A Simple Polynomial Time Framework to Reduce the Number of Paths in Multi-Path Routing

Vahab S. Mirrokni[1], Marina Thottan[2], Huseyin Uzunalioglu[3] and Sanjoy Paul[2]
[1] Laboratory for Computer Science, Massachusetts Institute of Technology
[2] Center for Networking Research, Bell Labs
[3] Advanced Network Planning, Bell Labs

Abstract— **The recent reduction in telecommunications spending has increased the importance of network planning to improve the return on investment on the existing network infrastructures. Therefore, tools that help in maximizing the bandwidth efficiency of the network at a minimum cost are essential. Previous work in this area focused on increasing bandwidth efficiency and reliability. In this work, in addition to increasing the bandwidth efficiency, we address the complexity of network management and operations. This issue is explicitly addressed by our novel framework, a simple polynomial time algorithm (*SimPol*) that achieves optimum network performance (in terms of congestion or bandwidth consumption) using only a small number of paths. The problem formulation is based on splittable multicommodity flows. Using *SimPol* we show that the total number of paths is at most $k+m$, where k and m are the numbers of demands and edges in the network, respectively. We extend the basic framework into an integer programming formulation to address the tradeoff between network congestion and the total number of paths. We also use *SimPol* to address the problem of implementing path/link policies such as bandwidth-limited paths. The performance of *SimPol* is evaluated through extensive simulations. We find that for large number of demands the LP-based framework provides a near-optimal solution of almost one path per demand. Using the integer programming approach, we can get exactly one path while losing about 10% to 50% in congestion depending on the number of demands. This congestion is, however, far better than the traditional shortest path routing. The framework is general and can be used in planning transport networks such as MPLS and ATM.**

I. Introduction

In all transport networks such as Frame Relay, MPLS, ATM, etc., there is a considerable amount of effort spent on network planning [1], [30], [18]. Planning is necessary to determine the required network capacity as well as to design efficient network management systems. Typically, service providers perform network planning on an annual or semi-annual basis [1]. Network planning is based on traffic measurements and forecasts [7]. During the planning phase, network paths are provisioned to handle predicted traffic demands. In general, provisioning traffic flows over a single path could result in traffic imbalances leading to poor network utilization [28]. However, by splitting the traffic demand across many paths it is possible to improve the overall network utilization through better load balancing [4]. Allowing traffic splitting raises two important issues, namely, the selection of routing paths and the number of paths required. In this paper, we address the problem of determining the number of paths required to meet the predicted traffic demands while ensuring optimal network performance. We also address congestion tradeoff issues with respect to the total number of paths required.

From a network management perspective it is desirable to minimize the number of paths. Multiple paths create significant overhead in terms of set up and maintenance. As the number of paths per demand increases, bandwidth per path, and hence, the statistical multiplexing gain decreases. In some special cases such as in MPLS networks there may even be restrictions (label space) that may limit the number of paths that can be provisioned in the network [9]. Furthermore, it is essential to have an estimate of the number of paths that must be monitored. This knowledge helps to optimize the management infrastructure that is required to manage customer service agreements. In this work using the splittable multi-commodity flow problem we define the problem of determining the minimum number of paths required to satisfy a given traffic demand matrix while ensuring optimum utilization of network resources.

Multi-commodity flow formulation has been widely used in solving different network optimization problems. Each of these problems differ in terms of their objective function. For example, a deterministic, multirate, multi-commodity flow problem is used in [18] [19] to maximize network revenue in ATM networks. In [1], the authors use a multi-commodity flow formulation to solve the problem of optimizing the restoration capacity of a network. Multi-commodity flow primitives have also been used to design minimum interference routing paths for provisioning dynamic demands [11]. The objective function in our work differs significantly from previous work. We minimize two objectives (viz) congestion and the number of paths, through the use of a polynomial time algorithm.

A. Problem Motivation

Multi-path routing is necessary to achieve optimum network utilization. An evaluation of multi path routing schemes was done by Nelakuditi and Zhang [20]. The authors proposed a dynamic multi-path routing algorithm. The goal of their work was to minimize the blocking probability for demands while still minimizing the total number of paths. In this work we consider a similar objective function in terms of minimizing the number of paths for static network planning scenarios.

In this paper we present a polynomial time algorithm (*SimPol*) to determine the number of paths that satisfies a given traffic demand matrix while optimally utilizing the network. We show that it is possible to obtain a near optimal solution that minimizes the congestion as well as the number of paths. The context for the application of this algorithm could be either a green-field scenario or an off-line re-optimization of a transport network. Our work assumes that a reasonable estimate of the traffic demand matrix is known in the network planning phase [8]. Traffic demands are assumed to be aggregated and point to point.

In the *SimPol* algorithm we use an LP-formulation of the multi-commodity flow problem to minimize congestion. The optimal path layout scheme obtained from *SimPol* can be used to design network capacity as well as a good network management system [16], [26]. The results of our algorithm have been tested on several different simulated topologies. The performance criteria used to evaluate the algorithm are (1) the average number of paths obtained per demand and (2) the maximum congestion on the network. We also compare our results with the standard shortest path routing scheme. Our results are corroborated by the extensive simulation-based studies in [20], where the authors considered the dynamic case. They show that near-optimal performance of the network can be attained using only a few paths for a given demand.

Contributions of our work: The main contribution of our work is a framework for network planning that uses a polynomial time approximation algorithm. The framework can be used to solve a splittable multi-commodity flow problem with optimum network utilization and with at most $k+m$ paths, where k is the number of demands and m is the number of edges in the network. This implies that if the number of demands is greater than the number of edges, *SimPol* provides a 2-approximation algorithm and on the average each demand uses less than two paths. We show a provable worst-case approximation factor for the algorithm. We extend this framework by using integer linear programming to accommodate two different network optimization applications.

The rest of the paper is organized as follows. Section II provides the basic problem formulation along with some related work. The polynomial time framework is presented in Section III. Extensions to this framework for other constraints and the integer programming formulation are described in Section IV. The practical applications of this framework is discussed in Section V. Simulation results are provided in Section VI, and is followed by a discussion of the results and conclusions in Section VII and Section VIII.

II. Problem Formulation

Consider an undirected network $G(V,E)$ of n nodes and m links. The current capacity of a link $e \in E$ is denoted by $c(e)$. This is the bandwidth that can be routed on link e. Also we are given a set of k demands. Each demand is defined by a triple (s_i, t_i, d_i), where s_i is the source (ingress router), t_i is the destination (egress router), and d_i is the bandwidth requested from s_i to t_i. Our goal is to find a set of paths or routes on which we can send all the given demands. In determining the set of paths, if we are restricted to assigning only one path for each demand, we call it an unsplittable flow. However, if we are allowed to send the flow for each demand on more than one path, we call it a splittable flow. As described in the introduction, better network performance can be achieved through the use of splittable flows. For network design, we want to use the best performance of the network in terms of congestion or total bandwidth consumption. One disadvantage of splittable flows is that we may end up with too many paths. Hence our goal is to minimize the total number of paths used to satisfy the given demands while ensuring the best performance of the network. The following objective functions are used to define the best performance of the network.

- **Minimizing Congestion *MinCon*:**
  In the *MinCon* problem, we want to minimize the maximum congestion on the links of the network. The congestion on any link e is defined as the ratio between bandwidth consumption on the edge $b(e)$ to the capacity of the edge $c(e)$. For minimum congestion ratio, we want to reduce $b(e)$ on any given edge.

- **Minimizing Bandwidth Consumption *MinBan*:** In *MinBan* we want to minimize the total bandwidth consumption on the network. A given traffic demand may be assigned to a single path or it could be balanced among multiple paths. The goal of this problem is to minimize the sum of all the bandwidth consumptions on all the edges of the network.

We call an algorithm A an (α, β)-approximation for *MinCon* problem if the congestion of the output of A is at most α times the optimum congestion and A has at most β times the minimum number of paths to route all demands (in minimum congestion). We use γ-approximation for *MinCon* instead of $(1, \gamma)$-approximation for *MinCon* according to the above definition.

A. Complexity Issues and Related Work

Both *MinCon* and *MinBan* problems are NP-complete. The *MinCon* problem is a generalization of the unsplittable flow problem [14], [6]. The *Unsplittable Flow Problem* (UFP) has been considered in several prior works. Klienberg [13] provides a comprehensive background on these problems. In fact, we state a stronger theorem, and show the tightness of our approximation factor.

Theorem 2.1: Given an undirected graph $G = (V, E)$ and a set of demand triples (s_i, t_i, d_i) $(1 \leq i \leq k)$, it is NP-Complete to decide if there is a multicommodity flow of congestion at most 1 which can be routed using $k + m - 2$ paths.

Proof: We give a straightforward reduction from the PARTITIONING problem. In PARTITIONING, we are given a set of numbers $\{d_1, d_2, \ldots, d_k\}$ such that $\sum_{1 \leq i \leq k} d_i = B$. The goal is to decide if there is a subset of d_i's whose summation is $\frac{B}{2}$. Given an instance of PARTITIONING, we construct an instance of *MinCon* such that there is a multicommodity flow of congestion less than 1 with at most $m + k - 2$ paths. Consider the graph $G = (V, E)$ with $V(G) = \{s, t\}$. There are two edges of capacity $\frac{B}{2}$ between s and t. Corresponding to each number d_i, we put a demand of value d_i between s and t i.e., a triple (s, t, d_i). Now, deciding if this flow is routable with a congestion of one via $k + m - 2$ paths is equivalent to solving the PARTITIONING problem. The result follows from the NP-completeness of PARTITIONING. ■

Many approximation algorithms for unsplittable flow problem are based on finding a splittable flow by solving a linear program. After solving a linear programming relaxation, we may end up with many paths for each commodity and we need to round the solution in a suitable way to obtain an approximation solution for UFP. Three main questions regarding UFP are: minimizing congestion, maximizing the routable demand, and obtaining the number of rounds to route all commodities unsplittably (given hard capacities). In order to minimize congestion, Raghavan and Thompson [22] give an $O(\log n)$ approximation algorithm using randomized rounding method. In fact, using Chernoff bound they proved that if we round the fractional flow to an unsplittable flow, we lose a factor of $O(\log n)$ in expectation. For maximizing routable demand, Guraswami et. al. [10] proved that UFP is hard to approximate within a factor of $m^{1/2-\epsilon}$ for any $\epsilon > 0$ in directed networks. Kolman and Scheideler [15] provide an $O(\sqrt{m})$ approximation via a greedy algorithm. In the case of single-source UFP where all demands are from a single source, constant factor approximations are known for all three of the above-mentioned problems. Dinitz, et.al., [6] have shown how to change a fractional flow to an unsplittable flow by violating each link by at most the maximum demand. Based on this algorithm, they have a 5-approximation for minimizing congestion, 4.43 approximation for maximizing the routable demand, and 5 approximation for minimizing the number of rounds to route all the demands. Note that this result implies a $(5, 5)$-approximation algorithm for *MinCon* in the case of a single-source unsplittable flow problem. For the single commodity case, Baier, et.al. [3] considered the maximum routable demand when there is a bound on the total number of paths. They have constant factor approximations for that problem. They also considered the multicommodity case when there is a bound on the number of used paths for each demand. Their objective function is to maximize the routable demand and is different from ours. The difference between *MinCon* and UFP is that we are not restricted to only one path for each demand and we want to use the best performance of the network. In *MinCon* the best performance corresponds to minimizing the congestion in the network while the total number of paths is not very large.

III. Simple Polynomial Time Framework : *SimPol*

In this section we propose a three step LP formulation framework that provides a polynomial time approximation algorithm for the *MinCon* and the *MinBan* problems. These problems can be formalized as linear programs in different ways. One way is to model the amount of flow according to demand j on edge e as the variable X_e^j. Now, it is sufficient to write flow preserving constraints on the nodes for each demand and also capacity constraints for each link. We call this LP, *edge-demand LP*. The size of this linear program (LP) is polynomial, and can be solved in polynomial time. The *edge-demand* LP is formulated as follows.

$$\text{Minimize} \qquad C \tag{1}$$
$$\sum_{1\le j\le k} X_e^j \le C c_e \qquad \forall e \in E(G) \tag{2}$$
$$\sum_{e=uv} X_e^j - \sum_{e=vu} X_e^j = 0 \quad \forall u \ne s_j \in V, j \in D \tag{3}$$
$$\sum_{e=s_jv} X_e^j - \sum_{e=vs_j} X_e^j = d_j \qquad \forall j \in D \tag{4}$$
$$X_e^j \ge 0 \quad \forall j \in D, e \in E(G) \tag{5}$$

where C corresponds to the maximum congestion on the edges, and is the parameter being minimized by this LP. X_e^j is the amount of flow on edge e corresponding to demand j. Constraint (2) corresponds to congestion on edge e and constraints (3) and (4) correspond to flow preserving constraints on the nodes.

Using this LP we can find the amount of flow on each edge corresponding to each demand. Each flow can be decomposed into a set of paths. After this decomposition, we may end up with at most m paths for each demand. Since we may remove one edge at each step of the decomposition, the number of paths for each demand is at most m.

Another way of formulating *MinCon* with linear programming is to assign a variable y_r^j to the amount of flow on each path r from s_j to t_j. We call this formulation the *route-demand* LP. Again, we write the capacity constraint for each edge. In this case the flow conservation constraints are captured by using a set of admissible paths R_j for each demand j from s_j to t_j. The *route-demand* LP is depicted below.

$$\text{Minimize} \qquad C \tag{6}$$
$$\sum_{r\in R_j} y_r^j = 1 \qquad \forall j \in D \tag{7}$$
$$\sum_{j\in D} \sum_{r\in R_j, e\in r} y_r^j d_j \le C c_e \qquad \forall e \in E(G) \tag{8}$$
$$y_r^j >= 0 \quad \forall j \in D, r \in R_j \tag{9}$$

where C again corresponds to the maximum congestion on the edges.

The flaw of the *route-demand* LP formulation is that the number of possible paths between s_j and t_j may be exponentially large. We can apply column generation techniques to solve this LP without writing the whole LP explicitly, i.e. we can find the basic feasible solution, check if they are optimum using a shortest path computation, and change the basis accordingly. However this technique does not solve the problem in polynomial time, and thus, a polynomial time algorithm is needed.

On the other hand, the advantage of the *route-demand* LP in terms of the number of paths, is that the number of constraints is only $m+k$. Therefore, there is an optimum solution corresponding to a basic feasible solution for this LP with at most $m+k$ nonzero variables. In other words, the number of paths used in this solution is at most $m+k$. In order to take the advantage of the *route-demand* LP and to resolve its flaw, we suggest the following polynomial time framework *SimPol*:

- **Initial flow:** Solve the *edge-demand* LP to get a flow X^j for each demand j.
- **Decomposition:** For each demand j, decompose X^j into a set of at most m paths. Call this set of paths for demand j, R_j.
- **Path selection** Find an optimum basic feasible solution to the *route-demand* LP with R_j as the set of candidate paths

Using this framework the size of the second LP is polynomial (and in fact very small). Furthermore, since we are using the same set of paths generated from the *edge-demand* LP, we can get the same solution in terms of optimum congestion. In addition, an optimal basic feasible solution for the second LP has at most $m+k$ nonzero variables. This means that we can route all the demands with the best possible congestion and use at most $m+k$ paths. Note that the bandwidth allocation for different paths obtained from this framework is not split equally among the different paths for a given demand. However, the bandwidth allocations obtained do achieve the minimum possible congestion.

The same framework can also be used to minimize the total bandwidth consumption *MinBan*. We can formulate the problem again using the *edge-demand* and *route-demand* LP's as shown below:

$$\text{Minimize} \qquad \textstyle\sum_{1\le j\le k, e\in E(G)} X_e^j \tag{10}$$
$$\sum_{1\le j\le k} X_e^j \le c_e \qquad \forall e \in E(G) \tag{11}$$
$$\sum_{e=uv} X_e^j - \sum_{e=vu} X_e^j = 0 \quad \forall u \ne s_j \in V, j \in D \tag{12}$$
$$\sum_{e=s_jv} X_e^j - \sum_{e=vs_j} X_e^j = d_j \qquad \forall j \in D \tag{13}$$
$$X_e^j \ge 0 \quad \forall j \in D, e \in E(G) \tag{14}$$

$$\text{Min} \quad \textstyle\sum_{1\le j\le k, r\in R(j)} d_j * l(r) * y_r^j \tag{15}$$

$$\sum_{r\in R_j} y_r^j = 1 \qquad \forall j \in D \tag{16}$$

$$\sum_{j\in D}\sum_{r\in R_j, e\in r} y_r^j d_j \le C c_e \qquad \forall e \in E(G) \tag{17}$$

where $l(r)$ for $r \in R(j)$ is the number of edges on route r.

We solve the first LP, decompose its solution, and feed the set of candidate paths to the second LP. By solving the second LP, we will get the best possible bandwidth consumption and the total number of used paths is at most $m+k$.

Theorem 3.1: All k demands can be routed with optimum network performance (in terms of minimum congestion and minimum bandwidth consumption) using at most $m+k$ paths, where m is the number of edges in the network. Furthermore this routing can be found in polynomial time. In particular, if the number of demands k, is large compared to the number of edges, namely if $k \ge m$, the above algorithm gives a 2-approximation algorithm for *MinCon*.

Proof: We observe that for each demand at least one path is required, thus the total number of paths is at least k. The result follows from above discussion. We discuss how to find a basic feasible solution in the subsection. ∎

For the greedy decomposition, we describe a proof of the above theorem which says that if we use a greedy decomposition, we can decompose a basic feasible solution of the *edge-demand* LP to at most $m+k$ paths. This proof is done by constructing a basic feasible solution to the *route-demand* LP, and uses insight from this LP to prove the result. We thank F. Bruce Shepherd for kindly providing this proof.

Consider the greedy flow decomposition of the flow X on edges. For each commodity X^i, we find a directed path P from s_i to t_i in the support of X_i. Let $\Delta_1 = \min_{a\in P} X_a^i$. We send Δ_1 flow down P_1 and reduce X^i accordingly. Then we repeat this process until we have decomposed X^i into flow paths $P_1^i, P_2^i, \dots, P_{l_i}^i$. Suppose we sent flow Δ_j^i on path P_j^i; hence $\sum_j \Delta_j^i = d_i$. Let the resulting path vector be $\mathbf{y}$.

We claim that after this decomposition, y is a basic feasible solution to the second LP, thus it has at most $m+k$ paths. To see this, suppose that $\mathbf{y}$ is not basic and so it can be written as $\frac{1}{2}(\hat{\mathbf{y}} + \tilde{\mathbf{y}})$, where $\hat{\mathbf{y}} \ne \tilde{\mathbf{y}}$. These two new solutions can be used to define solutions for the *edge-demand* LP as well. In particular for each commodity i and each arc a for instance define $\tilde{x}_a^i = \sum_{P\in\mathcal{P}_{s_i,t_i}} \tilde{y}^i(P)$. Similarly we may define another vector $\hat{\mathbf{x}}$ and clearly we then have that $\mathbf{x} = \frac{1}{2}(\hat{\mathbf{x}} + \tilde{\mathbf{x}})$. But then since $\mathbf{x}$ was basic, we must have $\hat{\mathbf{x}} = \tilde{\mathbf{x}}$, but we show that this is impossible by our choice of path decomposition. Indeed, we can find a minimum value j such that for some commodity i $\tilde{\mathbf{y}^i}(P_j^i) \ne \hat{\mathbf{y}^i}(P_j^i)$. One such value exists since $\hat{\mathbf{y}} \ne \tilde{\mathbf{y}}$. But then by our choice of the P_j^i's, there is an arc $a \in P_j^i$ that does not lie on any $P_{j'}^i$ with $j' > j$. But then $\tilde{x}_a^i \ne \hat{x}_a^i$. This is a contradiction with the way we choose our paths.

A. Finding the basic feasible solution

See Appendix.

IV. Extensions To the *SimPol* Framework

In this section we describe extensions to the *SimPol* framework that shows its usefulness for a general network optimization problem.

A. Integer Programming Formulation for Congestion vs. Number of Paths Tradeoff

In the last section, using the *route-demand* LP we provide an easy way to decompose a given multicommodity flow into a few number of paths in polynomial time. We can write an integer program that allows the evaluation of tradeoff issues between congestion and the total number of paths used. The integer program can be written as follows:

$$\text{Minimize} \quad \textstyle\sum_{j\in D, r\in R_j} z_r^j \tag{18}$$

$$\sum_{r\in R_j} y_r^j = 1 \qquad \forall j \in D \tag{19}$$

$$\sum_{j\in D}\sum_{r\in R_j, e\in r} y_r^j d_j \le \alpha C c_e \qquad \forall e \in E(G) \tag{20}$$

$$z_r^j \ge y_r^j \ge 0, z_r^j \in \{0,1\} \quad \forall j \in D, r \in R_j \tag{21}$$

where z_r^j is the zero-one variables, which is one if and only if path r is used i.e., $y_r^j > 0$. The parameter to be optimized is the total number of used paths. C is the best possible congestion obtained from the *edge-demand* LP. We can set the parameter α to be of any desired value. Thus for a given congestion αC, we can guarantee that the the solution to the above LP results in a feasible set of paths. We can also compromise on the congestion factor depending on our application and get better solutions in terms of the number of paths.

Notice that we cannot formalize our objective function i.e., the number of paths in the *edge-demand* LP-formulation. On the other hand, the size of the above integer linear program is very crucial in terms of running time, therefore we cannot consider all possible (exponentially many) candidate paths for each demand. Instead, we first run the *edge-demand* LP, find a set of paths, solve the *route-demand* LP to find a new set of paths, and feed it to the above integer program. This reduces the size (total number of paths) of the integer program substantially.

- **Initial flow:** Solve the *edge-demand* LP.
- **Decomposition:** Decompose this flow into a set of paths for each demand j, R_j.
- **Path selection** Solve the *route-demand* LP to find a new set of candidate paths for each demand, R'_j for demand j.
- **Integer linear program** Relax the congestion constraint by *a desirable* factor, $\alpha > 1$. Solve the above integer linear program to minimize number of paths using R'_j as the set of candidate paths.

Using this method, we achieved a significant improvement on the total number of paths while maintaining near optimum congestion.

B. Limited Bandwidth assignment: LimBan

In addition to the problem of minimizing congestion we consider another constraint where a single path can not be assigned more than a fraction β of the available bandwidth on each edge. Thus, the factor β aims at keeping the path bandwidth relatively small compared to the edge capacities in the network. To achieve this, it may be necessary to split a path into two or more paths. We call this multicommodity flow problem with the new constraint *LimBan* problem. In an instance of *LimBan* problem, in addition to the network, capacities on the edges and demand triples, we are given a constant β. A solution to *LimBan* is to find a set of paths that satisfies all demands while minimizing the maximum edge congestion. The solution must also satisfy the above bandwidth allocation constraint using only a small number of paths. From now on, we assume that $\frac{1}{\beta}$ is an integer number.

First we observe that the number of paths in the *LimBan* problem is dependent on the size of the demand as compared to the minimum link capacity. Notice we may need $\frac{d_{max}}{C_{min}}$ paths in the general case, where d_{max} is the maximum demand on a path and C_{min} is minimum link capacity in the network. In the case of $\beta = 1$ the *LimBan* problem is related to the problem of minimizing the number of rounds [14]. There is no known constant factor approximation for the multi-commodity case but exists in the single source case [6]. Therefore we make the natural assumption that $d_{max} \leq C_{min}$ [14]. Now we can show that this new problem *LimBan* can be reduced to the *MinCon* problem by splitting the edges of G to form a new graph G'. Notice that given this assumption, *MinCon* is the same as the *LimBan* problem with $\beta = 1$. For the remainder of the discussions on *LimBan* the above assumption on the size of the demands ($d_{max} \leq C_{min}$) holds.

Given an instance $G(V, E)$ with capacity function $c : E(G) \to N$, we construct a network $G'(V, E)$ as follows: G' has the same set of nodes i.e., $V(G') = V(G)$ and corresponding to each edge $e \in E(G)$ we put $t = \lceil \frac{1}{\beta} \rceil$ parallel edges $e_1, \ldots, e_t$ with capacities $c(e_i) = \beta c(e)$ for $1 \leq i \leq t$ ($\frac{1}{\beta}$ is integer). Clearly, any solution to *MinCon* problem on G' is a feasible solution of *LimBan* on G. From Theorem 3.1, we know that there exists an upper bound on the number of paths $(k + \frac{1}{\beta}m)$ for the best possible congestion on G'. The following theorem shows that the best possible congestion for *MinCon* on G' is at most twice the best possible congestion for *LimBan* on G.

Theorem 4.1: Using the above definition of G and G', if $\mathcal{P}$ is a feasible solution of *LimBan* on G with congestion C consisting of p paths, there is a feasible solution for the problem *MinCon* on G' with congestion at most $2C$ and at most p paths.

See Appendix for proof. The above theorem shows that there is a theoretical upper bound on the number of paths for *LimBan* on G using at most $k + \frac{1}{\beta}m$ paths with a congestion of $2C$. It means that we may not get satisfactory results in the worst case using this method. Therefore we extend the *SimPol* framework to solve *LimBan* by formulating an integer linear program.

Integer Programming Formulation

Here, we describe another application of our framework to address *LimBan* problem. We can formalize *LimBan* problem as the following integer linear program:

$$\text{Minimize} \quad \textstyle\sum_{j \in D, r \in R_j} z_r^j \tag{22}$$

$$\sum_{r \in R_j} y_r^j = 1 \qquad \forall j \in D \tag{23}$$

$$\sum_{j \in D} \sum_{r \in R_j, e \in r} y_r^j d_j \leq \alpha C c_e \qquad \forall e \in E(G) \tag{24}$$

$$y_r^j * d(j) * b(r)^{-1} \leq z_r^j \quad \forall j \in D, r \in R_j \tag{25}$$

$$y_r^j \geq 0, z_r^j \in N \quad \forall j \in D, r \in R_j \tag{26}$$

where $z_r^j = \lceil \frac{y_r^j * d(j)}{b(r)} \rceil$ and $b(r)$ is the maximum bandwidth we can assign to a path on route r. The parameter to be optimized is the total number of used paths. C is the optimum congestion obtained from the *edge-demand* LP. Since the maximum allowable bandwidth for each path depends on all edges on the path, the *edge-demand* LP cannot capture the objective function for the *LimBan* problem. The above integer program allows us to consider the maximum allowable bandwidth on different paths separately and distribute the flow among different paths depending on their maximum allowable bandwidth. In our simulation results we compare the results of this integer linear program with a naive manual decomposition of the flows obtained by the *SimPol* framework.

V. Applications

In the following section, we describe potential ways of utilizing the *Simpol* framework and its extensions to address practical problems that Network Service Providers, or network operators, face in the planning and operation of their networks. Note that a network operator would need to choose a subset of these applications based on the specific needs since some of these applications have conflicting goals.

- **Number of paths in MPLS routing:** There is a tradeoff between the number of paths and the network's operational complexity. The set-up and maintenance of paths result in overhead for the network operator. When there is more than one path for a given endpoint pair, it may be necessary to perform unequal traffic splitting at the network ingress to do load balancing between these paths. As pointed to in [23], unequal splitting can be provided by manually setting up rules that relate the routing prefixes to particular paths. This process requires traffic measurements at the level of routing prefixes, and increases the network operations cost. Furthermore, it may not be possible to achieve the ideal split of the traffic, resulting in non-optimal routing behavior. Once a path is set-up, traffic measurements through SNMP or similar mechanisms is essential to make sure that the network is operating as planned to provide the required service quality to its customers. Hence, reducing the number of paths reduces the network's operational cost. The *Simpol* framework provides a small number of paths for a given traffic demand matrix without compromising on maximum congestion in the network.
- **Flexible tuning of number of paths for optimal capacity planning:** A network operator would want to further reduce the number of paths at the expense of increased congestion at some of the links. The increase in congestion would be acceptable especially if the network has been overprovisioned. In such a network, the main constraint would be to reduce the operations cost. The integer programming extension, introduced in Section IV, addresses this tradeoff through the parameter α. For a given network, proper choice of α would result in only a single path per demand, thus avoiding the complicated task of manually setting-up special filters to provide path selection based on routing prefixes. This result is also useful even if the network operator prefers to adapt the IGP metrics to reflect the path layout in an IP network without actually setting up the paths as described in [23].
- **Incorporate policies in path selection:** Limited bandwidth assignment framework, introduced in Section IV, prevents a single path from dominating the total bandwidth of a link. The factor β aims at keeping the path bandwidth relatively small compared to the link sizes to avoid reroute failures. This constraint is useful to handle link failures or link maintenance, where a network operator takes the link down for maintenance purposes. In either case, all paths going through the affected link have to be rerouted. Rerouting can be performed automatically by the originating node of the path or performed manually by the network operator. When a path with large bandwidth is to be rerouted, it is possible that there would be no single end-to-end path with the required amount of residual bandwidth. This leads to a reroute failure, which can be handled by manually splitting the original path demand into multiple smaller paths and attempting to route these demands sequentially. This solution is only possible for network operator-initiated rerouting attempts, and would increase the network's operational cost. If an edge-router-initiated re-routing attempt fails, the communication between the path endpoints will be interrupted. Our formulation solves this problem during the path design phase, and alleviates the problem of manually splitting paths during failures or maintenance downtimes. Note that this approach increases the number of paths per demand to address the issue of successful path restoration.
- **Applications of un-reserved capacity:** The *MinCon* problem aims at minimizing the maximum congestion on the links of the network. This optimization in fact maximizes the unreserved bandwidth at the most congested link. Note that this is important since the unreserved bandwidth can be used in a number of ways. For example, the network operator would decide to resize the paths based on traffic measurements. Reducing the maximum congestion increases the chance of allocating more bandwidth to paths without having to change the routing due to

insufficient bandwidth. Furthermore, to handle link failures, some paths may need to be rerouted. The *MinCon* problem increases the available bandwidth for restoration paths. Another application for the unreserved bandwidth is to handle traffic that does not flow through the established paths. For example, the paths can be used for traffic that requires a certain Quality of Service (QoS), and the best-effort traffic would use the unreserved bandwidth over the network. The *MinCon* problem results in more bandwidth available for best-effort traffic over the links with maximum congestion. Similarly, network operators could establish new paths to handle short-lived traffic using the unreserved bandwidth in the network. An algorithm like MIRA [11] can be used to set up paths using the unreserved bandwidth over the network.

VI. Simulation Results

The performance of the *SimPol* algorithm was studied on several different simulated topologies. *SimPol* has been evaluated on known topologies that have been used in previous work. Most of these topologies are relatively small and provide an intuitive understanding of our framework. We also generated large topologies using BRITE [17]. The BRITE topology generator uses a Waxman model to generate flat topologies [29]. The model parameters used were $alpha = 0.15$ and $beta = 0.20$, where $alpha$ captures the relationship between short and long edges and $beta$ measures the degree of connectivity in the network. We generated two sets of topologies, the first set had 100 nodes with 200 bidirectional edges and the second set had 60 nodes with 120 bidirectional edges. The linear programs in *SimPol* were solved using the commercially available CPLEX solver [5]. We used CPLEX since it is based on simplex and automatically produces basic solutions for an LP. In our simulation scenarios, the traffic demands are uni-directional with equal bandwidth requirement for a given endpoint pair in opposite directions. The framework also works well for asymmetric demands. The edge capacities are obtained from a uniform distribution over the range of [50, 200] bandwidth units. In each simulation scenario, we report the total number of paths obtained in the *edge-demand* LP, in the *route-demand* LP, and in the integer program. We also report on the congestion in the network. In some cases, the value of congestion is greater than 1. This congestion is a scalable factor of the demand values and does not affect our path layout scheme. It is only used as a relative measure of congestion for comparing with shortest path.

Top.	m	k	C	LP(1)	LP(2)	IP(1.1)	IP(1.5)
Reg	58	8	1.27	13	12	12	10
KL	56	90	2.13	111	101	93	90
NSF	20	56	1.43	60	60	57	56
US	58	30	4.07	34	33	30	30

TABLE I

Performance of *SimPol* on known topologies. m: Number of directed edges, k: Number of demands, C: Congestion, LP1: *edge-demand*, LP(2):*route-demand*, IP(1.1), IP(1.5) Integer Programs with $\alpha = 1.1, 1.5$

A. SimPol *on Known Topologies*

SimPol was run on 4 different known topologies: (1) a *regular* topology that has a lattice like structure [31], (2) the KL topology [11], (3) NSF-Net map topology [18] and (4) a US map topology [20]. Table I shows the results for the known topologies. First we would like to point out that, as expected, from Theorem 3.1 the total number of paths obtained using the *route-demand* LP is always less than $k + m$. C represents the best possible congestion that could be obtained for this network.

It is interesting to note that the *edge-demand* and the *route-demand* LP's have comparable, if not the same number of paths. This is due to the fact that the simplex solver in the *edge-demand* LP returns a *bfs* for the *route-demand* LP (proof shown in Section II). By increasing the congestion by 10% we observe that on the average we have a gain of 6% in terms of the number of paths. We see that for these topologies our *SimPol* (even without integer program step) framework performs extremely well.

B. SimPol *on Large Topologies*

In each of our large topology examples we used two types of demand matrices. In one case we used a random demand matrix where the bandwidth for each traffic demand was drawn from a uniform distribution over a range of [5, 40] bandwidth units. We also studied a fixed demand case where all source-destination pairs have the same demand of 22 bandwidth units. The number of demands in each case varied as 40, 200, and 400 demands. The results for the 60-node and 100-node networks with random demand distribution is shown in Table II.

Again we see that the total number of paths from the *route-demand* LP is less than $k + m$. Interestingly, for large topologies, we notice that the route-demand LP does improve the number of paths as compared to the *edge-demand* LP by about 12%. Therefore, doing the second LP improves by this factor, even though the

n	m	k	C	LP(1)	LP(2)	IP(1.1)	IP(1.5)
60	240	20	0.15	54	48	37	28
		60	0.37	128	104	79	63
		100	0.43	193	164	122	101
		140	0.57	242	202	157	140
		180	0.83	254	224	191	180
		220	0.94	310	271	226	220
		260	1.31	328	291	262	260
		300	1.41	380	338	302	300
		340	1.41	430	402	345	340
100	400	40	0.35	67	52	50	43
		200	0.87	293	251	235	200
		400	1.07	571	519	415	400

TABLE II

PERFORMANCE OF *Simpol* WITH RANDOM TRAFFIC DEMAND MATRIX.n: NUMBER OF NODES, m: NUMBER OF DIRECTED EDGES, k: NUMBER OF DEMANDS, C: CONGESTION, LP1: *edge-demand*, LP(2):*route-demand*, IP(1.1), IP(1.5): INTEGER PROGRAMS WITH $\alpha = 1.1, 1.5$

k	MLP(1)	C	LB	C	% gain in paths
20	54	0.15	34	0.19	37
60	147	0.31	109	0.39	26
100	223	0.43	156	0.54	30
140	270	0.57	201	0.71	26
180	319	0.83	263	1.04	26
220	385	0.94	318	1.18	17
260	489	1.31	444	1.64	9
300	560	1.41	502	1.76	10
340	501	1.41	428	1.76	15

TABLE III

PERFORMANCE OF *LimBan* WITH RANDOM TRAFFIC DEMAND MATRIX.NUMBER OF NODES = 60, NUMBER OF EDGES = 120, $\alpha = 1.25$, k: NUMBER OF DEMANDS, C: CONGESTION, MLP: MODIFIED *edge-demand* LP, LB:*LimBan* IP

provable worst case number is $m + k$ in both cases. Also we see that by compromising on the congestion it is possible to reduce the total number of paths. For a congestion factor of 1.1, we get a gain of 26% in the number of paths while for a congestion factor of 1.5 we get a gain of 34%. It is interesting to note that by increasing the congestion to a factor of 1.5 we observe that for large demands we only use one path per demand. Furthermore as the number of demands increases we observe that even with a slight increase in congestion to 1.1 we can get almost 1 path per demand. For the fixed demand case the results are tabulated in Table V in the Appendix. From Table V we see that in a 60-node network even for a small increase in congestion of 10%, in the case of large demands we obtain only one path per demand. On the average, the improvement in the *route-demand* LP is about 14% while it is 27% and 35% for the congestion factors of 1.1 and 1.5, respectively.

C. *Application of* SimPol *for* LimBan

The *SimPol* framework was extended to the *LimBan* application. *LimBan* limits the bandwidth of a path over a link to a certain fraction α of the link bandwidth. As described in Section IV, we incorporate the bandwidth restriction explicitly by writing an integer program in the second step of the *SimPol* framework. We compare the number of paths obtained from the *LimBan* integer program to that obtained by manually decomposing the *edge-demand* LP. In the manual decomposition we solve the *MinCon* problem and selectively decompose only those paths that violate the bandwidth constraint. The results are tabulated in Table III. Note that both of these methods (manual path selection and integer programming based algorithm) give better results compared to the case of splitting every edge into $\frac{1}{\beta}$ edges and solving the *MinCon* problem on that network.

Notice that since we allow splitting the paths obtained from the *edge-demand* LP, the upper bound $k + m$ on the number of paths does not hold. When compared to the manual decomposition, we get about 22% reduction in the number of paths by explicitly taking into consideration β, the bandwidth restriction on the path. This reduction in the number of paths is obtained by allowing an increase in congestion to 1.25 times the best possible congestion. Note that the change in the percentage gain for different number of demands is due to the maximum demand value and the minimum link bandwidth, both of which are randomly chosen in the simulated scenarios.

VII. DISCUSSION

In this section we provide a discussion on the performance of the *SimPol* framework as well as the extensions to the framework. As described in our simulation results, the overall network performance in terms of congestion can be significantly improved by using multi-path routing. The framework presented in this paper is a useful tool to evaluate tradeoffs between network congestion and the number of paths required per demand.

Comparison with *Shortest-Path routing*: The goal of the *SimPol* framework was to optimize network performance in terms of minimizing congestion using a small number of paths per demand. Table IV compares the performance of *SimPol* with respect to *shortest-path* routing (minimum-hop routing). The Table shows results for both the fixed and random demand

k	C	C_D	C_{SP}	Imp. in C
fixed				
200	1.23	1.84	3.52	1.91
400	1.41	2.1	5.72	2.72
Random				
200	0.87	1.3	2.78	2.13
400	1.07	1.6	5.08	3.17

TABLE IV

COMPARISON OF *Simpol* WITH *SP: Shortest Path* SCHEME. 200 NODE NETWORK. RANDOM AND FIXED TRAFFIC. k: NUMBER OF DEMANDS, C: BEST CONGESTION, C_D: CONGESTION FOR $\alpha = 1.5$ AND NUMBER OF PATHS = NUMBER OF DEMANDS, C_{SP}: SHORTEST PATH CONGESTION

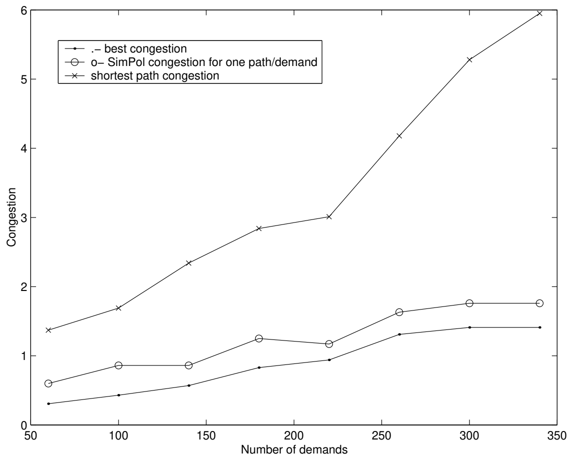

Fig. 1. Comparison of congestion in the *SimPol* framework with respect to *shortest-path* routing. 60-node network with random demand matrix.

cases for the 200 node network. The performance was similar in the case of the 60 node network and is shown in Figure 1.

A comparison of the congestion incurred in the shortest path scheme versus the *SimPol* framework for the 60-node network is shown in Figure 1. From Figure 1, the simulation results, and the Table IV, we observe that for a 65% increase in the number of paths the congestion in the network can be reduced to less than 4 times the congestion in *shortest-path* routing. When the number of paths is equal to the number of demands, the congestion from the *SimPol* framework (using a value of 1.5 for α) is on the average 2.5 times less congested than in the *shortest-path* scheme. Note that this congestion factor could be improved by a smaller value of α in the range of $(1, 1.5)$. For the demands shown here, we see that using a slight trade off in congestion we were able to obtain a single path for each of the demands. The congestion from the *SimPol* framework closely follows the best possible congestion. For this graph, we checked four different values of α and put the best value in which we get exactly one path per demand.

It is clear that in every case the *SimPol* framework out-performs the *Shortest-path* routing. Also with increasing number of demands the *SimPol* framework is more stable unlike the *Shortest-path* scheme which tends to blow up in congestion for larger demands.

Running time of *SimPol*

See Appendix.

Evaluation of the Integer Programming Formulation for Congestion Tradeoff:

Figures 2 and 3 present a comparison of the *route-demand LP*, integer program with congestion factors of 1.1 and 1.5, and *shortest-path* routing. The comparison is in terms of the number of paths obtained by each of these algorithms as a function of the number of demands. For the 100-node fixed demand case, *route-demand* LP gives the best improvement in terms of percent gain in the number of paths. In the integer programming framework, for the 100-node network, we see that there is no difference in performance between the fixed and random traffic demands. For larger increases in congestion we notice that in the 60-node network, the improvement in the fixed traffic demand is better than in the random demand case. For both 60- and 100-node networks, regardless of the randomness of the traffic demand, the *route-demand* LP has the highest number of paths for all demands. The number of paths is reduced in most cases through the use of integer program. As the congestion factor is increased, we notice that the number of paths tends to exactly one path per demand. From Figures 2 and 6, we see that using the integer programming framework, when the number of demands is large compared to the number of edges in the graph, we can route all demands using a single path per demand. This is true even for a 10% increase in congestion. However, in the case of small number of demands, we obtain one path per demand by losing 50% in congestion. Figure 4 summarizes the results as the percentage gain in the number of paths with respect to *edge-demand* LP for *route-demand* LP and integer program with congestion factors of 1.1 and 1.5. The impact of trading of the congestion factor in terms of the number of paths is shown in Figure 5. Notice that when the congestion is increased to some critical value it is possible to obtain one or almost one path per demand.

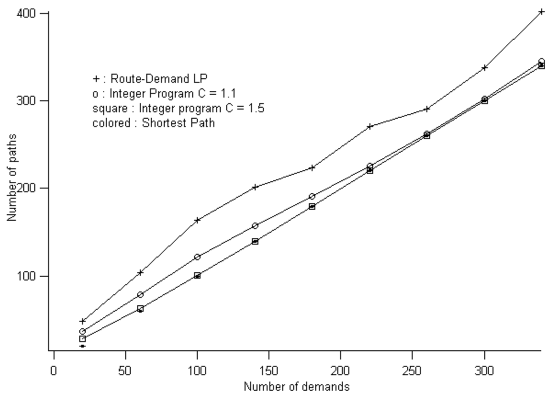

Fig. 2. Graphical Comparison of the *SimPol* framework with respect to *shortest-path* routing. 60-node network with random demand matrix

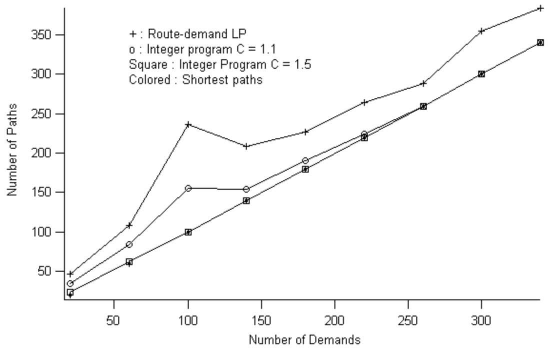

Fig. 3. Graphical Comparison of the *SimPol* framework with respect to *shortest-path* routing. 60-node network with fixed demand matrix

Policy Assignment Schemes:
LimBan is an example of using the *SimPol* framework to incorporate path/link based policy constraints [24]. One of the advantages of using *SimPol* is that we can add new optimization functions through the use of integer linear program formulations. *SimPol* provides a small number of candidate paths that optimize network performance. This candidate set can be used for any additional optimization functions. Using a small number of candidate paths can significantly improve the running time of the integer programs.

Open Theory Problems:
SimPol gives a 2-approximation algorithm for the case of large number of demands. It would be interesting to get constant factor approximations for the *MinCon* problem for small number of demands. As mentioned in Section II, a 5-approximation [6] is known for the single-source unsplittable case. Getting constant factor for the multi-commodity case is of theoretical interest. The problem of decomposing a given flow with the minimum number of paths is also an interesting question. Notice that in

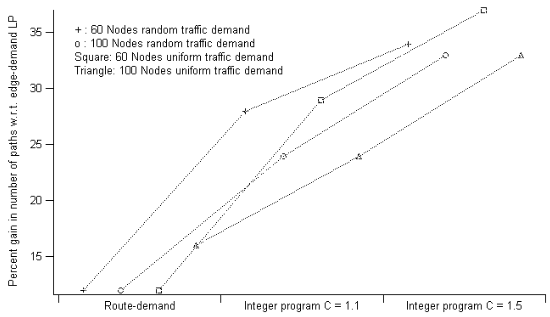

Fig. 4. Graphical Comparison of the steps of the *SimPol* framework. uniform=fixed demand

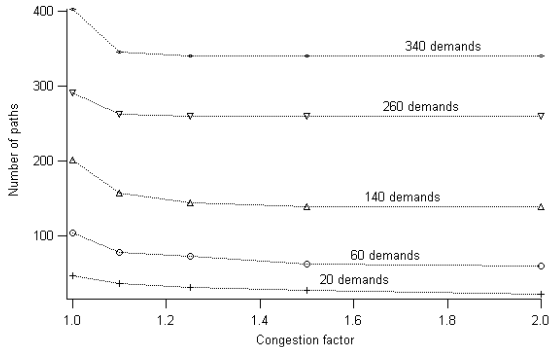

Fig. 5. Comparison of the steps in the *SimPol* framework in terms of the % gain in the number of paths

this work we find a flow and then decompose it into a set of paths. But if the flow is given, then we cannot use the property of the basic feasible solution. It would be nice to design approximation algorithms to decompose a given (multicommodity)-flow to the minimum number of paths. Yet, another issue is to consider the online case, when demands change or arrive at different times. To the best of our knowledge, there is no provable approximation algorithms for this problem. In this paper we address the problem in the static planning stage where we assume that all demands are known. It would be interesting to consider the case of minimizing the number of paths in the presence of average or worst case uncertainty.

VIII. Conclusion

In this work we have presented a polynomial time framework *SimPol* for multi-path routing or the splittable multi-commodity flow problem. The framework obtains a path lay-out scheme that minimizes congestion in the network using only a few paths. Using this framework we show a provable upper bound of $k + m$ paths. This implies that when the number of demands is greater than the number of edges, then on the average each demand uses less than 2 paths. Working on this problem has raised a number of interesting new theory problems that have been discussed in detail in Section VII.

The *SimPol* framework presented here is versatile and is able to naturally account for various policies that arise in the network planning process. In this paper we show two specific applications: congestion trade-off and limited bandwidth assignment strategy. We show that by using an effective trade-off in terms of congestion we can get almost one path per demand. It was observed that even increasing the congestion obtained by *SimPol* by a factor of 1.5 it still performs better than the *Shortest-path* routing scheme in terms of congestion. The *SimPol* framework can be incorporated as a modular tool into a network planning system. The framework is general and can be applied to any transport network.

IX. Acknowledgements

The authors would like to thank F. B. Shepherd for providing the proof on the total number of paths after greedy decomposition of the *edge-demand* LP. We would also like to thank David Houck and Mikkel Thorup for useful discussions on the problem and Mohammad Mahdian for providing access to the CPLEX solver. This work was done while the first author was on an internship at the networking center in Bell Labs Research.

References

[1] K. Ambs, S. Cwilich, M. Deng, D. J. Houck, D. F. Lynch, and D. Yan. Optimizing restoration capacity in the at&t network. *Interfaces*, 30(1):26–44, Jan - Feb 2000.

[2] D Applegate and M. Thorup. Load optimal mpls routing with $n + m$ labels. In *Proc. of IEEE Infocom*, 2003.

[3] G. Baier, E Kohler, and M. Skutella. On the k-splittable flow problem. In *In the proceeding of ESA 2002, 10th Annual European Symposium of Algorithms*, pages 101–113, 2002.

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

[5] ILOG CPLEX. http://www.ilog.com/products/cplex/.

[6] Y. Dinitz, N. Garg, and M. X. Goemans. On the single-source unsplittable flow problem. In *IEEE Symposium on Foundations of Computer Science(FOCS)*, pages 290–299, 1998.

[7] A. Dwivedi and R. E. Wagner. Traffic model for usa long distance optimal network. *Proc. Optical Fiber Communication Conference*, pages 156–158, 2000.

[8] A. Feldmann, A. Greenberg, C. Lund, N. Reingold, J. Rexford, and F. True. Deriving traffic demands for operational ip networks: Methodology and experience. *IEEE/ACM Transactions on Networking*, 9, 2001.

[9] A. Gupta, A. Kumar, and R. Rastogi. Exploring the trade-off between label size and stack depth in mpls routing. *Proc. IEEE INFOCOM*, 2003.

[10] V. Guruswami, S. Khanna, B. Rajaraman, F. B. Shepherd, and M. Yannakakis. Near-optimal hard-ness results and approximation algorithms for edge-disjoint paths and related problems. In *Proc. of ACM STOC*, pages 19–28, 1999.

[11] K. Kar, M. Kodialam, and T. V. Lakshman. Minimum interference routing with applications to (mpls) traffic engineering. *Proc. IEEE INFOCOM*, pages 884 – 893, 2000.

[12] N. Karmarkar. A new polynomial-time algorithm for linear programming. *Combinatorica*, 4:373–395, 1984.

[13] J. Kleinberg. *Approximation algorithms for disjoint paths problems*. Phd Thesis, MIT, 1996.

[14] J. M. Kleinberg. Single-source unsplittable flow problem. In *IEEE Symposium on Foundations of Computer Science(FOCS)*, pages 68–77, 1996.

[15] P. Kolman and C. Scheideler. Improved bounds for the unsplittable flow problem. In *Proceedings of the 13th Annual ACM–SIAM Symposium on Discrete Algorithms SODA'02*, pages 184–193, 2002.

[16] L. Li, M. Thottan, B. Yao, and S. Paul. Distributed network monitoring with bounded link utilization. *Proc. IEEE INFOCOM*, 2003.

[17] A. Medina, A. Lakhina, I. Matta, and J. Byers. Brite: An approach to universal topology generation. *Proc. MASCOTS*, Aug 2001.

[18] D. Mitra, J. A. Morrison, and K. G. Ramakrishnan. Atm network design and optimization: A multirate loss network framework. *IEEE/ACM Transactions on Networking*, 4(4):531–543, Aug 1996.

[19] D. Mitra and K. G. Ramakrishnan. A case study of multiservice, multipriority traffic engineering design for data networks. *Proc. IEEE GLOBECOM*, pages 1077–1083, Dec, 1999.

[20] S. Nelakuditi and Z. Zhang. On selection of paths for multi-path routing. *IWQoS*, June 2001.

[21] C. H. Papadimitriou and K. Steiglitz. *Combinatorial Optimization: Algorithms and Complexit*. Dover, 1998.

[22] P. Raghavan and C. Thompson. Randomized rounding: A technique for provably good algorithms. *Combinatorica*, 7:365–374, 1987.

[23] A. Sridharan, R. Guerin, and C. Diot. Achieving near-optimal traffic engineering solutions for current ospf/is-is networks. *Proc. IEEE INFOCOM*, 2003.

[24] S. Suri, M. Waldvogel, D. Bauer, and P. R. Warkhede. Profile-based routing and traffic engineering. *Computer Communications*, 25, 2002.

[25] Eva Tardos. A strongly polynomial algorithm to solve combinatorial linear programs. *Operation Research*, 34:250–256, 1986.

[26] M. Thottan, K. Swanson, M. Cantone, T. K. Ho, J. Ren, and S. Paul. Sequin: An snmp-based mpls, diffserv network monitoring system. *Bell Labs Technical Journal*, Aug 2003.

[27] Pravin. M. Vaidya. Speeding-up linear programming using fast matrix multiplication. In *In the proceeding 30th symp. of Foundations of Computer Science FOCS*, pages 332–337, 1989.

[28] C. Villamizar. Mpls optimized multipath mpls-omp. Internet draft, 1999.

[29] B. M. Waxman. Routing of multi-point connections. *IEEE JSAC*, 6(9):1617–1622, 1988.

[30] X. Xiao, A. Hannan, B. Bailey, and L. Ni. Traffic engineering with mpls in the internet. *IEEE Network Magazine*, pages 28–33, Mar 2000.

[31] S. Yilmaz and I. Matta. On the scalability tradeoffs in mpls and ip routing. *Technical Report Boston University, Computer Science Department*, 2002.

Appendix

Proof of theorem 4.1

Proof: Since $\mathcal{P}$ is a feasible solution of *LimBan* on G, each path $r \in \mathcal{P}$ can be mapped on the corresponding edges of G' without exceeding the capacity of the edges on the path. Consider an edge e and paths $r_1, r_2, \ldots, r_k \in \mathcal{P}$ that contains e. In G' we should assign each path r_i $(1 \leq i \leq k)$ to an edge e_j $(1 \leq j \leq t)$. We assign $r_1, r_2, \ldots, r_p$ to e_1 such that $\sum_{1 \leq i \leq p} cap(r_i) \geq c(e_1)$ and $\sum_{1 \leq i \leq p-1} cap(r_i) < c(e_1)$ where $ban(r_i)$ is the bandwidth assigned to path r_i. Using the fact that $ban(r_i) < \beta c(e) = c(e_i)$, $\sum_{1 \leq i \leq p} ban(r_i) < 2c(e_i)$. If we continue assigning these paths to e_i's in the same way, the total congestion on each edge e_i is at most twice the congestion on edge e. ■

Finding Basic Feasible Solutions(Subsection III-A) For the above result, we need to find a basic feasible solution for the *route-demand* LP. This is a well-known concept in the theory of linear programming. Corresponding to each variable in the linear program, there is a column vector of that variable's coefficients. Consider the linear program $Ax = b$. Suppose that there are m linearly independent columns A_j of A i.e., rank$(A) = m$. A basis of A is the set of m linearly independent collection of columns $\mathcal{B} = \{A_{k_1}, \ldots, A_{k_m}\}$. The basic solution corresponding to the basis $\mathcal{B}$ is to set to zero all variables whose column is not in $\mathcal{B}$ and solve the remaining full-rank system to find the basic variables in the basis. In other words, if B is a nonsingular matrix for $\mathcal{B}$, then $x_i = 0$ if $A_i \notin \mathcal{B}$ and $x_{k_t} =$ the t'th column of $B^{-1}b$. A basic feasible solution (bfs) is the solution that satisfies the equation $Ax = b$. It is straightforward to see from the above definition that the number of nonzero variables in the bfs is at most m, the number of constraints.

For a short and complete description of how to find a bfs refer to Applegate and Thorup [2]. We know that the simplex algorithm finds basic feasible solutions. Linear programming solvers like CPLEX [5] use simplex to solve linear programs. We use CPLEX as the LP solver to get our experimental results. However, we cannot use simplex for the theoretical worst-case polynomial time algorithm. We can use the interior-point algorithm and a crossover algorithm to get a basic solution [12], [27]. This algorithm is not strongly polynomial. As for a strongly polynomial time algorithm, we can use Tardos's algorithm [25].

Running time of *SimPol*:
The running time for the *SimPol* framework is polynomial. It is a function of the number of demands and the number of edges in the network. In our simulations, for the case of 100 node and 400 demand the *edge-demand* LP takes about a few seconds. The *route-demand* LP takes only a few milli seconds. The integer programs are not polynomial, however since it uses the candidate paths from the LP it is possible to get reasonable running times. This is one of the benefits of the *SimPol* framework since it allows the formulation of several policy constraints as integer programs that can be solved reasonably quickly. In the case of the congestion tradeoff integer linear program, we see that depending on the value of the congestion factor α we have different running times. For $\alpha > 1.1$ the running time is in the order of seconds while for $\alpha < 1.1$ the running time might be in the order of minutes. However, in the reported results we terminated the program after at most five minutes. Note that for smaller values of α the number of paths increases and therefore the IP takes longer to complete. This is inherent to branch and bound technique used in CPLEX [5], [21].

n	m	k	C	LP(1)	LP(2)	IP(1.1)	IP(1.5)
60	240	20	0.20	57	47	34	24
		60	0.27	134	108	84	62
		100	0.31	252	236	155	100
		140	0.56	231	209	154	140
		180	0.75	262	227	191	180
		220	0.94	297	264	224	220
		260	1.22	322	289	260	260
		300	1.22	392	355	301	300
		340	1.41	427	384	340	340
100	400	40	0.18	146	110	90	57
		200	1.23	243	217	207	200
		400	1.41	509	443	411	400

TABLE V

PERFORMANCE OF *Simpol* WITH FIXED TRAFFIC DEMAND. n: NUMBER OF NODES, m: NUMBER OF EDGES, k: NUMBER OF DEMANDS, C: CONGESTION, LP1: *edge-demand*, LP(2):*route-demand*, IP(1.1), IP(1.5): INTEGER PROGRAMS WITH $\alpha = 1.1, 1.5$

Additional Results For Discussion Section: Figures and Tables

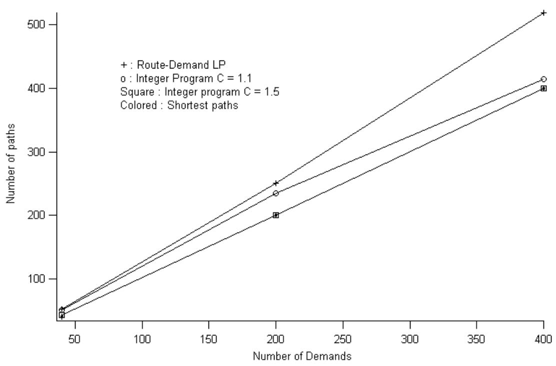

Fig. 6. Graphical Comparison of the *SimPol* framework with respect to *shortest-path* routing. 100 node network with random demand matrix