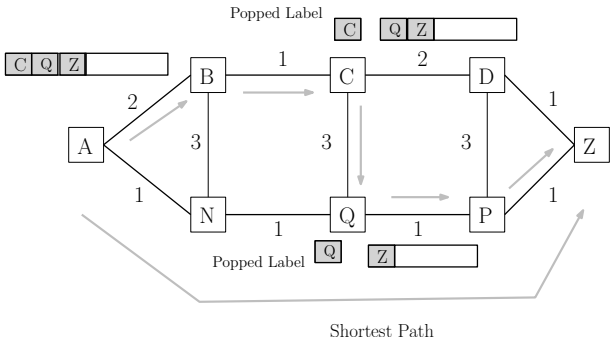

Fig. 1. Illustration of Segment Routing.

are routed along shortest paths as in standard IGP routing protocols. The other extreme is when each hop is specified in the packet header and this resembles explicit path routing. This fine grained control of the routing path enables the easy deployment of network functions like service chaining where the packet has to pass through a set of middle boxes when it goes from the source to destination. Segment routing can also be used for steering traffic to avoid hot spots in the network and hence improve network utilization. There are two basic types of segments: node and adjacency. A node segment identifies a router node. Node segment IDs are *globally unique* across the domain. An adjacency segment represents a local interface of a node. Adjacency segment IDs are typically only *locally significant* on each node. The MPLS data plane can be leveraged to implement segment routing essentially without modification since the same label switching mechanism can be used. Segment labels are distributed across the network using simple extensions to current IGP protocols and hence LDP and RSVP-TE are no longer required for distributing labels. As a result, the control plane can be significantly simplified. Moreover, unlike MPLS, there is *no need to maintain path state* in segment routing except on the ingress node, because packets are now routed based on the list of segments they carry. The ingress node has to be modified since it needs to determine the path and add the segment labels to the packet. For traffic planning problems where the objective is to route traffic so that no link is overloaded, it is generally enough to consider segment routes with just two segments. An example of a 2-segment route is shown in Figure 2. In the case of two segment routing, traffic is routed though one intermediate node. In Figure 2, the path $A-B-C-D-Z$ is represented as two segments, one from A to C and the other from C to Z. 2-segment routing can be implemented with just one extra label and gives excellent performance. Therefore, in this paper we will primarily restrict attention to two segment routing. The generalization to segments having more than two hops is straightforward and just involves higher computational times.

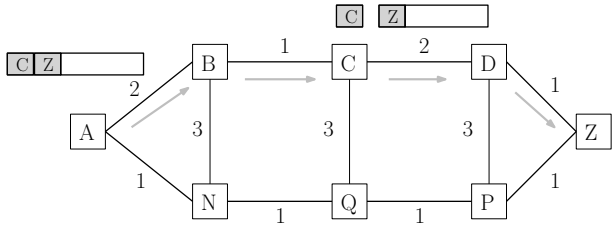

Fig. 2. Illustration of 2- Segment Routing

IV. System Model

We assume that the segment routed network is controlled by an SDN controller as shown in Figure 3. The SDN controller has full knowledge of the network topology and can communicate with all the routes in the network. Traffic

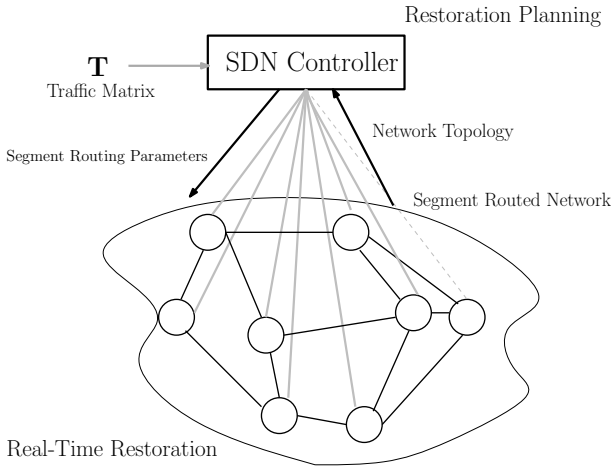

Fig. 3. Segment Routing in an SDN.

matrix information is fed to the SDN controller. Using this traffic matrix and network topology information the SDN controller determines the segment routed paths for each source-destination pair in the network. This information is fed to the edge routers that are ultimately responsible for pushing the segment labels on the packet header. Once the segments are set up, packets are routed through the network using these segments.

A. Segment Routing and Restoration

A major advantage that segment routing offers compared to explicit path routing is that when there are failures in the network, the IGP protocol recomputes the shortest path.

Therefore the segments are repaired when there are failures in the network without any intervention. This is useful, even in SDN networks, since the central controller then does not have to reroute the potentially large number of connections that may have to be rerouted with strict time constraints when there are network failures.

B. Restoration Requirements

There is a large body of research in restoration especially in optical networks. (See [12] for a survey). There is also work in IP/MPLS restoration. See for example [13] that deals with path restoration in MPLS networks. Apart from the networking community, restoration has also been studied in the optimization [3] and theoretical computer science [5] communities. Typical failure scenarios considered in the literature are

- Single link failures
- Single node failures
- Shared Risk Link Group (SRLG) failures

In the case of SRLG failures, multiple links can fail together. SRLG models networks where several logical links share the same physical infrastructure. Since the failure scenarios are independent, in each of these cases, there is potential to share restoration bandwidth among the independent failure scenarios. In the first part of the paper, we focus on the single link failures. This a scenario of interest in practice and it makes the description of the algorithm simpler. In Section VII we consider the the more complex SRLG failures (which subsumes node failures).

V. Mathematical Model

The network is represented by a graph $G = (N, E)$, where the nodes are the routers connected by directed links. Link e has an IGP link weight $w(e)$ and capacity $c(e)$. We use n to represent the number of nodes in the network and m to represent the number of links. We do not assume that the network is symmetric. The aggregate amount of traffic between nodes i and j is denoted by t_{ij}. Traffic between nodes i and j can be split across multiple paths between i and j. We assume that this split is flow based. In other words, we assume that the source node splits the traffic using a hashing scheme that ensures that all packets belonging to the same flow are routed on the same path (thus maintaining packet ordering). We assume that individual flows are relatively small compared to the total link capacity. This ensures that traffic can be spread arbitrarily between different paths. Assume that the link weights are fixed and all routing is along shortest paths using this link weight as the metric. Let S_{ij} denote the set of links on the shortest path from i to j. Note that when there are multiple shortest paths between nodes, then the network can split traffic across these equal cost paths (ECMP [9]). In the first part of the paper, we assume that there is a unique shortest path between the each pair of nodes. This is done purely to describe the algorithm. In Section VII, we show how to extend the restoration algorithm for the case where ECMP is used.

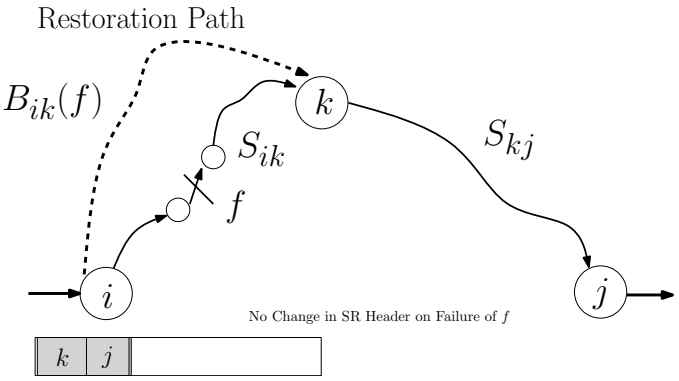

Fig. 4. Traffic from i to j is routed along a two segment path $i-k$ and $k-j$. If link f on the shortest path S_{ik} from i to k fails, packets will be routed from i to k along the restoration path $B_{ik}(f)$.

VI. Restoration Planning Problem for Single Link Failures

In this section, we assume that all flows in the network have to be protected against single link failures. All the IGP link weights are assumed to be given. This implies that the shortest path route between any pair of nodes is fixed. If there are several alternate shortest paths, then we assume that one of these paths is used for routing. In Section VII, we consider the case where equal cost multi-path is used to route packets. We denote the set of links in the shortest path between nodes i and j as S_{ij}. If some link $f \in S_{ij}$ fails, then the nodes in the network will recompute the shortest path after eliminating link f and packets will be routed along this new shortest path. Let $B_{ij}(f)$ represent the set of links in the shortest path between nodes i and j when link f fails. Note that some of the links in S_{ij} might be contained in $B_{ij}(f)$. We use $N_{ij}(f) = B_{ij}(f) \backslash S_{ij}$ to denote the set of new links on which there will be (ij) traffic flow when link f fails. Figure 5 illustrates the shortest path, restoration path and the set of new links in the restoration path between nodes i and j. We can set $B_{ij}(f) = \emptyset$ if $f \notin S_{ij}$. Since the IGP weights are known, the set of links S_{ij} and $N_{ij}()$ can be pre-computed for all pairs of nodes. Since every flow is 2-segment routed, the decision that the SDN controller has to make for each (ij) flow is to pick the intermediate node k that splits the path into two segments. If the $k = i$ or $k = j$, then the connection in routed on the the shortest path S_{ij}. If $k \neq i, j$, then the flow is first routed on the shortest path from i to k and then on the shortest path from k to j. The objective of the SDN controller is to pick these segments, such that there is sufficient link bandwidth to handle any single link failure. Let x_{ij}^k denote the amount of traffic between i and j that is routed through node k. Recall that in 2-segment routing we just have to pick one intermediate node through which flow is routed. Intermediate node $k = i$ or $k = j$ represents the shortest path from i to j. Let x_{ij}^k amount of traffic is routed from i to j through intermediate node k. We want to ensure that all the traffic between i and j is routed through some intermediate node. Therefore, $\sum_k x_{ij}^k = t_{ij} \ \ \forall i, j$.

A. Computing the Link Load

The traffic on a link can be split into primary traffic and restoration traffic. Primary traffic is the amount of traffic on the link as a result of routing flows on the link when there are no failures in the network. Restoration traffic is the traffic that flows on a link due to some failure in the network. Let $P(e)$ denote the primary flow on link e and $R(e,f)$ denote the restoration flow on link e, if link f fails. If link $e \in S_{ik}$ or S_{kj}, then there will be a traffic of x_{ij}^k that will flow on link e. Therefore, the total amount of primary traffic $P(e)$ on link e will be

$$P(e) = \sum_{(ijk): e \in S_{ik} \cup S_{kj}} x_{ij}^k.$$

The amount of bandwidth reservation for restoration traffic on link e should equal the maximum amount of flow that can result on link e due to the failure of link f in the network. This is due to the fact that link failures are independent and we only need to have enough bandwidth to carry traffic in the worst case. In figure 5, note that link e is on the restoration path for the segment $i-j$ that is carrying 10 units of traffic and segment $i-k$ carrying 7 units. Since there are no common links between these two segments, any single link failure only results in a maximum flow of 10 units and this is the reservation that we need to make on e for restoration. If $e \in N_{ik}(f) \cup N_{kj}(f)$

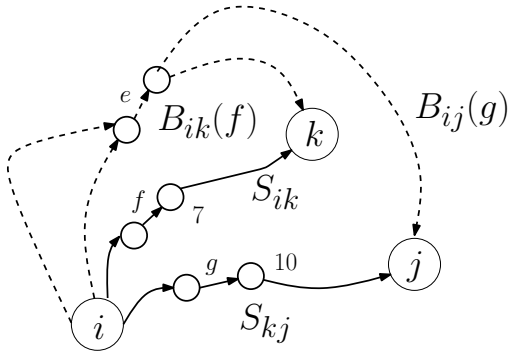

Fig. 5. Sharing Restoration Bandwidth: Link $e \in B_{ik}(f) \cap B_{ij}(g)$. The flow on segment $i-j$ is 10 units and on segment $i-k$ is 7 units. We only need to reserve 7 units of restoration bandwidth on link e.

and link f fails, then there will be a flow of x_{ij}^k on link e. Note that if $e \in B_{ik}(f) \cap S_{ik}$ or $e \in B_{kj}(f) \cap S_{kj}$ then it is already carrying a flow of x_{ij}^k that is routed from i to j through node k (before any failures). Therefore there will not be any *additional* traffic on link e if link f fails. The amount of restoration traffic $R(e,f)$ on link e if f fails is given by

$$R(e,f) = \sum_{\substack{(ijk): \\ e \in N_{ik}(f) \cup N_{kj}(f)}} x_{ij}^k.$$

Since the link failures are independent, we need to ensure that $R(e) + \max_f R(e,f) \leq C(e)$ for all links e. The routing objective is to find a set of segments such that the maximum link load under any failure scenario is as small as possible. We can formulate the problem of determining x_{ij}^k such that the maximum link utilization is minimized. In other words, the objective is to minimize ϕ such that $P(e) + \max_f R(e,f) \leq \phi \ \ c(e)$. Instead of formulating this problem directly, we introduce a new variable $\lambda = \frac{1}{\phi}$. Instead of scaling the link capacity, we instead scale up the traffic by a factor of λ. The inverse of the maximum link utilization is called the *throughput*. Our objective is to maximize the throughput λ such that when the traffic matrix is scaled up by λ the resultant primary and restoration traffic still fits in the network. The resultant maximum link utilization is $\frac{1}{\lambda}$. This formulation that resembles the maximum concurrent flow problem, permits a simple fully polynomial time approximation scheme. Towards this end, we write the restoration planning problem for the single link failure can be written as the following linear program:

$$\max \lambda$$

$$\begin{aligned}
\sum_{\substack{(ijk): \\ e \in S_{ik} \cup S_{kj}}} x_{ij}^k + \sum_{\substack{(ijk): \\ e \in N_{ik}(f) \cup N_{kj}(f)}} x_{ij}^k &\leq c(e) \ \ \forall\, f \ \ \forall e \quad (1) \\
\sum_k x_{ij}^k &\geq \lambda\, t_{ij} \ \ \forall\, (ij) \quad (2) \\
x_{ij}^k &\geq 0 \ \ \forall (ijk)
\end{aligned}$$

There are $O(n^3)$ variables and $O(n^2+m^2)$ constraints. We can directly solve this linear programming problem. We develop a simple primal-dual algorithm to solve the problem. We associate dual variables $\pi(e,f)$ with the constraint (1) and θ_{ij} with constraints (2). The dual is the following linear programming problem:

$$\min \sum_e c(e) \sum_f \pi(e,f)$$

$$\begin{aligned}
\sum_{e \in S_{ik}} \left[\sum_f \pi(e,f) + \sum_{f \in N_{ik}(e)} \pi(f,e) \right] &+ \\
\sum_{e \in S_{kj}} \left[\sum_f \pi(e,f) + \sum_{f \in N_{kj}(e)} \pi(f,e) \right] &\leq \theta_{ij} \ \ \forall k \quad (3) \\
\sum_{ij} t_{ij} \theta_{ij} &= 1 \\
\pi(e,f) &\geq 0 \ \forall e, f \\
\theta_{ij} &\geq 0 \ \forall (ij)
\end{aligned}$$

Given a pair of nodes i and j and a link $e \in S_{ij}$, we define

$$\ell_{ij}(e) = \sum_f \pi(e,f) + \sum_{f \in N_{ij}(e)} \pi(f,e) \quad (4)$$

to be the length of link e. The running time for this step for each pair (ij) is $O(m)$. We can now write constraint (3) as

$$\theta_{ij} \geq \sum_{e \in S_{ik}} \ell_{ik}(e) + \sum_{e \in S_{kj}} \ell_{kj}(e).$$

Therefore, for a given source-destination pair (ij), we can set

$$\theta_{ij} = \min_k \sum_{e \in S_{ik}} \ell_{ik}(e) + \sum_{e \in S_{kj}} \ell_{kj}(e). \quad (5)$$

The *best intermediate node* for source-destination pair (ij) is the intermediate node k that achieves the minimum value of θ_{ij}. Finding the best intermediate node for each pair of nodes involves finding the cost of every link on the shortest path. In the worst case the shortest path can have $O(n)$ links, Therefore, to evaluate the cost of picking a particular intermediate node will take $O(nm)$ time and finding the best intermediate node will take $O(n^2m)$ time. We use $\boldsymbol{\pi}$ to represent the $m \times m$ vector $\pi(e, f)$. We define $D(\boldsymbol{\pi}) = \sum_e c(e) \sum_f \pi(e, f)$ and $\rho(\boldsymbol{\pi}) = \sum_{ij} t_{ij}\theta_{ij}$. Note that θ_{ij} values are a function of $\boldsymbol{\pi}$). The dual problem can now be reformulated as the following:

$$\min_{\boldsymbol{\pi} \geq 0} \frac{D(\boldsymbol{\pi})}{\rho(\boldsymbol{\pi})}.$$

The primal-dual algorithm for solving this problem is a Fully Polynomial Time Approximation Scheme (FPTAS). In an FPTAS, we are given an ϵ and the algorithm finds a solution that is within $(1 - \epsilon)$ of the optimal solution in running time that is a function of the problem parameters and $\frac{1}{\epsilon}$. The algorithm starts off by initializing $\pi(e, f) = \frac{\delta}{c(e)}$ for all e, f where δ is a number that is computed based on ϵ and the network parameters. All flows are initialized to zero. The algorithm works in multiple *phases* where each phase comprises of one *iteration* through each source-destination pair (ij) such that $t_{ij} > 0$. We call each of these source destination pairs with non-zero demand a *demand pair*. In each iteration corresponding to source-destination pair (ij) traffic is routed from i to j in multiple *steps* until a total traffic of t_{ij} has been routed. In each step the following computations are done:

1) Set the amount of traffic routed in this iteration to $t' = t_{ij}$.
2) Find the length $\ell_{ij}(e)$ for each link e based on the current $\pi(e, f)$ as shown in Equation (4).
3) Find the best intermediate node k as per Equation (5).
4) Find the minimum capacity link in $S_{ik} \cup S_{kj}$ and set
$$m = \min_{e \in S_{ik} \cup S_{kj}} c(e).$$
5) Send a flow of $\Delta = \min\{m, t'\}$ from i to j through k. Set $x_{ij}^k \leftarrow x_{ij}^k + \Delta$.
6) Update the dual values. Let $P_{ij} = S_{ik} \cup S_{kj}$ denote links in the two segments that are used for the routing flow.
$$\pi(e, f) \leftarrow \pi(e, f)\left[1 + \epsilon\frac{\Delta}{c(e)}\right] \ \forall\, f \ \forall\, e \in P_{ij}$$
$$\pi(f, e) \leftarrow \pi(f, e)\left[1 + \epsilon\frac{\Delta}{c(f)}\right] \ \forall\, f \in N_{ij}(e) \forall\, e \in P_{ij}$$
7) Set $t' \leftarrow t' - \Delta$

This process of finding the best segments and routing flow is repeated until the termination condition is met. The running time for each demand pair is $O(n^2m)$ and there are upto n^2 demand pairs, making the running time for each phase $O(n^4m)$. Note that this is an over-estimate. The number of demand pairs could be far less than $O(n^2)$ and the cost of each step in the algorithm depends on the length of the shortest path between node pairs which is typically far less than the upper bound of n. The algorithm is described more formally below:

Data: Graph $\mathbb{G} = (\mathbb{V}, \mathbb{E})$; link capacity $c(e)$; Traffic Matrix $\mathbf{T}$, approximation accuracy ϵ;
Result: Approximate Maximum Throughput λ; feasible flow allocation $\mathbb{X}$;
Set $\delta = m^{-\frac{2}{\epsilon}}(1 - \epsilon)^{\frac{1}{\epsilon}}$;
Initialize $\pi(e, f) := \delta/c(e),\ \forall e, f$; $x_{ij}^k := 0,\ \forall (ijk)\ u = 0$;
while *dual objective* $D(\boldsymbol{\pi}) < 1$ **do** `/* phases */`
  $u := u + 1$;
  foreach *Demand pair* (i, j) **do** `/* iteration */`
    Setup traffic demand $t'_{ij} := t_{ij}$ to be fulfilled;
    while $t'_{ij} > 0$ **do** `/* steps */`
      Find Best Intermediate Node k
      Set $P_{ij} = S_{ik} \cup S_{kj}$
      Path-Update:
      $\Delta := \min\{t'_{ij},\ \min_{e \in P_{ij}} c(e)\}$;
      $x_{ij}^k := x_{ij}^k + \Delta$;
      $\pi(e, f) \leftarrow \pi(e, f)(1 + \epsilon\Delta/c(e)),\ \forall e \in P_{ij},\ \forall f$;
      $\pi(f, e) \leftarrow \pi(f, e)(1 + \epsilon\Delta/c(f)),\ \forall f \in N_{ij}(e),\ \forall e \in P_{ij}$;
      $t'_{ij} \leftarrow t_{ij} - \Delta$;
    end
  end
end
$x_{ij}^k \leftarrow x_{ij}^k / \log_{1+\epsilon} 1/\delta,\ \forall\ (ijk)$;
$\lambda := (u - 1)/\log_{1+\epsilon} 1/\delta$;

Algorithm 1: SRR: FPTAS For Segment Routing with Restoration

The algorithm is very simple to implement and has a very good running time even on large networks.

Theorem VI.1. *Let b be the dual optimal solution. Assume that traffic is scaled such that $b > 1$. Algrorithm SRR obtains a $(1-\epsilon)^{-3}$ approximation to the segment routing with restoration problem with running time $O\left(n^4 m \frac{b}{\epsilon} \log_{1+\epsilon} \frac{m}{1-\epsilon}\right)$.*

We outline the proof of this theorem in the Appendix.

VII. Handling ECMP and SRLG Failures

The primal dual approach developed in this paper can be generalized to networks where failures are of a more general nature as well as the case traffic between a source and destination is split across multiple equal cost paths. In the last section, we planned restoration paths in the case where the network has to be resilient to single link failures.

A. Shared Risk Link Group Failures

In practice, the network may subjected to more serious failures, including node failures. In addition, there can be several links in the network that share physical infrastructure. This results in multiple links failing at the same time when the physical infrastructure fails. The term Shared Risk Link group (SRLG) refers to a set of links that share a risk and can fail at the same time. In the SRLG failure model, each SRLG is specified as a set of links. A SRLG family is a collection of subsets of links that can fail at the same time. We use F to denote a set of links that fail at the same time. We use $\mathcal{F}$

to denote a collection of subset of links. In the case of single link failures, the collection $\mathcal{F} = \Big\{\{e_1\}, \{e_2\}, \ldots, \{e_m\}\Big\}$. Let $E(v_j)$ represent the set of links with v_j as one of the endpoints. Then node failures can be represented by the collection of sets $\mathcal{F} = \Big\{E(v_1), E(v_2), \ldots, E(v_n)\Big\}$. Unlike link failures where the segment routing headers are unchanged, in the case of node failures (or more generally SRLG failures) SR headers may have to be changed. For example, Figure 6 shows the failure of node k that forms the end of an intermediate segment. In this case, one option for the source is to route the packet directly to the destination j along the shortest paths S_{ij}. The methodology developed in this paper can be almost directly applied to any restoration problem as long as it is known in advance what path will be taken when there is any given failure. In segment routed networks, it generally will be shortest path from source to destination when all the failed links are removed from the network. This may involve changing the segment labels when there are failures.

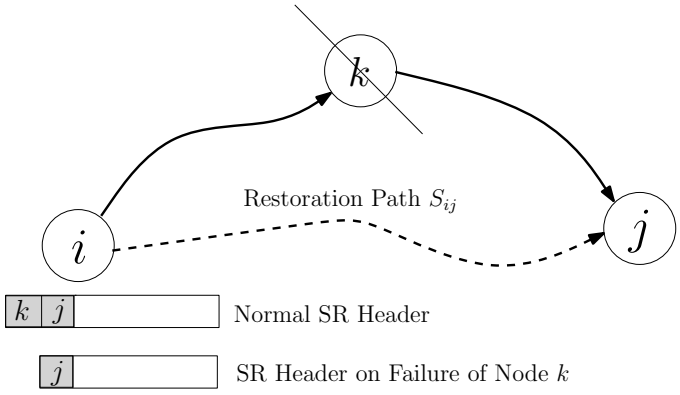

Fig. 6. Restoration on Node Failure

B. Equal Cost Multi-Paths

A routing feature commonly used in networks to distribute load is Equal Cost Multi-Path (ECMP) routing. In ECMP routing, traffic is split evenly across all minimum cost paths. The split is done by hashing on the flow id of the packet to ensure that packets belonging to the same flow are routed on the same path to avoid packet reordering at the destination. When ECMP is used, it is easy to figure out apriori the *fraction* of traffic between any pair of nodes that will be routed on a given link. This information is enough for us to formulate the restoration planning problem in networks using ECMP. Let $0 \leq \alpha_{ij}(e) \leq 1$ denote the fraction of traffic from i to j that goes through link e. In the case of standard shortest path routing, $\alpha_{ij}(e) = 1$ for all $e \in S_{ij}$ and is zero otherwise. For any given source-destination pair, it is easy to compute $\alpha_{ij}(e)$ if the IGP link weights are known. Let $\beta_{ij}(e, F)$ denote the fraction of the traffic from i to j that goes on link e if there is a failure F in the network. Note that in the SRLG model, F can be multiple links. The primary flow $P(e)$ on link e

$$P(e) = \sum_{(ijk)} \Big[\alpha_{ik}(e) + \alpha_{kj}(e)\Big] x_{ij}^k,$$

If there is failure F in the network, the amount of excess flow $\Delta_{ij}(e, F)$ on link e for flows between i and j is given by

$$\Delta_{ij}(e, F) = \Big[\beta_{ij}(e, F) - \alpha_{ij}(e)\Big]^+ x_{ij}.$$

where $[a]^+ = \max\{a, 0\}$. If we route flow x_{ij}^k from i to j via the two segment path $i - k$ and $k - j$ then the amount of primary traffic $P(e)$ on link e will be

$$P(e) = \Big[\alpha_{ik}(e) + \alpha_{kj}(e)\Big] x_{ij}^k.$$

The amount of restoration traffic on link e due to failure F due to the flow x_{ij}^k is

$$R(e, F) = \Big[\Delta_{ik}(e, F) + \Delta_{kj}(e, F)\Big] x_{ij}^k.$$

We can now formulate the problem of maximizing throughput when the routing uses ECMP and the network is subjected to SRLG failures as the following linear programming problem:

$$\max \lambda$$

$$\sum_{(ijk)} \Big[\alpha_{ik}(e) + \alpha_{kj}(e)\Big] x_{ij}^k +$$

$$\sum_{(ijk)} \Big[\Delta_{ik}(e, F) + \Delta_{kj}(e, F)\Big] x_{ij}^k \leq c(e) \quad \forall\, F \;\; \forall e \quad (6)$$

$$\sum_k x_{ij}^k \geq \lambda\, t_{ij} \forall (i, j) \quad (7)$$

$$x_{ij}^k \geq \forall (i, j, k)$$

Note that that the values of $\Delta_{ij}(e, F)$ only depends on the network topology, link IGP metric and whether ECMP is used. These values can be precomputed. As in the single link failure case, if we associate dual multipliers $\pi(e, F)$ with the constraints (6) and θ_{ij} with constraints (7) we can write the dual to the linear programming problem:

$$\min \sum_e c(e) \sum_f \pi(e, f)$$

$$\sum_e \alpha_{ik}(e) \sum_F \pi(e, F) + \sum_F \sum_e \Delta_{ik}(e, F)\pi(e, F) \;+$$

$$\sum_e \alpha_{kj}(e) \sum_F \pi(e, F) + \sum_F \sum_e \Delta_{kj}(e, F)\pi(e, F) \leq \theta_{ij} \forall k$$

$$\sum_{ij} t_{ij}\theta_{ij} = 1$$

$$\pi(e, f) \geq 0 \;\forall e, F$$

$$\theta_{ij} \geq 0 \;\forall i, j$$

For a given set of $\pi(e, f)$ values, we can set

$$\ell_{ij} = \sum_e \alpha_{ij}(e) \sum_F \pi(e, F) + \sum_F \sum_e \Delta_{ij}(e, F)\pi(e, F).$$

We can set

$$\theta_{ij} = \min_k \ell_{ik} + \ell_{kj} \ \ \forall(ij)$$

as in the single link failure case. The rest of the algorithm follows the same pattern as the algorithm for single link failure. The only difference is in the running time to compute the best intermediate node. The running time in the SRLG failure case, also depends on the number of elements in the set $\mathcal{F}$ in addition to the network size.

VIII. Experimental Results

We have tested the performance of the algorithm on three network topologies.

- Topology 1: 15 nodes and 56 directed links.
- Topology 2: 33 nodes and 100 directed links.
- Topology 3: 59 nodes and 190 directed links.

Stub nodes which are not 2-connected with the rest of the network (and hence cannot recover from single link failures) are removed from the topology. We have experimented with all links having the same capacity as well as cases where the link capacities are uniform $U[10 : 100]$. The traffic matrix is varied and in each of the experiments we experimented with 50 random traffic matrices. Traffic is randomly and uniformly distributed $U[1 : 10]$. In all the experiments, we compare the throughput of the different algorithms. Recall the throughput is the inverse of the maximum link utilization.

A. Link Weights

In practice, link weights are either set to one or to the inverse of the link capacity. In all the experiments, we set the weight to one and let the segment routing adapt to the link capacities. Setting the weight to one ensures that all segments are routed in a bandwidth efficient manner. Segment routing automatically picks the two hop segments to ensure that there are no significant congestion points in the network.

B. Algorithms Compared

We compare the following algorithms.

- Shortest path routing with no restoration (SP).
- Shortest path routing with restoration (SPR).
- Segment routing (2-segment routing) without restoration (SR).
- Segment routing (2-segment routing) with restoration (SRR).
- Shared Local Restoration (SLR).
- Multicommodity flow without restoration (MC).

The shared loop restoration is a local restoration scheme studied in [1]. In this scheme restoration is done at the point of failure. There is sharing of restoration bandwidth across independent failure. The scheme uses arbitrary paths and these paths have to be set up using MPLS. Though the scheme is bandwidth efficient, it is hard to implement since tunnels have to be set up explicitly. Contrast this with segment routing with restoration that does not require any tunnel set up.

C. Comparison of 2-segment Routing (without restoration) and Maximum Concurrent Flow

The algorithms without restoration are run with the view of studying how effective 2-segment routing is when compared to multicommodity flow routing. In the case of multicommodity flow routing, we solve the maximum concurrent flow problem. The solution to the maximum concurrent flow problem gives a lower bound on the performance of any routing algorithm. Figure 7 compares the throughput of segment routing (without restoration) and maximum concurrent flow. The comparison is over all experiments in all the topologies. Note that two segment routing performs almost as well as maximum concurrent flow. The reason seems to be fact that just deviating from the shortest path is enough to avoid congestion in the network. Compare this to Figure 11 which compares the performance of shortest path routing with multicommodity flow. Note that shortest path routing performs significantly worse than 2-segment routing.

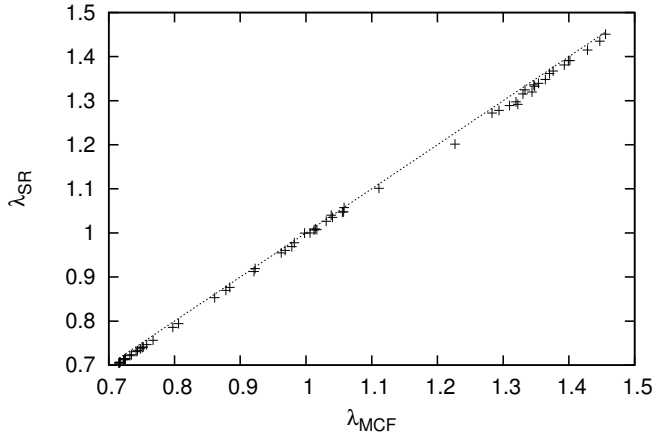

Fig. 7. Throughput comparison of 2-segment routing and multicommodity flow

D. Comparison with Shortest Path Restoration

In the shortest path with restoration, connections are routed on the shortest path. If there is a failure on the shortest path, the alternate shortest path is computed and the connection is routed on this alternate path. We again assume that restoration bandwidth is shared. In the restoration model that we study in this paper, we assume that failure information is propagated through the network by OSPF and all nodes compute the shortest path after removing the failed link. We show that enormous bandwidth efficiency of the segment routing with restoration compared to shortest path routing restoration.

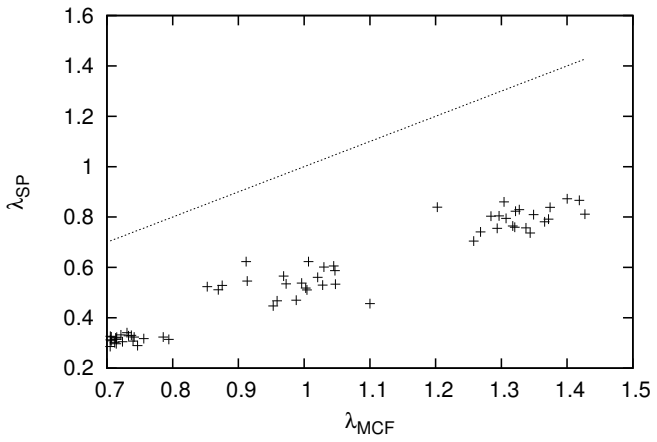

Fig. 8. Throughput comparison of shortest path routing and multicommodity flow

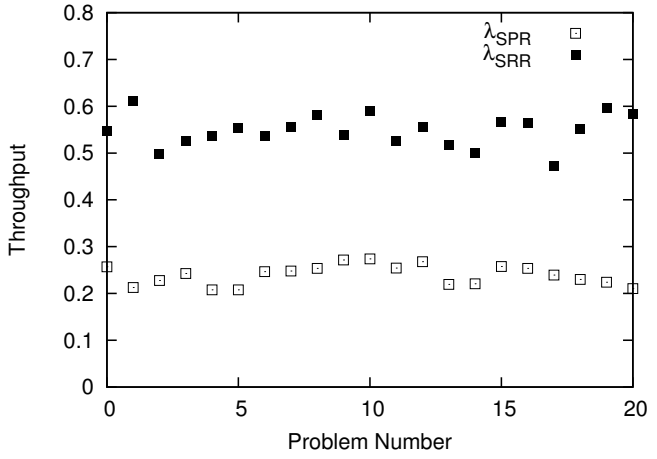

Fig. 9. 15- Node Topology: Comparison of Shortest Path with Restoration and Segment Routing with Restoration

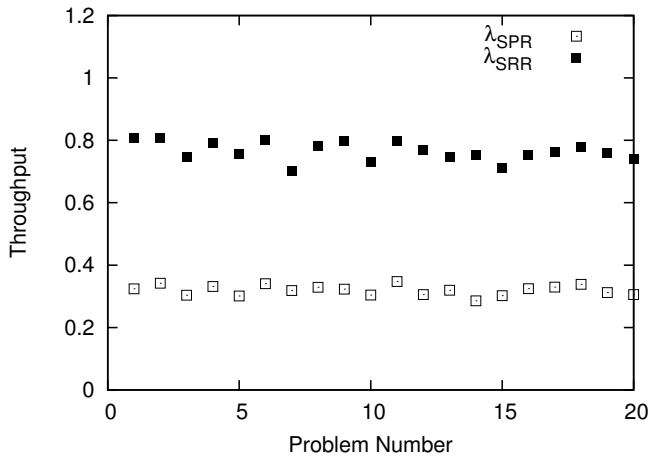

Fig. 10. 33- Node Topology: Comparison of Shortest Path with Restoration and Segment Routing with Restoration

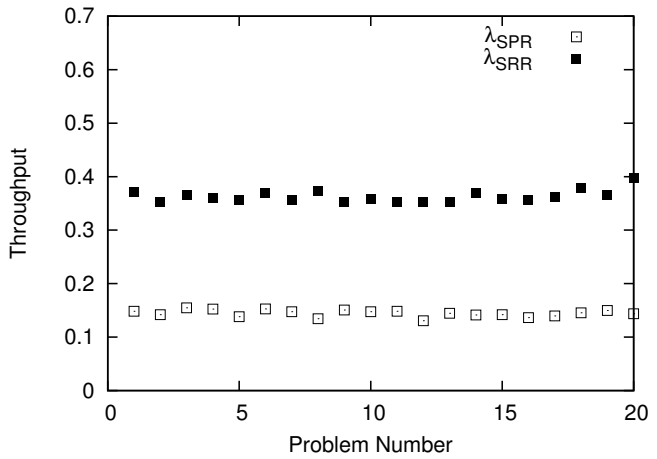

Fig. 11. 59- Node Topology: Comparison of Shortest Path with Restoration and Segment Routing with Restoration

E. Comparison with Shared Local Restoration

We now compare the throughput of segment routing with restoration with shared local restoration. Figure 12 shows that segment routing is slightly more throughput efficient that shared local restoration in spite of the fact that it is very simple to implement unlike shared local restoration.

REFERENCES

[1] R.S. Bhatia, M. Kodialam, T.V. Lakshman, S. Sengupta, "Bandwidth Guaranteed Routing with Fast Restoration Against Link and Node Failures," IEEE/ACM Trans. Networking,16(6):1321–1330, 2008.

[2] R.S. Bhatia, F. Hao, M. Kodialam, T.V. Lakshman, "Optimized Network Traffic Engineering using Segment Routing," IEEE Infocom,2015.

[3] D. Bienstock, G. Muratore, "Strong inequalities for capacitated survivable network design problems," Math. Programming, 89:127147, 2001.

[4] A. Farrel et. al, "A Path Computation Element (PCE)-Based Architecture," IETF, RFC 4655, Aug. 2006.

[5] C. Chekuri, A. Gupta, A. Kumar, J. Naor, D. Raz, "Building Edge-Failure Resilient Networks," Integer Programming and Combinatorial Optimization, 2002.

[6] C. Filfils et. al, "Segment Routing Architecture," IETF, Internet Draft, Dec. 2013.

[7] C. Filfils et. al, "Segment Routing with MPLS data plane," IETF, Internet Draft, Apr 2014.

[8] N. Garg, and J. Konemann, "Faster and Simpler Algorithms for Multi-commodity Flow and other Fractional Packing Problems," SIAM Journal on Computing, 37(2):630-652, 2007.

[9] C. Hopps, "Analysis of an Equal-Cost Multi-Path Algorithm," IETF, RFC 2992, Nov. 2000.

[10] Y. Liu, D. Tipper, P.Siripongwutikorn , "Approximating Optimal Spare Capacity Allocation by Successive Survivable Routing," IEEE/ACM Trans. Networking,13(1):198-211, 2005.

[11] "Software-defined networking: the new norm for networks," Open Networking Foundation, 2012.

[12] S. Yu , B. Mukherjee, "Survivable Optical WDM Networks," Springer, 2005.

[13] C. Huang, V. Sharma, K. Owens, and S. Makam, "Building reliable MPLS networks using a path protection mechanism," Communications Magazine, 2002.

IX. APPENDIX

We now outline a proof of Theorem 1. The proof if Theorem 1 follows the proof of approximation algorithm for the maximum concurrent flow problem in [8]. Unlike the max concurrent flow problem which has m capacity constraints, the

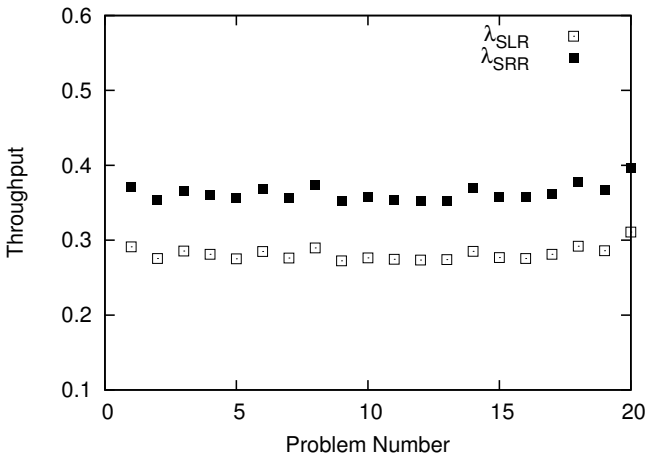

Fig. 12. 59- Node Topology: Comparison of Shared Local Restoration and Segment Routing with Restoration

segment routing restoration problem has m^2 constraints. This changes the constants that have to be used but the proof follows the same steps as in [8]. Note that the algorithm is much more complex than the FPTAS for maximum concurrent flow. This is reflected in the fact that the overall running time of SRR algorithm is quite different from the maximum concurrent flow problem.

Proof. Let $\boldsymbol{\pi}_P^I$ represent the dual vector at the end of iteration I of phase P. Each iteration represents one demand pair and each phase one loop through all demand pairs. We use $\boldsymbol{\pi}_P$ to represent the dual vector at the end of phase P after all source-destination pairs are routed in that phase. We stop the algorithm at the end of the phase when the value of $D(\boldsymbol{\pi}_P) \geq 1$. Let this happen at the end of phase t. Assume that the demands are scaled such that the dual solution $b > 1$. Assume that iteration I corresponds to demand pair (ij). After iteration I of phase P where t_{ij} units of flow has been routed from i to j along different best intermediate points, we can write

$$D(\boldsymbol{\pi}_P^I) \leq D(\boldsymbol{\pi}_P^{I-1}) + \epsilon t_{ij}\theta_{ij}(\boldsymbol{\pi}_P^{I-1}).$$

We have written $\theta_{ij}(\boldsymbol{\pi}_P^{I-1})$ to indicate that θ_{ij} is a function of $\boldsymbol{\pi}_P^{I-1}$. Adding these inequalities over all I we get

$$D(\boldsymbol{\pi}_P) \leq D(\boldsymbol{\pi}_{P-1}) + \epsilon \sum_{(ij)} t_{ij}\theta_{ij}(\boldsymbol{\pi}_{P-1}).$$

From the definition of $\rho(\boldsymbol{\pi}_P)$ and using the fact the $\boldsymbol{\pi}$ is non-decreasing, we can write

$$D(\boldsymbol{\pi}_P) \leq D(\boldsymbol{\pi}_{P-1}) + \epsilon\rho(\boldsymbol{\pi}_{P-1}).$$

Let b be the dual optimal solution. Then Since $b \leq \frac{D(\boldsymbol{\pi}_P^{I-1})}{\rho(\boldsymbol{\pi}_P^{I-1})}$ we have,

$$D(\boldsymbol{\pi}_P) \leq \frac{D(\boldsymbol{\pi}_{P-1})}{1-\frac{\epsilon}{b}}.$$

Since $D(0) = m^2\delta$, we have for $P \geq 1$

$$\begin{aligned} D(\boldsymbol{\pi}_P) &\leq \frac{m^2\delta}{\left(1-\frac{\epsilon}{b}\right)^i} \\ &\leq \frac{m^2\delta}{\left(1-\frac{\epsilon}{b}\right)}\left(1+\frac{\epsilon}{b-\epsilon}\right)^{i-1} \\ &\leq \frac{m^2\delta}{\left(1-\frac{\epsilon}{b}\right)}e^{\frac{\epsilon(i-1)}{b-\epsilon}} \\ &\leq \frac{m^2\delta}{(1-\epsilon)}e^{\frac{\epsilon(i-1)}{b(1-\epsilon)}} \end{aligned}$$

where the last inequality used the assumption that $b > 1$. The procedure stops at the first phase where

$$1 \leq D(\boldsymbol{\pi}_t) \leq \frac{m^2\delta}{(1-\epsilon)}e^{\frac{\epsilon(t-1)}{b(1-\epsilon)}}$$

which implies that

$$\frac{b}{t-1} \leq \frac{\epsilon}{(1-\epsilon)\ln\frac{1-\epsilon}{m^2\delta}}.$$

If there are $t-1$ phases for every source-destination pair (ij), we have sent $(t-1)t_{ij}$ units of flow.

Claim

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

Every $c(e)$ units of flow through edge-pair (e, f) increases its weight by at least a factor of $1+\epsilon$. Initially the weight is $\frac{\delta}{c(e)}$. After $t-1$ phases $D(\boldsymbol{\pi}_{t-1}) < 1$. Therefore the flow through e in the first $t-1$ phases of the algorithm is less than $\log_{1+\epsilon}\frac{1}{\delta}$ times its capacity. The flow can be scaled by this amount to make it feasible.

Let γ represent the ratio of the primal to the dual solution. Then

$$\gamma < \frac{\beta}{t-1}\log_{1+\epsilon}\frac{1}{\delta}.$$

Substituting the bound on $\frac{\beta}{t-1}$ we get

$$\begin{aligned} \gamma &< \frac{\epsilon\log_{1+\epsilon}\frac{1}{\delta}}{(1-\epsilon)\ln\frac{1-\epsilon}{m^2\delta}} \\ &= \frac{\epsilon}{(1-\epsilon)\ln(1+\epsilon)}\frac{\ln\left(\frac{1}{\delta}\right)}{\ln\left(\frac{1-\epsilon}{m^2\delta}\right)}. \end{aligned}$$

Setting $\delta = m^{-\frac{2}{\epsilon}}(1-\epsilon)^{\frac{1}{\epsilon}}$ we get

$$\begin{aligned} \gamma &\leq \frac{\epsilon}{(1-\epsilon)^2\ln(1+\epsilon)} \\ &\leq (1-\epsilon)^{-3} \end{aligned}$$

which holds if ϵ is small enough. Therefore, by choosing ϵ appropriately, the ratio of the primal to the dual solution can be made arbitrarily small. The number of phases is $O(\frac{b}{\epsilon}\log_{1+\epsilon}\frac{m^2}{1-\epsilon})$. The running time for each phase is $O(m^3)$ and this gives the overall running time of the algorithm. Note that the running time, as stated depends on the value of b. If we have an apriori upper bound on b then we can remove b from the running time by taking an additional logarithmic factor. In practice, it is relatively easy to run the algorithm for a few iterations and then scale the capacities until $b > 1$. □