DIMACS Technical Report 2004–24
May 2004

Network Augmentation for Confluent Flow in Data Networks

by

Randeep Bhatia
Bell Labs
Murray Hill, NJ 07974

Nicole Immorlica[1]
MIT CSAIL
Cambridge, MA 02139

Tracy Kimbrel
IBM Watson Research Center
Yorktown Heights, NY 10598

Vahab S. Mirrokni
MIT CSAIL
Cambridge, MA 02139

Joseph (Seffi) Naor
Computer Science Department
Technion, Haifa 32000, Israel

Baruch Schieber[2]
IBM Watson Research Center
Yorktown Heights, NY 10598

[1]Affiliated Graduate Student Member
[2]Permanent Member

DIMACS is a collaborative project of Rutgers University, Princeton University, AT&T Labs–Research, Bell Labs, NEC Laboratories America and Telcordia Technologies, as well as affiliate members Avaya Labs, HP Labs, IBM Research and Microsoft Research. DIMACS was founded as an NSF Science and Technology Center.

ABSTRACT

Service providers rely on the management systems housed in their Network Operations Centers (NOCs) to remotely operate, monitor and provision their data networks. Due to the growing complexity and size of the data networks and the services provisioned on them, there has been a tremendous increase in management traffic. Traffic engineering for management traffic to avoid congestion resulting in packet losses and retransmissions is essential for the smooth functioning of these networks.

In this paper we study the traffic engineering of networks that use hop-by-hop routing (as is the case with most inter-domain and intra-domain routing protocols) for carrying management traffic. In such networks the set of routing paths connecting the NOC with the service provider's POPs (points of presence) form a tree rooted at the gateway router connected to the NOC. Given the management traffic need for each individual network node, we are interested in the minimal topological changes (link and/or link weight modifications) to ensure low-congestion routing of the management traffic. We show that the general versions of this problem are hard to solve. However, for some simpler cases in which new (physical or virtual) links provide a direct bypass for connecting network nodes to the NOC and the underlying network is a tree, we present efficient algorithms. We use these algorithms as the basis for designing efficient heuristics for alleviating congestion in general non-tree service provider network topologies.

1 Introduction

Service providers rely on their management systems housed in their Network Operations Centers (NOCs) to remotely operate, monitor, and provision their data networks. These management systems and the network used for carrying management traffic are critical resources without which customer services cannot be quickly provisioned, billing data cannot be collected, software upgrades and backups cannot be performed, and faults cannot be identified and fixed. The growing complexity and size of data networks, and the services provisioned on them, has resulted in a tremendous increase in management traffic. This traffic is either routed in-band with the data traffic on a common network, using the same router interfaces through which customer traffic flows, or it is routed out-of-band on a secure, separate network, dedicated to carrying management traffic, using router management interfaces which are dedicated exclusively to management traffic. Historically, traffic engineering and capacity planning have been done without regard to management traffic requirements. However, traffic engineering for management traffic, the goal of which is to avoid congestion which results in packet losses and retransmissions, is now becoming essential for the smooth functioning of service provider networks.

Generally speaking, a service provider's network consists of a number of management domains defined by a partition of the network topology. Each domain is managed by a Network Operations Center (NOC), which is connected to its managed domains via a management gateway router (MGR) within each domain. The MGR within a domain receives and forwards management traffic from/to the routers within the domain. Typically, the MGR does not originate or carry data packets (achieved, for example, by selecting large weights for its links), and it is only a source or destination for management traffic. Fig. 1 depicts these entities of a management domain. As already mentioned, management traffic may be routed in-band with data traffic, or separately out-of-band on a dedicated network. In either case, typically, hop-by-hop routing (as is the case with most inter-domain and intra-domain routing protocols, e.g. OSPF, ISIS, BGP, etc.) is used for routing management traffic. In other words, packets are routed based only on their destination address on a shortest path to the destination. (More precisely routers may load balance on equal cost shortest path as in OSPF. However, we make the simplifying assumption that packets are routed on a single path to the destination). This results in management packets being routed on a shortest path rooted tree (SPRT), rooted at the management gateway. Ideally, the management traffic load on a link on this SPRT must not exceed a certain percentage of the link's bandwidth. Beyond this increase in percentage, congestion is likely to occur, resulting in deteriorated performance of the management systems. With the increase in management traffic being routed in service providers networks, congestion is becoming more and more common, and efficient solutions are needed for alleviating this congestion.

The goals of traffic engineering are to enhance the performance of the network, while expending network resources economically. One way of alleviating management traffic congestion without requiring additional resources is to change link weights. There are many widely accepted reasons to avoid changing link weights arbitrarily. These include the en-

suing load on the control plane and on the routers for computing new shortest paths and the delay in before the network settles down to the new SPRT which causes, for example, out-of-sequence packet arrivals. Also, unless carefully constructed, the new SPRT may not be worth the network disruption. It is also known that, in general, computing a congestion-free SPRT, and hence "optimal" link weights, is a computationally very hard problem [4]. However, changing relatively few carefully chosen weights to alleviate congestion has been shown to be an effective approach to Traffic Engineering [8]. Another way of alleviating congestion is by changing network topology by adding new links. However, as with changing link weights, unless this is done carefully it can have negative ramifications on the performance of the network. In addition, many routers may be running at their full capacity and supporting nearly their maximum possible adjacency, thus precluding arbitrary topology changes.

In this paper we are interested in traffic engineering to alleviate congestion for management traffic that involves relatively few carefully selected link additions and link weight modifications in the network. We consider only link additions directly connecting the MGR (or the root of the SPRT) to other routers in the domain. The new additions may be for virtual links which are created as paths connecting the MGR with the routers on a lower layer network (MPLS, Optical), to avoid tying up precious router physical interfaces needed for carrying customer traffic. Once added, the links are assigned weights and affect the SPRT of the new network. We also allow changing weights on some of the links in the SPRT of the original network. There are several reasons for allowing only new links that extend directly to the MGR from the other routers:

- Minimal impact on data traffic: Often, management traffic is carried in-band over the same network as data traffic; thus adding new links may impact the topology for the data traffic as well. However, since the MGR does not typically originate or carry any data traffic, links added to the MGR do not result in any changes in the routing paths for data packets.

- Lower bandwidth: Since, by design, the links on the MGR do not carry any data traffic, and since the data requirements for management traffic are relatively small compared to those for data traffic, the new links need not have very high bandwidth (as supported by our simulations), and these links can even be provisioned by creating low bandwidth paths on a lower layer, advertised as virtual links on the routing plane.

- Node adjacency: In many cases the data routers run at their full capacity and support nearly their maximum possible adjacency, thus precluding any arbitrary increase in their routing adjacency. The MGR, however, does not have to process as much data or control traffic, and hence can support additional load and links.

- Efficiency: Intuitively, the traffic engineering problem is easier to solve given this restriction (as also supported by our theoretical results in Theorem 1), and it is the method of choice for many service providers.

The traffic engineering problem as defined above is very hard to solve in its full generality. As mentioned earlier, it is even hard to find whether there exists a congestion-free SPRT,

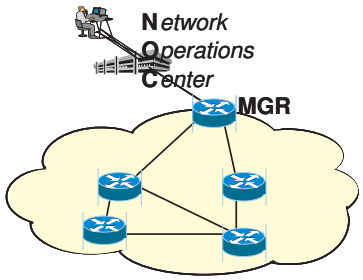

Figure 1: A NOC controlled management domain

for a given network, even if arbitrary link weight changes are allowed [4]. We show that this remains the case even if the underlying network is a tree (the SPRT), and new links (each with its own specified capacity) can be added between any pair of routers. However, when the underlying network is a tree, and new links can only be added between the MGR and the other routers, we design an efficient algorithm for the problem based on dynamic programming (DP). Our simulation of this algorithm on service provider network data, using the SPRT as the underlying network, shows that in most cases congestion can be eliminated by adding very few links at low bandwidth (hence removing very few links from the SPRT). Motivated by these results, we propose a heuristic for general service provider network topologies that uses the same set of new links (and modifies the weights of only those links that are removed) as computed by the algorithm for the SPRT of the underlying graph. Our simulation results show that the heuristic performs well even for general service provider network topologies.

Our dynamic programming (DP) algorithms for tree based networks are designed to support many natural constraints such as the bandwidth limit and cost of the potential new links. We note that link costs are used to model service provider priorities, monetary costs, etc. In addition, budget constraints can be used to trade off the number of new links added against the traffic engineering gains, etc. We are able to show that all these algorithms have very good worst case performance as well. For our simulations we also consider a natural greedy heuristic for tree based networks, and compare its experimental performance with our DP algorithms.

Tree-based networks arise naturally in other contexts, including Content Distribution Networks (CDN). For many applications ranging from distributing rich-media content to collecting billing data, CDNs often organize their deployment of servers in the form of a rooted tree (typically rooted at the Network Operations Center) with each node forwarding data from its children to its parent and vice versa [4]. Our techniques are equally applicable for alleviating congestion in these networks as well.

1.1 Related work

There exists a large body of work on the traffic engineering of networks for data traffic. We briefly touch on this work in this section. The work of [8, 6, 7] deals with intra-domain traffic engineering with applicability to interior gateway protocols such as OSPF and IS-IS etc. They show how routing can be improved by adjusting link weights based on a network-wide view of the data traffic and topology within a routing domain. A method to alleviate link load in IP backbone using deflection routing is proposed in [11]. The work of [1] addresses the importance of accurate knowledge of traffic demands for obtaining good network utilization, and they show that there exist a robust routing protocol for which nearly optimal utilization is possible on ISP networks even with a fairly limited knowledge of traffic demands. Practical algorithms for constructing this routing protocol are presented in [2, 10].

Confluent flows (another name for single source/destination hop-by-hop routing) and their effect on congestion have been considered in both the networking and theory communities. In [14] this problem is studied in the context of IP routing. They compare the traditional source invariant IP routing with routing that considers both source and destination (e.g. MPLS). They also design source invariant routing schemes with better performance. Also, in [15], this problem is considered for the purpose of traffic engineering for quality-of-service routing. Confluent flows are studied in a similar model [13] (but not with given source-destination demands) for the purpose of minimizing the total cost of installed capacity on the links of the network under the *hose* model.

Recently, [4] and [3] studied the relation between confluent flow and the well-studied general splittable flow and unsplittable flow problems [5, 12]. In particular, [4] and [3] present approximation algorithms for the minimum congestion confluent flow problem and the maximum throughput confluent flow problem. For the minimum congestion confluent flow, [3] shows a tight $\Theta(\log n)$-approximation for the minimum congestion confluent flow problem and a constant factor approximation for the maximum throughput confluent flow problem in general graphs. We prove that network augmentation for confluent flow in general graphs is not approximable within any factor (if P$\neq$NP), and we focus on augmenting a confluent flow on trees. Network augmentation has been considered in different settings, especially in the context of connectivity augmentation. See [9] for a survey. To the best of our knowledge, our work is the first attempt to find a set of minimum cost edges to augment a confluent flow with a guaranteed approximation factor.

1.2 Results

Our results can be broadly classified into algorithmic results in Sections 2, 3 and 4 and simulation results in Sections 5 and 6. In Sections 2 we present hardness results in terms of the in-approximability of the problem for general network topologies and for the tree topology when augmenting links are allowed between any pair of routers. We complement these results with fast approximation schemes (FPTAS) for the problem when the underlying network is a rooted tree and augmenting links can only connect to the root. Specifically in Section 3 we design a dynamic programming based FPTAS for the problem of minimizing

the total cost (e.g. number of links etc.) of the augmenting links needed to transform a given tree into a congestion free tree. In the case where the augmenting link have uniform cost (e.g. when minimizing number of links) our algorithm works in polynomial time. In Section 4 we allow for a budget constraint used to trade off the number of new links added against the traffic engineering gains, etc. For the budget-constrained problem we design a dynamic programming based FPTAS. In Section 5 we design a heuristic for our problem that is applicable to general network topologies. There we also describe a "greedy" heuristic which is a natural alternative to our dynamic programming algorithm for the more restricted tree topologies. In Section 6 we show that our heuristic for the general network topologies works very well on service provider networks.

2 Background and preliminaries

We model our network as a graph $G = (V, E)$. Each vertex $v \in V$ represents a router with s_v units of management data that must be routed between it and the management gateway. We say that v is a source of s_v units of flow. The management gateway is represented by a special *root* vertex $r \in V$. The edge set E represents links in the network. Each link, or edge e, has a hard capacity constraint c_e; that is, edge e cannot carry more than c_e units of traffic, in a congestion free routing. A flow function from several sources to a single sink is said to be *confluent* if all the flow reaching a vertex leaves on the same edge. The edges carrying non-zero flow in a confluent flow function induce a tree in the graph.

In networks that use hop-by-hop routing, edges have weights and for each vertex v, the s_v units of flow for vertex v is routed along the shortest path from v to r. These paths form a tree at vertex r, and the flows on the resulting tree is *confluent*. For the given set of weight assignments to the edges the routed flows may violate the edge capacities. Therefore, the traffic engineering goal is to install additional edges of minimum cost together with their weight assignment so that a shortest path routing can carry the flows from all the nodes to the root. As mentioned earlier typically it is desirable for these augmenting edges to extend from a node directly to the root. We note however that the unrestricted problem (where we can add edges between nodes as well as from nodes to the root), is in-approximable even for trees if P≠NP. Our reduction is from the NP-hard *confluent flow* problem, defined as follows.

Definition 1 *Given a graph $G = (V, E)$ with root $r \in V$, capacities c_e on the edges, and in which each vertex v is a source of s_v units of flow, decide whether all the sources can be routed to the root confluently (i.e., the edges with non-zero flow form a tree).*

Theorem 1 *There is no approximation algorithm for the unrestricted version of our problem even for trees unless P=NP.*

Proof. Suppose we are given an instance of the confluent flow problem on a (connected) graph $G = (V, E)$, and let $T = (V, E')$ be a spanning tree of G. Consider an instance of the

unrestricted version of our problem for tree T where the cost of adding any edge in $E \backslash E'$ is zero, and any other possible external edges have cost infinity. Then G has a confluent flow if and only if there is a set of augmenting edges A of cost zero and an assignment of weights to $E' \cup A$ such that the routing is feasible. Thus, an approximation algorithm for our problem would yield a solution to the confluent flow problem instance. □

In the rest of the paper we will restrict ourselves to the case where only edges going from the nodes directly to the root can be added. This allows us to define the *network augmentation problem.*

Definition 2 *We are given a graph $G = (V, E)$ with root $r \in V$, capacities c_e together with weights $w_e > 0$ on the edges, and in which each vertex v is a source of s_v units of flow. Furthermore, for each node v, for price/cost p_v, we can route c_v units of flow directly from v to the root r. The network augmentation problem is to find a set of edges A of minimum total cost from nodes directly to the root, and a weight assignment for edges $E \cup A$, such that the routing on the shortest paths tree is feasible.*

A proof similar to that of Theorem 1 shows that the problem is in-approximable in general graphs. Therefore, we further restrict ourselves and study our problem on trees, giving us a heuristic for the problem in general graphs by applying our algorithms to the shortest paths tree of the original network.

When we restrict our algorithms to trees, we have the advantage that we may not need to change the weights of the edges belonging to the original network. Consider the following procedure for choosing weights for the edges in the augmentation set. We can proceed sequentially as follows. At each step we add an edge from a node to the root and delete an edge from the original tree. Suppose that edge (v, r) is added and edge (x, y) is deleted, where the weight of the (unique) path from v to x is W_1 and the weight of the (unique) path from y to r is W_2. Then, the weight of edge (v, r) should satisfy:

$$\text{For vertex } x: \quad W_1 + w(v, r) < w(x, y) + W_2,$$

$$\text{For vertex } y: \quad W_2 < w(x, y) + W_1 + w(v, r).$$

That is,

$$W_2 - W_1 - w(x, y) < w(v, r) < W_2 - W_1 + w(x, y),$$

and there is a feasible choice for $w(v, r)$. Note that if all the edge weights computed by this procedure are non-negative then with this choice of weight setting for the augmenting edges we do not need to change any edge weights in the original network. Even when some weights turn out to be negative we can use the standard edge weight modification procedure to make all weights non-negative, since it can be shown that all cycles in the graph remain non-negative by our choice for $w(v, r)$. However this would also require changing some edge weights in the original graph. For the above reasons, from now on until Section 5, we ignore the assignment of edge weights and thus our problem becomes equivalent to the following confluent flow problem.

Definition 3 *We are given a graph $G = (V, E)$ with root $r \in V$, capacities c_e on the edges, and in which each vertex v is a source of s_v units of flow. Furthermore, for each node v, for price p_v, we can route c_v units of flow directly from v to the root r. The network augmentation problem is to find a set of edges of minimum cost from nodes (directly) to the root, and a confluent flow satisfying all the sources s_v.*

The network augmentation problem is at least as hard as the weakly NP-hard knapsack problem. In the knapsack problem, we are given a finite capacity knapsack, and a set of items, where each item has a weight and a value. The goal is to find a maximum value subset of the items, such that its weight does not exceed the capacity of the knapsack.

Theorem 2 *The network augmentation problem is weakly NP-hard.*

Proof. Given an instance of the knapsack problem, let C denote the capacity of the knapsack, and suppose we are given n items, where each item v has weight s_v and value p_v. Construct a tree T with n leaves, where leaf v is a source of s_v units of flow. These leaves all stem from a single vertex u which is a source of 0 units of flow. The edge from a leaf v to vertex u has capacity $c_{(v,u)} = s_v$. Vertex u is adjacent to the root r through an edge of capacity C. We can buy an edge from any leaf v to the root at cost p_v and this edge has capacity $c_v = s_v$. We prevent the solution from buying an edge from u to the root by assigning this edge infinite cost and zero capacity. It is not hard to see that the optimal solution of this instance of the network augmentation problem yields an optimal solution to the knapsack instance. □

Thus, we consider approximation algorithms for our augmentation problem. The best approximation we can hope for is a *fully polynomial-time approximation scheme* (FPTAS). An approximation algorithm is an FPTAS for a minimization (maximization) problem if it finds a $(1+\epsilon)$-approximation ($(1-\epsilon)$-approximation) solution in time which is polynomial in the input size and $1/\epsilon$. We present an FPTAS for several versions of our problem, and optimal algorithms for some restricted instances.

2.1 Terminology

In the rest of this paper, we consider the network augmentation problem on trees only. Thus, we are given an initial tree T which we must augment and change into a new tree T' on which flow is routed. For ease of discourse, we consider vertices and edges to be oriented with respect to the original tree T. We say that the root of the input tree T is at the *top*, and that a vertex v (edge e) is *below* vertex v' (edge e') if it is farther away from the root than v' (e'). For an edge (x, y) in the tree we assume that vertex x is below vertex y. We will write $V(G)$ to indicate the vertex set of graph G and $E(G)$ to indicate the edge set of graph G.

3 Routing flow at minimum cost

In this section we present a dynamic-programming-based FPTAS for the network augmentation problem on trees. We describe the dynamic programming for the case of a binary tree only. The general case can be reduced to this case with only a constant factor increase in the size of the table maintained in the dynamic programming and with only a constant factor increase in the running time. For simplicity we omit the details of this construction. (We note that the general case cannot be handled by simply adding "dummy" nodes to make the tree binary, since a confluent flow in the "binarized" tree may not necessarily translate to a confluent flow in the original tree.)

For each edge $e = (v, u) \in E$ (with, say, u above v) and each possible cost ρ, we find the smallest possible flow $f(e, \rho)$ on edge e in while spending at most ρ in the subtree T_v rooted at v. That is, we want to satisfy the maximum possible flow from sources in T_v, including v, and route as little flow as possible through e, or even reverse the flow on e and carry flow from sources outside T_v through e into T_v and eventually to the root through purchased edges.

Note that in our setting, a flow f is a function from a subset of edges E' (exactly those edges on which flow is routed) to the real numbers which describes how flow is routed on edges E'. A positive flow of $f(e, \rho)$ on edge e indicates that there are $f(e, \rho)$ units of flow from sources in T_v unsatisfied by flow f. We need to route these $f(e, \rho)$ units to the root through u, so edge e will carry flow $f(e, \rho)$ toward the root. A negative flow of magnitude $|f(e, \rho)|$ on e indicates that all the flow from sources in T_v is satisfied by purchased edges to the root from vertices below e, and we can push additional flow through u into T_v while maintaining feasibility of the solution. Edge e will carry up to $|f(e, \rho)|$ units of flow in the downward direction, from u to v.

To formalize this, we first define the "flow limiting" function $L(c, f)$ for a positive capacity c and flow f as follows. If $f > c$, $L(c, f) = \infty$. Otherwise, $L(c, f) = \max(-c, f)$. We wish to solve the recurrence

$$f((v, u), \rho) = \min\{L(c_{(v,u)}, s_v - Z_v \cdot c_v + \bar{f})\}$$

$$\bar{f} = Z_{(x,v)} \cdot f((x, v), \rho_x) + Z_{(y,v)} \cdot f((y, v), \rho_y)$$

where the minimum is over $\rho_x, \rho_y, Z_{(x,v)}, Z_{(y,v)}, Z_v \in \{0, 1\}$ and $f((v, u), \rho)$ denotes the minimum flow from v to its parent u given that cost ρ is paid to buy edges to the root from vertex v and vertices in the subtrees rooted at the children x and y of v, Z_v denotes a binary decision variable indicating whether the edge from v to r is purchased, and $Z_{(x,v)}$ and $Z_{(y,v)}$ denote binary decision variables indicating whether flow may be carried on edges (x, v) and (y, v), respectively, subject to the following constraints:

- $\rho \geq \rho_x + \rho_y + Z_v \cdot p_v$;
- if $f((x, v), \rho_x) > 0$ (resp. $f((y, v), \rho_y) > 0$) then $Z_{(x,v)} = 1$ (resp. $Z_{(y,v)} = 1$);

- at most one of $f((v,u),\rho)$, $-Z_{(x,v)} \cdot f((x,v),\rho_x)$, $-Z_{(y,v)} \cdot f((y,v),\rho_y)$, and Z_v is positive.

The first two constraints ensure that the flow into v (in whichever direction) can be routed as necessary at a total cost of ρ or less, and the third ensures that the resulting solution forms a tree. The flow limiting function L ensures that the flow on (v,u) (in either direction) does not violate the capacity $c_{(v,u)}$. In the case of a violation of an edge's capacity in the original direction toward the root, there is no solution of cost ρ. The flow value of infinity will propagate to an edge incident on the root r. If $f((v,u),\rho) = \infty$ for a child v of r and all values of ρ, there is no feasible solution at any cost.

Note that as a degenerate case, it is possible if $s_v = 0$ (and only in this case) that all of the flows out of v are non-positive (a negative would indicate capacity that is available but unused) and v is an isolated vertex in the final solution.

First, we show how to solve the problem optimally when the costs p_v are polynomially bounded and have polynomially many distinct values (i.e., $p_v \in \{0, \ldots, n^c\}$ for some constant c) and the tree is binary. Note the maximum total cost is n^{c+1} We build a dynamic programming table that indicates the amount of flow that must be sent on edge (v,u) given that ρ is spent in the subtree rooted at v, for each cost $\rho \leq n^{c+1}$ for all $e \in E$.

Algorithm 1 *For each edge $e = (v,u) \in T$ with u above v, let T_v be the subtree rooted at v. We compute the minimum flow from v to u, or the maximum amount of flow $f(e,\rho)$ that we can feasibly push* into *T_v, for cost ρ. Note that if we can push flow into T_v (and eventually to the root through edges in T_v and some purchased edge), then the flow on e will be negative. If we must carry some flow from sources in T_v through e in the original direction from v to u, then the flow on e will be positive. Thus, we want to minimize $f(e,\rho)$ subject to the cost and feasibility constraints.*

We begin with the leaves and compute the possible flows for each edge for each cost in bottom-up fashion. For an edge e incident to a leaf v, if $c_e \geq s_v$, then edge e can carry flow s_v for each $\rho < p_v$. If $c_v \geq s_v$, then edge e can carry flow $-\min(c_e, (c_v - s_v))$ for each $\rho \geq p_v$.

Now we would like to compute the set of possible flows for each cost on an edge (v,u) given the costs and corresponding flows on the edges (x,v) and (y,v) immediately below (v,u) in T. Let ρ_x and ρ_y be a pair of costs from the tables for (x,v) and (y,v) respectively. Let $f((x,v),\rho_x)$ and $f((y,v),\rho_y)$ be the corresponding flows on (x,v) and (y,v).

There are three cases to consider:

- *If $f((x,v),\rho_x) > 0$ and $f((y,v),\rho_y) > 0$, then, if we can do so without violating capacities,*
  - *we can route all leftover flow below e up through e at no additional cost: $\rho = \rho_x + \rho_y$, $f(e,\rho) = f((x,v),\rho_x) + f((y,v),\rho_y) + s_v$.*
  - *we can route all leftover flow below e and possibly some flow from sources above e on edge (v,r) at additional cost p_v: $\rho = \rho_x + \rho_y + p_v$, $f(e,\rho) = -\min(c_e, c_v - (f((x,v),\rho_x) + f((y,v),\rho_y) + s_v))$.*

- *If $f((x,v),\rho_x) > 0$ and $f((y,v),\rho_y) \leq 0$ (the case $f((x,v),\rho_x) \leq 0$ and $f((y,v),\rho_y) > 0$ is analogous), then, if we can do so without violating capacities,*
  - *we can route all leftover flow below e and possibly some flow from sources above e on edge (y,v) at no additional cost: $\rho = \rho_x + \rho_y$, $f(e,\rho) = -\min(c_e, |f((y,v),\rho_y)| - (f((x,v),\rho_x) + s_v))$.*
  - *we can route all leftover flow below (x,v), s_v, and possibly some flow from sources above e on edge (v,r) at additional cost p_v (in this case, edge (y,v) will have zero flow): $\rho = \rho_x + \rho_y + p_v$, $f(e,\rho) = -\min(c_e, c_v - (f((x,v),\rho_x) + s_v))$.*
  - *we can route all leftover flow below (x,v) and s_v up edge e at no additional cost (in this case, edge (y,v) will have zero flow): $\rho = \rho_x + \rho_y$, $f(e,\rho) = f((x,v),\rho_x) + s_v$.*
- *If $f((x,v),\rho_x) \leq 0$ and $f((y,v),\rho_y) \leq 0$, then, if we can do so without violating capacities,*
  - *we can route flow s_v and possibly some flow from sources above e on edge (x,v) at no additional cost (in this case, edge (y,v) will have zero flow): $\rho = \rho_x + \rho_y$, $f(e,\rho) = -\min(c_e, |f((x,v),\rho_x)| - s_v)$.*
  - *we can route flow s_v and possibly some flow from sources above e on edge (y,v) at no additional cost (in this case, edge (x,v) will have zero flow): $\rho = \rho_x + \rho_y$, $f(e,\rho) = -\min(c_e, |f((y,v),\rho_y)| - s_v)$.*
  - *we can route flow s_v on edge (v,r) at additional cost p_v (in this case, edges (x,v) and (y,v) will have zero flow): $\rho = \rho_x + \rho_y + p_v$ and $f(e,\rho) = -\min(c_e, c_v - s_v)$.*
  - *we can route flow s_v on edge e at no additional cost (in this case, edges (x,v) and (y,v) will have zero flow): $\rho = \rho_x + \rho_y$, $f(e,\rho) = s_v$.*

After performing all such computations, for each resulting cost ρ, we add an entry to our table F_e. The value of the entry is the flow f achieved at cost ρ for which $f(e,\rho)$ is minimized.

For the final step, suppose x and y are children of the root r. The algorithm reports the value $\rho_x + \rho_y$ where ρ_x and ρ_y are the minimum costs for which $|f((x,r),\rho_x)| \leq c_{(x,r)}$ and $|f((y,r),\rho_y)| \leq c_{(y,r)}$, respectively.

The standard mechanism of recording pointers between entries in the dynamic programming tables and backtracking through them yields an algorithm to find an optimal flow as well as its cost.

Theorem 3 *The above procedure finds an optimal solution in polynomial time if all edge costs are integers bounded by a polynomial.*

Proof. For the running time, suppose for all v, $p_v \in \{0, \ldots, n^c\}$. Then there are only n^{c+1} distinct values for the cost of flows on an edge and all edges below it. Thus we must find for each vertex a table of at most n^{c+1} values. Each of these is found by considering at most n^{c+1} combinations of flows on the edges below it as specified by the recurrence relation.

By construction, every flow considered for each edge is constructed from feasible flows for the subtrees below it in such a way that the resulting flow is confluent and doesn't violate capacities. Furthermore, it routes all flow from sources in the subtree below the edge either to the root through purchased edges below it, up through the edge in question, or by purchasing a new edge, to eventually reach the root. Therefore, every flow considered in the final step of the algorithm is a feasible flow.

It remains to show that the flow computed is optimal. Fix an optimal solution. Let ρ_e^* be the cost paid by the optimal solution in the subtree below e, and let f_e^* be the flow on e. (Recall that f_e^* may be negative.) We claim that the value of $f(e, \rho_e^*)$ computed for the flow on edge e for cost ρ_e^* is at most f_e^*. We prove the claim by induction.

For an edge $e = (v, u)$ immediately above a leaf v, suppose the optimal solution does not buy the edge (v, r) from v to the root. The algorithm computes $f(e, 0) = s_v$ as needed. Suppose the optimal solution buys (v, r) incurring cost p_v. Then it routes at most c_v through this edge and the flow on e is at least $\max(s_v - c_v, -c_e)$. (Again, this may be negative.) This is the value $f(e, p_v)$ specified in the recurrence and computed by the algorithm.

Consider an edge $e = (v, u)$ with edges $e_x = (x, v)$ and $e_y = (y, v)$ immediately below e. Let T_v, T_x, and T_y be the subtrees rooted at v, x, and y respectively, let ρ_v^*, etc., denote the costs paid by the optimal solution in these subtrees, and let $f_{(v,u)}^*$, etc., denote the flows in the optimal solution. Let Z_v^*, $Z_{(x,v)}^*$, and $Z_{(y,v)}^*$, respectively, indicate whether the optimal solution places flow on edges (v, u), (x, v), and (y, v). By the induction hypothesis, $f((x, v), \rho_x^*) \leq f_{(x,v)}^*$ and $f((y, v), \rho_y^*) \leq f_{(y,v)}^*$. Thus the value for the flow on e at cost ρ^* computed using these settings of the decision variables Z_v, $Z_{(x,v)}$, and $Z_{(y,v)}$ is at most f_e^*, and the minimum $f((v, u), \rho_v^*)$ specified in the recurrence and computed by the algorithm is at most this value as well.

□

Consider a uniform instance, in which all costs are the same (and thus the objective is to minimize the number of augmentations). Note that the above procedures finds an optimal solution in polynomial time for such an instance.

Consider an arbitrary instance. We can round the prices of that instance and use the dynamic program to find an approximately optimal solution. The rounding works by "guessing" the maximum price of an edge bought in an optimal solution, which is a lower bound on the overall cost of an optimal solution.

Algorithm 2 Input:

- *An instance of the network augmentation problem.*
- *Positive* $0 < \epsilon \leq 1$ *(We will find a* $1 - \epsilon$*-approximation for the problem).*

1. *If the given instance is feasible as is, terminate.*
2. *For each* $P \in \{p_v\}$*, let* $S_P = \{v | p_v \leq P\}$.
   (a) *Let* $K_P = \frac{\epsilon P}{n}$.

(b) *For each* $v \in S_P$, *let* $p_v^P = \lceil \frac{p_v}{K_P} \rceil$.

(c) *For each* $v \notin S_P$, *let* $p_v^P = \infty$ *(i.e., set the decision variable* $Z_v = 0$ *in the dynamic program).*

(d) *Solve the instance optimally for prices* p_v^P*; denote this solution by* O_P.

3. *Output the solution* O_P *with minimum cost according to the original (unrounded) prices.*

Theorem 4 *Algorithm 2 is an FPTAS for the network augmentation problem with arbitrary costs.*

Proof. First notice that the algorithm runs in time polynomial in n and $1/\epsilon$ as we solve n dynamic programs and the prices in each dynamic program are bounded by $\lceil n/\epsilon \rceil$.

Let $C(S)$ be the cost of solution S in the original instance and $C_P(S)$ be the cost in the rounded instance with rounding/cutoff parameter P. For any price p_u feasible given the cutoff, $K_P p_u^P - K_P \leq p_u \leq K_P p_u^P$. Therefore,

$$K_P C_P(S) - nK_P \leq C(S) \leq K_P C_P(S)$$

provided S does not use edges of cost more than P.

Let O_P be the solution output by our algorithm, P^* be the maximum price of an edge in an optimal solution, O_{P^*} be the solution considered by our algorithm for $P = P^*$ (optimal for the rounded instance with rounding/cutoff parameter P^*), and O be the optimal solution to the original instance. Then,

$$\begin{aligned} C(O_P) &\leq C(O_{P^*}) \\ &\leq K_{P^*} C_{P^*}(O_{P^*}) \\ &\leq K_{P^*} C_{P^*}(O) \\ &\leq C(O) + nK_{P^*} \\ &= C(O) + \epsilon P^* \\ &\leq (1+\epsilon) C(O). \end{aligned}$$

□

4 Maximizing throughput with budget constraint

Next we consider the budget-constrained network augmentation problem. In this problem, we can add edges of total cost no more than a given budget B, and we seek to maximize the amount of flow routed subject to this budget constraint. Again, we require that flow is routed confluently. However, we permit solutions to route only part of the flow sourced at a node.

In the following we reduce the knapsack problem to the budget-constrained network augmentation problem.

Theorem 5 *The budget-constrained network augmentation problem is weakly NP-hard.*

Proof. Given an instance of the knapsack problem with capacity C and n items of weights p_v and values s_v, construct a tree T with zero-capacity edges and n leaves, where leaf v is a source of s_v units of flow. For each vertex v, we can add an edge of cost p_v with capacity s_v at v. The budget constraint B in the network augmentation problem is equal to the knapsack size C. It is easy to see that an optimal solution of the knapsack instance can be obtained from an optimal solution of the budget-constrained network augmentation instance. □

Again, the best we can hope for is to find a fully polynomial approximation scheme.

4.1 An FPTAS for maximizing throughput

In this section, we describe a natural dynamic program to solve the budget-constrained network augmentation problem when the source values and capacities are polynomially bounded. We will use this dynamic program to design an FPTAS for the case in which we have a lower bound on the capacity of each edge in terms of the size of the maximum source. Then we generalize this FPTAS to an FPTAS for the general budget-constrained network augmentation problem. As before we describe the dynamic programming for the case of a binary tree only and note that the general case can be reduced to this case as well.

Assuming sources and capacities are polynomially bounded, a variation on algorithm 1 solves the problem optimally in polynomial time. As before, our algorithm constructs a table for each edge. However, the table is now indexed by the form of the solution – the flows on the edges and flows satisfied by added edges – and the entries of the table are the costs. More specifically, for an edge e, for each possible flow on e, and for each amount of flow already satisfied by added edges below e, we store the cost of such a solution.

We begin with the leaves and compute the costs in bottom-up fashion. For an edge e incident to a leaf v, edge e can carry any flow $f_e \leq \min(s_v, c_e)$ for cost $\rho = 0$ without satisfying any flow sourced at v through (v, r). If $c_v \geq s_v$, then edge e can carry any flow $0 \geq f_e \geq -\min(c_e, (c_v - s_v))$ for cost $\rho = p_v$ while satisfying $\min(s_v, c_v)$ flow sourced at v. Of course, we only need to remember the best solution for a given cost entry, and so the table for a leaf will have up to two entries: flow $\min(s_v, c_e)$ and satisfied flow 0 at cost 0; flow $-\min(c_e, \max(c_v - s_v, 0))$ and satisfied flow $\min(s_v, c_v)$ at cost p_v.

The inductive step for an edge e that is not incident to a leaf is the same as before, only now we must also record the amount of flow satisfied below e as well as the flow through e and the cost. We simply add the amounts of flow satisfied below each of the children's edges. As in the case of an edge incident to a leaf above, we must cap the flows through the given edge and the purchased edge to the root, if used, by their capacities.

For the final step, we combine the tables of the edges incident to the root as described in the inductive step. We discard all solutions whose costs are greater than the budget B, and return the remaining solution with the maximum amount of satisfied flow.

It is not hard to see that this algorithm computes the optimal solution in polynomial time when the capacities and sources are polynomially bounded. We can use this algorithm to get an FPTAS for an instance with arbitrary capacities and sources.

First, we present an FPTAS for the case in which capacities are not too small compared to sources; that is, the ratio between the minimum capacity and maximum satisfiable flow from a single source is bounded from below. Let us introduce some definitions and notation first.

Definition 4 *Let D be the largest amount of flow from a single node which can be routed to the root by spending at most B (D is a lower bound on OPT). Note D can be just a fraction of the flow sourced at the node.*

By the above definition of D, it is straightforward to see that $D \leq \text{OPT} \leq nD$ where OPT is the value of the optimal solution.

Definition 5 *Let $c_{\min}$ be the smallest capacity c_e of an edge e or c_v of an added edge c_v, i.e.,*

$$c_{\min} = \min(\min_{e \in E} c_e, \min_{v \in V} c_v).$$

Let R be the minimum ratio between any capacity and D, i.e., $R = \frac{c_{min}}{D}$. Let p be the maximum of 1 and $\log_n R$, i.e., $p = \max(1, \log_n R)$. Note that $\frac{c_e}{D} \geq \frac{1}{n^p}$ for all edges.

The Algorithm **BCNA1** for the budget-constrained network augmentation problem is as follows:

Algorithm 3 Input:

- *Instance I of the budget-constrained network augmentation problem.*
- *Positive $0 < \epsilon \leq 1$ (We will find an $1 - \epsilon$-approximation for the problem).*

1. *Given ϵ, let $\epsilon' = \epsilon'' = \frac{\epsilon}{2}$.*
2. *Cap all sources at D and capacities at nD (this does not change OPT, so from now on we will assume our instance originally satisfied this condition and thus $D = d_{max}$).*
3. *Let $K = \frac{\epsilon' D}{n^p}$.*
4. *For each source s_v, let $s'_v = \lceil \frac{s_v}{K} \rceil$.*
5. *For each capacity c_e, let $c'_e = \lceil \frac{c_e}{K} \rceil$.*
6. *Solve the instance optimally for sources s'_v and capacities c'_e to get solution O'.*
7. *Scale down O' by a factor of $(1 - \epsilon'')$ and output the corresponding ("unrounded") solution S. That is, if O' routes $\alpha s'_v$ fraction of the flow sourced at node v along edge e, then O routes $\alpha(1 - \epsilon'')s_v$ flow sourced at node v along edge e.*

Theorem 6 *The Algorithm* **BCNA1** *is a* $1-\epsilon$*-approximation algorithm for the budget-constrained network augmentation problem with the running time polynomial in* n^p *and* $\frac{1}{\epsilon}$*. In particular, it is an FPTAS if* $\frac{D}{c_{\min}}$ *is bounded above by any polynomial in* n*.*

Proof. It is easy to see that capping sources at D and capacities at nD will not change the value of the optimal solution. Now, we need to bound the number of different flows on edges in the rounded instance. In fact after rounding, all sources are integers between 1 and $\lceil \frac{2n^p}{\epsilon} \rceil$ and capacities are integers between 1 and $\lceil \frac{2n^{p+1}}{\epsilon} \rceil$. Thus each flow on an edge is the sum of sources and capacities and is an integer between 1 and $\lceil \frac{2n^{p+2}}{\epsilon} \rceil$, which is polynomial in n^p and $\frac{1}{\epsilon}$. Thus, the running time of the dynamic program is polynomial if n^p is a polynomial, or equivalently if $\frac{D}{c_{\min}}$ bounded above by a polynomial in n.

Let $F(Q)$ be the amount of flow satisfied by any solution Q in the original instance and $F'(Q)$ be the amount of flow satisfied by solution Q in the rounded instance (a solution Q specifies the fraction of a source to route on any original or added edge). For any source s_u, $Ks'_u - K \leq s_u \leq Ks'_u$. Therefore,

$$KF'(Q) - nK \leq F(Q) \leq KF'(S).$$

Let S be the solution output by our algorithm, O' be the optimal solution of the rounded instance, and O be the optimal solution of the original instance. Now, consider the budget-constrained network augmentation instance with sources Ks'_v as sources and Kc'_e and Kc'_v as capacities. The value of the optimal solution for this instance is $KF'(O')$. The sources and capacities in this instance are not smaller than the corresponding sources and capacities in the original instance (with sources and capacities s_v and c_e), thus the value of the optimal solution of this instance should not be smaller than that of the original instance. Therefore,

$$KF'(O') \geq F(O).$$

Thus,

$$\begin{aligned} F(S) &= (1-\epsilon'')F(O') \\ &\geq (1-\epsilon'')(KF'(O') - nK) \\ &\geq (1-\epsilon'')(F(O) - nK) \\ &= (1-\epsilon'')(F(O) - \frac{\epsilon' D}{n^{p-1}}) \\ &\geq (1-\epsilon'')(F(O) - \epsilon' D) \\ &\geq (1-\epsilon'')(1-\epsilon')F(O) \\ &\geq (1-\epsilon)F(O) \end{aligned} \tag{1}$$

It remains to prove that S is feasible. Let f be the flow on edge e in S and f' be the flow on edge e in O'. We know that $f'_e \leq c'_e$ and we need to prove that $f_e \leq c_e$. First we prove that $(1-\epsilon'')c'_e \leq c'_e - 1$.

$$\begin{aligned} \frac{c_e}{D} \geq \frac{1}{n^p} &\Rightarrow \\ \epsilon'' = \epsilon' \geq \frac{\epsilon' D}{n^p c_e} &\Rightarrow \\ \epsilon'' \geq \frac{1}{c'_e} &\Rightarrow \\ (1-\epsilon'')c'_e \leq c'_e - 1 & \qquad (2) \end{aligned}$$

Now,

$$\begin{aligned} f(e) &\leq K(1-\epsilon'')f'_e \\ &\leq K(1-\epsilon'')c'_e \\ &\leq K(c'_e - 1) \\ &= Kc'_e - K \\ &\leq c_e \qquad (3) \end{aligned}$$

Thus the flow f in solution S is feasible.

□

The flaw of Algorithm **BCNA1** is that its running time depends on the ratio $\frac{D}{c_{\min}}$ for an arbitrary instance. In the following, we overcome this problem by removing a set of edges with small capacities and proving that removing this set of edges does not change the value of the optimal solution. Then, using the Algorithm **BCNA1** in the new instance, we design an FPTAS for the general budget constrained network augmentation problem.

The Algorithm **BCNA2** is as follows:

Algorithm 4 Input:

- *Instance I of the budget constrained network augmentation problem.*
- *Positive $0 < \epsilon \leq 1$ (We will find a $1-\epsilon$-approximation for the problem).*

1. $\epsilon_1 = \epsilon_2 = \frac{\epsilon}{2}$
2. *Consider the set E_1 of all edges of the capacity less than $\frac{D}{n^{q+1}}$ where $q = \log_n \frac{1}{\epsilon_1}$. Remove all edges in E_1 from the graph. Let the resulting instance after removing these edges be I'.*
3. *Call Algorithm* **BCNA** *on instance I' and parameter ϵ_2. Let the output be O'.*
4. *Output O'.*

Theorem 7 *The Algorithm* **BCNA2** *4 is an FPTAS for the budget-constrained network augmentation problem. More precisely, this algorithm is a* $(1-\epsilon)$*-approximation for the budget-constrained network augmentation problem with the running time polynomial in* n *and* $\frac{1}{\epsilon}$.

Proof. In the instance I', $\frac{c_{\min}}{D} \geq \frac{1}{n^{q+1}}$, thus from Theorem 6, the running of Algorithm **BCNA1** is a polynomial in $n^{\max(1,q+1)}$ and $\frac{1}{\epsilon_2}$, thus it is a polynomial in $n^{1+\log_n \frac{1}{\epsilon_1}} = \frac{n}{\epsilon_1} = \frac{2n}{\epsilon}$ and $\frac{1}{\epsilon}$.

Let OPT$'$ be the value of the optimal solution of instance I'. From Theorem 6, we know that $F(O') \geq (1-\epsilon_2)\text{OPT}'$. Let OPT be the value of the optimal solution of instance I. Since each removed edge can carry at most $\frac{D}{n^{q+1}}$ flow and there are at most n separate edges (or paths) to the root, $\text{OPT} \leq \text{OPT}' + \frac{D}{n^q}$. Therefore,

$$\text{OPT}' \geq \text{OPT} - \frac{D}{n^q} \geq \text{OPT}(1 - \frac{1}{n^q})$$

By the definition of q, $n^q = \frac{1}{\epsilon_1}$, thus, $\frac{1}{n^q} = \epsilon_1$, thus $\text{OPT}' \geq \text{OPT}(1-\epsilon_1)$. From this inequality, we have

$$\begin{aligned} F(O') &\geq (1-\epsilon_2)\text{OPT}' \\ &\geq (1-\epsilon_2)(1-\epsilon_1)\text{OPT} \\ &\geq (1-\epsilon)\text{OPT} \end{aligned} \tag{4}$$

as desired. This completes the proof of the FPTAS.

□

5 Heuristics for general network topologies

In this section we present our heuristic for alleviating congestion in more complex non-tree service provider network topologies. Here we assume that there are no budget constraints and all flows need to be routed. We start out by describing a "greedy" heuristic which is a natural alternative to the dynamic programming (DP) algorithm for the more restricted tree topologies. We use this heuristic as a benchmark for comparing the DP algorithm both when used directly on tree topologies and when used to design heuristics for the more general network topologies.

5.1 Greedy algorithm for tree topologies

Informally, the greedy algorithm looks for those nodes "underneath" a congested edge, whose augmenting edge can bypass some of the traffic on the congested edge directly off to the root. The greedy algorithm repeatedly picks that congested edge and a node underneath it which

can be used to bypass the most traffic off the congested edge. The algorithm makes this choice with no regard to the cost or the capacity of the augmenting edges used for the bypass. The algorithm is repeated until there are no more congested edges or there are no "bypass" nodes under congested edges. Note that the greedy algorithm can be efficiently implemented and it is a natural alternative for alleviating congestion in tree topologies.

More formally, the algorithm works as follows. We say an edge is *tight* if the total flow of the nodes that route traffic up to the root through this edge exceeds the edge capacity. An edge e is *maximally tight* if it is the lowest tight edge in the given tree (no other tight edge in the subtree under this edge). From this definition, it is clear that if there are no maximally tight edges, then the given network is congestion free. We say a vertex is a "good" vertex if the total flow of the nodes that route traffic up to the root through this node is at most the capacity of the augmenting edge that connects this node to the root. For any maximally tight edge e, we call vertex v a *maximal good vertex for edge* e if v is the uppermost good vertex below e.

Given tree T with root r, the greedy algorithm finds a maximally tight edge e. It then finds the maximal vertex v for e such that the nodes in the subtree T_v have the maximum total traffic flow. The algorithm adds edge (v, r) and removes the edge just above v in the original tree, and recurses on the remaining tree until there are no more maximally tight edges, or until there are no more good vertices for some maximally tight edge (in this case no solution is found).

Algorithm 5 *1. If there is no maximally tight edge, or a tight edge has no maximal good vertices under it, then terminate successfully, or unsuccessfully, respectively.*

2. Otherwise, let e be a maximally tight edge. Find all maximal good vertices for edge e, say $S = \{v_i\}$.

3. Find the vertex $v_i \in S$ with the maximum value of $\sum_{w \in V(T_{v_i})} s_w$.

4. Add edge (v_i, r) and remove the edge just above v_i to form tree T'.

5. Repeat the algorithm for tree T'.

5.2 Heuristic for general network topologies

Recall that here we do not assume any budget constraints and that all flows need to be routed. Our heuristic for general network topologies works as follows. Let the given network be denoted by G.

Algorithm 6 *1. Compute the shortest path rooted tree (SPRT) T (based on the provisioned link weights) rooted at the management gateway router (MGR).*

2. Use a tree algorithm (DP or Greedy) to modify the SPRT, using new links of minimum total cost, to obtain a congestion free rooted tree T'.

3. *Augment G with the links in $T' - T$.*

4. *Set the weights of these links appropriately (equal to the weights of other links on the MGR).*

5. *Set the weights of the links in $T - T'$ to a large value so that they are no longer in the new SPRT.*

6 Simulations

In this section we present our simulation results on service provider backbone network topologies and management traffic requirements. Our main goal is to study the performance of the tree based algorithms when used as a heuristic for alleviating congestion of management traffic in general network topologies (as presented in Section 5.2). In addition we are interested in comparing the performance of the dynamic programming (DP) algorithm with the natural "Greedy" heuristic (Section 5.1) for tree topologies. Here we assume that there are no budget constraints and all flows need to be routed.

The metrics that we use to measure the performance of the heuristics is the improvement in the congestion (defined as the maximum ratio of the link load to the link bandwidth), the extent of the modifications (total number or cost of updated links) to the network and the amount of extra bandwidth per link needed for alleviating congestion. We are also interested in measuring the extent of the modifications that are needed to the SPRT to turn it into a congestion free rooted tree for routing the management traffic and its dependence on the bandwidth of the additional links.

The data for our simulations comprises of five independent domains in a service providers backbone network. We call these domains A, B, C, D, E. Each domain consists of one management gateway which forms the root of the SPRT for that domain on which all the management traffic of the domain is routed in-band along with the data traffic within that domain. A link is considered congested for management traffic if the amount of management traffic on the link is more than 5% of its bandwidth. The links within the domains have bandwidths ranging from $DS1$ to $OC12$. There are between 50 to 100 routers in each domain and each router's management requirement varies from $50Kbps$ to $2000Kbps$. The congestion in the five domains, measured as the maximum ratio of the load of a link to the bandwidth of the link in the rooted SPRT, is $96, 5, 218, 414, 91$ respectively.

Our first set of results is shown in Fig. 2, Fig. 3, Fig. 4. Here a congestion free routing tree is computed by adding and deleting links from the SPRT, where all the newly added links must have the same bandwidth and the goal is to minimize the number of new links added (or deleted). In this case both the Dynamic Programming (DP) and the Greedy algorithm have almost the same performance in terms of the number of new links added (Cost on the Y-axis) to the SPRT, and the results for the DP are shown in Fig. 4 as a function of the (common) bandwidth of the newly added links. As shown in Fig. 4, starting at a link bandwidth of approximately $1000Kbps$, congestion free routing is possible in some domains.

Note that the bandwidth of the newly added links is exclusively for management traffic, since data traffic is not routed via the management gateway router. At approximately $2000Kbps$, congestion free routing is possible in all domains. Note also that except for one domain, where 7 new links are needed, all domains can be made congestion free by adding at most 3 links. The Greedy algorithm does not always find the optimal solution as is exhibited in Fig 9 which compares the performance of the Greedy algorithm with that of the DP algorithm for domain D, in terms of the number of new links added to the SPRT.

The congestion free routing computed by the Greedy and the DP algorithms is used to modify the topology of the domains and the congestion of the new rooted SPRT is shown in Fig. 2 and Fig. 3, respectively. The modification involves adding to the domain the new links (added to the original SPRT) at the specified bandwidth, setting their link weight (weight used in shortest path computation) appropriately (equal to the weights of other links on the Management Gateway Router (MGR), to prevent data traffic from transiting via the MGR) and setting the link weights of the links removed from the original SPRT to a high value. Note that since the actual trees found by the Greedy and the DP algorithm may be different, even though they have the same number of new links, the heuristic exhibits different performance for the two algorithms. Both the heuristics exhibit somewhat erratic behavior at small bandwidths (between $1000 - 3000Kbps$), which can be attributed to the possibility that at small bandwidths these new links are prone to higher congestion, since they are at distance 0 from the root. At higher bandwidths ($7000Kbps$ and above) the DP based heuristic is able to alleviate congestion in all the domains, while the Greedy based heuristic is able to significantly bring down the congestion in all domains except the one which had small congestion to begin with. More precisely in some of our simulations of the domain B (with small congestion) applying the heuristic resulted in worse congestion. However, note that in our results we show the better between the before and after congestion values in all our graphs.

The next set of results are for the case when the bandwidth and the cost of the newly added link varies with the node on which it is incident. The link bandwidths vary uniformly within a factor 16 of the minimum link bandwidth and the link costs vary from 1 to 4, with higher bandwidth links costing more. Note that these results show that the Greedy algorithm may not in some cases even find a feasible solution, even though there exists one which the DP algorithm is guaranteed to find. This is exhibited in Fig. 7 and Fig. 8, which shows the total cost of the newly added links for the DP and the Greedy algorithms, respectively, on the Y-axis. Here the X-axis is labeled with the minimum bandwidth of the newly added links. Fig. 5 and Fig. 6 show the performance of the heuristic based on the solution computed using the DP and the Greedy algorithm, respectively, in terms of the impact on the congestion in the new network. Note that in this case the DP based heuristic shows somewhat erratic behavior. A possible reason is that a solution of minimum cost (which the DP is guaranteed to find) may use a large number of links each of small cost and the more the number of links that are modified in the original network, the more the variability in the resulting congestion. However, the Greedy algorithm does not take into account the variability in the cost of the links, and hence attempts to minimize the total

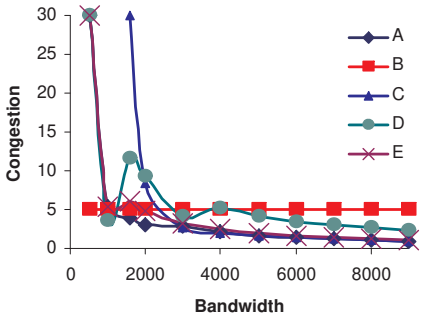

Figure 2: Greedy Algorithm based Heuristic

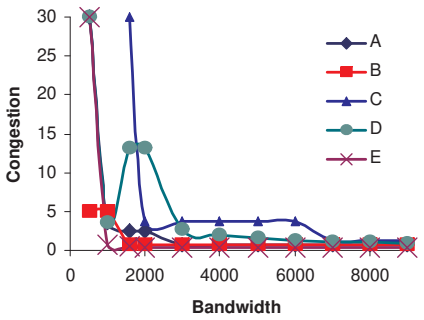

Figure 3: Dynamic Programming (DP) based Heuristic

number of modified links, thus exhibiting less variability.

Our results conclusively establish that congestion can be alleviated for management traffic in the service provider network by topological changes involving a small number of links and their weights and at little extra bandwidth. The heuristics based on the Greedy and the DP algorithm performs quite well and are ideal candidates for limiting this congestion. In addition, at lower bandwidths (of the newly added links), the DP algorithm based heuristic is able to find a good solution even when the Greedy algorithm based heuristic does not succeed in alleviating congestion.

7 Conclusion and open problems

In this paper we designed efficient heuristic for traffic engineering of management traffic in data networks. We showed both analytically and by simulation that these heuristics have

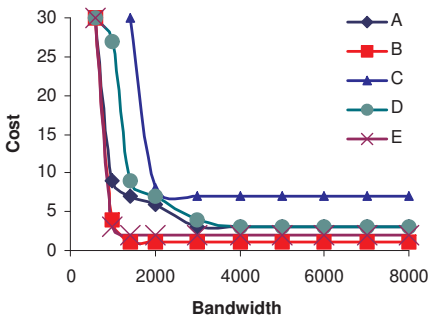

Figure 4: Number of links added as a function of their common bandwidth by the DP

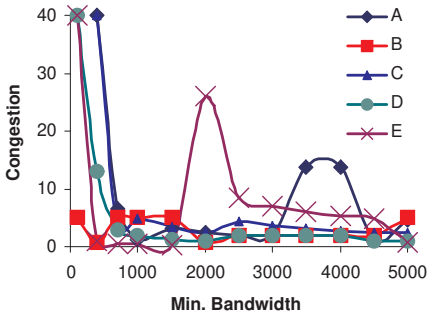

Figure 5: DP based heuristic with non-uniform link bandwidths and costs

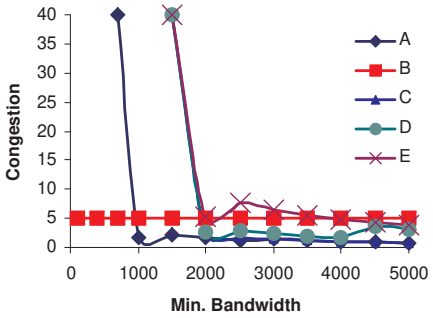

Figure 6: Greedy based heuristic with non-uniform link bandwidths and costs

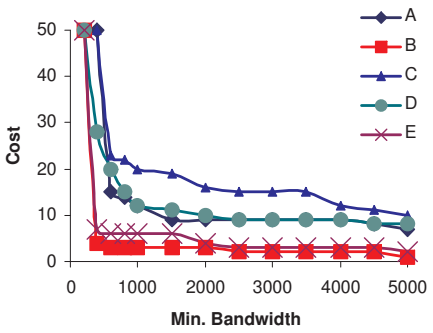

Figure 7: Cost of new links added by DP Algorithm to the SPRT

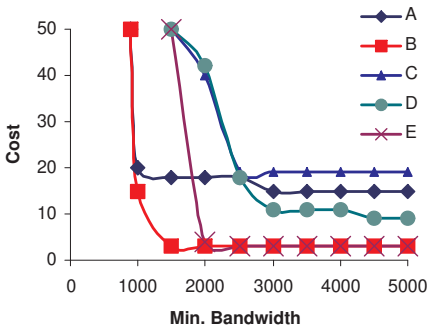

Figure 8: Cost of new links added by Greedy Algorithm to the SPRT

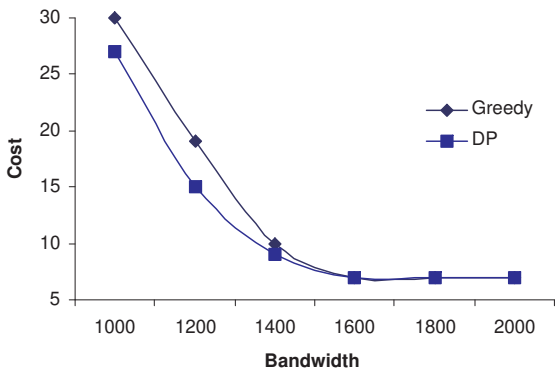

Figure 9: Comparison of Greedy with DP on domain D

good performance in service provider networks. Our work leaves out some open questions which we now describe. We feel that our algorithm for the problem with budget constraints is not complete since it does not guarantee that it would satisfy all or none of the demand at each node (i.e., when the demand at a node is not splittable). It is desirable to develop a PTAS for this problem. It's not clear how to do this with hard budget and capacity constraints. However we feel, that our algorithm probably solves this problem with at most a $1 + \epsilon$ violation of the capacity constraints.

References

[1] D. Applegate and E. Cohen. Making intra-domain routing robust to changing and uncertain traffic demands: Understanding fundamental tradeoffs. In *Proceedings of the ACM SIGCOMM*, 2003.

[2] M. Bienkowski, M. Korzeniowski, and H. Racke. A practical algorithm for constructing oblivious routing schemes. In *Proceedings of the fifteenth annual ACM symposium on Parallel algorithms and architectures (SPAA)*, pages 24–33, 2003.

[3] J. Chen, R. Klienberg, L. Lovasz, R. Rajaraman, R. Sundaram, and A. Vetta. (almost) tight bounds and existence theorems for confluent flows. In *To appear in STOC 2004*, 2003.

[4] J. Chen, R. Rajaraman, and R. Sundaram. Meet and merge: approximation algorithms for confluent flow. In *In proceedings, Symposium on Theory of Computing (STOC)*, pages 373–382, 2003.

[5] L. Ford and D. Fulkerson. *Flows in Network*. Princeton University Press, 1962.

[6] B. Fortz, J. Rexford, and M. Thorup. Internet traffic engineering by optimizing OSPF weights. In *Proceedings of the 19th IEEE INFOCOM*, pages 118–124, 2000.

[7] B. Fortz, J. Rexford, and M. Thorup. Traffic engineering with traditional IP routing protocols. *IEEE Communications Magazine*, 40(10):118–124, 2002.

[8] B. Fortz and M. Thorup. Optimizing OSPF/IS-IS weights in a changing world. *IEEE Journal on Selected Areas in Communications (Special Issue on Recent Advances on Fundamentals of Network Management.)*, 20(4):756–767, 2002.

[9] A. Frank. Connectivity augmentation problems in network design. *Mathematical Programming: state of the art*, pages 34–63, 1994.

[10] C. Harrelson, K. Hildrum, and S. Rao. A polynomial-time tree decomposition to minimize congestion. In *Proceedings of the fifteenth annual ACM symposium on Parallel algorithms and architectures (SPAA)*, pages 34–43, 2003.

[11] S. Iyer, S. Bhattacharrya, N. Taft, and C. Diot. An approach to alleviate link overload as observed on an IP backbone. In *Proceedings of the 22nd IEEE INFOCOM*, 2003.

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

[13] A. Kumar, R. Rastogi, A. Siberschatz, and B. Yener. Algorithms for provisioning virtual private networks in the hose model. In *Proc. of ACM SIGCOMM*, pages 135–148, 2001.

[14] D. Lorenz, A. Orda, D. Raz, and Y. Shavit. How good can IP routing be? In *DIMACS technical report: 2001-17*, 2001.

[15] J. Wang and K. Nahrsdedt. Hop-by-hop routing algorithms for premimum-class traffic in diffserv networks. In *Proc. of IEEE Infocom*, 2002.