Traffic Engineering of Management Flows by Link Augmentations on Confluent Trees

Randeep Bhatia [*] Nicole Immorlica [†] Tracy Kimbrel [‡] Vahab S. Mirrokni [§]
Seffi Naor [¶] Baruch Schieber [||]

February 3, 2005

Abstract

Service providers rely on the management systems housed in their Network Operations Centers (NOCs) to remotely operate, monitor and provision their data networks. Lately there has been a tremendous increase in management traffic due to the growing complexity and size of the data networks and the services provisioned on them. Traffic engineering for management flows to avoid congestion resulting in loss of critical data (e.g. billing records, network alarms etc.) is essential for the smooth functioning of these networks. As is the case with most intra-domain routing protocols the management flows in many of these networks are routed on shortest paths connecting the NOC with the service providers POPs (points of presence). These collection of paths thus form a "confluent" tree rooted at the gateway router connected to the NOC. The links close to the gateway router may form a bottleneck in this tree resulting in congestion. Typically this congestion is alleviated by adding layer two tunnels (virtual links) that bypass the traffic off some links of this tree by routing it directly to the gateway router. The traffic engineering problem is then to minimize the number of virtual links needed for alleviating congestion.

In this paper we formulate a traffic engineering problem motivated by the above mentioned applications. We show that the general versions of this problem are hard to solve. However, for some simpler cases in which the underlying network is a tree, we design efficient algorithms. We use these algorithms as the basis for designing efficient heuristics for alleviating congestion in general non-tree service provider network topologies.

[*]Bell Labs, Lucent Technologies, 2A-244, 600-700 Mountain Avenue, Murray Hill, NJ 07974. E-mail:*randeep@research.bell-labs.com*.

[†]MIT, 232 Vassar Street; Cambridge, MA 02139. E-mail:*nickle@theory.lcs.mit.edu*.

[‡]IBM T.J. Watson Research Center, P.O. Box 218, Yorktown Heights, NY 10598. E-mail:*kimbrel@us.ibm.com*.

[§]MIT, 232 Vassar Street; Cambridge, MA 02139. E-mail:*mirrokni@theory.lcs.mit.edu*.

[¶]Computer Science Department, Technion, Haifa 32000, Israel. E-mail:*naorcs.technion.ac.il*.

[||]IBM T.J. Watson Research Center, P.O. Box 218, Yorktown Heights, NY 10598. E-mail: *sbar@watson.ibm.com*.

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 or it is routed out-of-band on a secure, separate network, dedicated to carrying 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 providers network consists of a number of management domains defined by a partition of the network topology. The domains are 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, and it is only a source or destination for management traffic. Fig. 1 depicts these entities of the management domains. In connection oriented networks (e.g. ATM, MPLS etc.) management and data traffic flows over connections (e.g. Virtual Circuits (VC), Label Switch Paths (LSP) etc.) whose paths are computed using a shortest path algorithm (e.g. Constrained Shortest Path First (CSPF)) and for which resources (e.g. bandwidth) are reserved on the links along the path. However, unlike connections for data traffic which may be provided strict QoS guarantees, very little (or no) bandwidth and resources are allocated for management connections and also no traffic policing and shaping is performed. Thus, typically these connections are routed over shortest paths, regardless of the resource limitations of the links on the shortest paths, and irrespective of the actual amount of management traffic flowing over them. No QoS guarantees are provided to these connections and flow control is done by dropping packets at intermediate routers or switches. The management traffic flows are thus prone to congestion and losses which has an adverse impact on the normal operations of service providers networks.

The collection of paths for the management connections, in a management domain, form a shortest path rooted tree (SPRT), rooted at the management gateway router of the domain. We will refer to this as a “confluent” (tree [6, 5]). Ideally, the management traffic load on a link on this SPRT must not exceed a certain percentage of the links bandwidth. Beyond this increase in percentage, congestion is likely to occur. Even if a mix of data and management traffic is routed on the link, the management traffic is the first to be dropped since it has lower priority (QoS) than a customer’s data traffic. By its very nature the links closer to the root in the SPRT carry more load and are more prone to congestion. A common and intuitive way of alleviating the congestion is then to create Layer 2 tunnels between a node v down in the tree and the root. These Layer 2 tunnels are typically created as bandwidth guaranteed connections over a separate part of the network (many times using explicit routing to prevent the tunnel from taking resources away from the already congested links in the SPRT). Such a tunnel can be used to route all the data coming into a node v directly off to the root thus alleviating the congestion on all the upstream nodes on the path from v to the root in the SPRT. These Layer 2 tunnels can be thought of as virtual links between nodes of the SPRT and the root and are treated as any other physical link for the purpose of route computation. Once added, these links are assigned weights and affect the SPRT of the new network. By choosing low weights for these virtual links and by changing weights on some of the links in the SPRT of the original network the new SPRT can be made to include all the new links and to eliminate a given set of previously congested links in the SPRT. Typically in these connection oriented networks (unless the connections are explicitly routed), paths for connections are constantly re-balanced so that they eventually settle onto the new SPRT. Thus in these networks the goal of traffic engineering is to determine the "minimal" set of Layer 2 tunnels that can be used to alleviate congestion for management flows.

ATM networks - an example: We now present some more details of this traffic engineering problem for ATM networks. Typically in these networks there are 3 types of links that carry management traffic. These can be low bandwidth (e.g. T1) "management links" that are designated to allow only management connections over them. The Connection Admission Control (CAC) rejects any requests for data connections over these links. Typically a portion of all other links' bandwidth ($0-5\%$, almost equivalent to a T1) is also reserved for carrying management traffic (again implemented using CAC). Finally Virtual Links which are created as Virtual Paths (VP) may also be used for carrying management traffic. Typically the management Permanent Virtual Circuits (PVC) are provisioned with no QoS guarantees by setting them up as best effort Unspecified Bit Rate (UBR) class circuits. Thus little or no bandwidth is reserved for these circuits. Since no policing and shaping is done for them and since these circuits have the lowest priority, they are the first to suffer packet losses under link congestion. The ATM networks typically use CSPF algorithm to compute paths for the PVCs and since the CAC reserves no or very little bandwidth for the management PVCs they tend to get routed on a shortest path regardless of the resource limitations of the links of the shortest path and irrespective of the actual amount of management traffic flowing over it. In these networks, service providers typically alleviate congestion of management flows by creating Virtual Paths (VP) of specified bandwidth and QoS guarantees that are then advertised as new management links in the control plane. These Virtual Paths usually provide a shortcut to a node so it can send its management traffic directly to the Management Gateway Router in its domain.

The goal of traffic engineering is to enhance the performance of the network, while expending network resources economically. An efficient scheme for alleviating congestion for management traffic by virtual link augmentation must carefully balance the resulting increase in the node adjacencies and the bandwidth resources dedicated for management traffic with the eventual gain in terms of enhanced performance of the service providers management systems. This is what we attempt to study in this paper. The traffic engineering problem as defined above is very hard to solve in its full generality (in general network topologies). Just the problem of determining whether the existing network can have a congestion-free SPRT for any link weight modifications is NP hard [6]. We show that this remains the case if the underlying network is a tree and new virtual links (each with its own specified capacity) may be added between "any" pair of nodes of the network. 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 shows that in most cases congestion can be eliminated by adding very few links at low bandwidth. Motivated by these results, for general network topologies we propose a heuristic that runs our DP algorithm on the SPRT of the network and uses the computed virtual links for lowering congestion in the original network. Note that although the augmentation of the network with these new links is not guaranteed to alleviate all the congestion our simulation results show that it indeed lowers the congestion considerably.

Our dynamic programming (DP) algorithm for tree based networks are designed to support many natural constraints such as the bandwidth 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.

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 tree rooted typically at the NOC with each node forwarding data from its children to its parent and vice versa [6]. Our techniques are equally applicable for alleviating congestion in these networks as well.

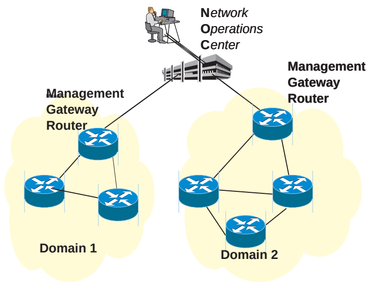

Figure 1: A NOC controlled management domains

1.1 Related work

To the best of our knowledge the problem formulation and solutions presented in this paper are unique when compared to the prior work done in this area of congestion control for management traffic. The work that comes closest to our work is that of [16]. They study a problem of selecting the minimum number of nodes to be used for monitoring in a management domain, such that when management traffic flows on pre-determined routes from the monitored nodes to the monitoring nodes the links stay congestion free. They present an Integer LP formulation and present heuristics without any worst case guarantees for the problem. Their work differs from ours since we do not know the routes a-priori, we reduce congestion by network augmentations and since they assume a distributed monitoring system. The work of [13] considers a similar problem of placement of measurement instrumentation but with additional distance constraints between the monitoring nodes and their monitored nodes. They use graph theoretic results to design heuristics for their problems. Both of these works allow for a distributed monitoring setting where management traffic can flow to a number of monitoring agents in a given domain. However ([4, 2]) the more commonly implemented monitoring schemes in service provider networks depend on a single point in the network (the MGR connected to the NOC) for actively gathering management information of a given domain. This is done for simplicity and cost effectiveness since requiring the distribution of specialized instrumentation software and/or hardware can be cumbersome and expensive to deploy and manage inside the production network. Thus [4] presents under this constraint a problem of computing the minimum number of nodes where measurement of link bandwidth information is sufficient to get a network wide view. The same is also done for measuring link latencies. The work of [4] extends it to the case when links may suffer failures.

There is 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 [10, 8, 9] 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 [12]. The work of [1, 3] is for online routing schemes which achieve nearly optimal utilization on ISP networks even with a fairly limited knowledge of traffic demands.

Problems related to congestion for Confluent flows (another name for single source or destination shortest path routing flows) have been considered before in the literature. In [17] these problems are studied in the context of IP routing. In this context they compare the traditional source invariant IP routing with routing that considers both source and destination and they design source invariant routing schemes with better performance guarantees. Also, in [18], this problem is considered for the purpose of traffic engineering for quality-of-service routing. Confluent flows are also studied in [15] for the purpose of minimizing the total cost of installed capacities on the links of the network under the *hose* model.

Recently, [6] and [5] studied the relation between confluent flow and the well-studied general splittable flow and unsplittable flow problems ([7, 14]). They [6] and [5] present approximation algorithms for the minimum congestion confluent flow problem and the maximum throughput confluent flow problem. In particular, [5] shows a tight $\Theta(\log n)$-approximation for the minimum node-congestion confluent flow problem and a constant factor approximation for the maximum throughput confluent flow problem in general graphs in the special case that the capacity of all nodes is the same. Our problem is different in that we are interested in augmenting the network to get a desired congestion and we are not restricted to uniform capacities. Network augmentation has been considered in different settings, especially in the context of connectivity augmentation. See [11] for a survey. To the best of our knowledge, our work is the first attempt to find a set of minimum cost links to augment a confluent flow with a guaranteed approximation factor.

1.2 Our Results

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 links have uniform cost (e.g. when minimizing number of links), our algorithm finds an optimal solution 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. For this budget-constrained problem we also design a dynamic programming based FPTAS. In Section 5 we design a heuristic for our problem that is applicable to general network topologies. In Section A.1 we show that this heuristic for the general network topologies works very well on service provider networks.

2 Preliminaries and Hardness results

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 shortest path 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 *For trees there is no approximation algorithm for the unrestricted version of our problem 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. $\square$

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.*

We also define a natural variation of this problem which we call the *budget constrained network augmentation problem*. Here the service provider has a budget B and the goal is to find a set of edges of total cost at most B joining the nodes directly to the root, such that using these edges the maximum amount of management traffic can be routed congestion free on the augmented graph. Note that here we allow a source to send fractional amount of its s_v units of traffic.

Both of our augmentation problems are 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. $\square$

Theorem 3 *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. $\square$

Thus, we consider approximation algorithms for our augmentation problems. 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.

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 resulting binary tree may not necessarily translate to a confluent flow in the original tree.)

For each edge $e = (v, u) \in E(T)$ (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 $f((v,r),\rho) = \infty$ 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. 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 4 *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.
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 5 *The above algorithm 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. Again, the best we can hope for is to find a fully polynomial approximation scheme (see hardness result in Section 2). In the following we omit some proofs for lack of space.

We now describe a natural dynamic program to solve this 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 the algorithm in Section 3 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.
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.

$\square$

We omit the proof of the following theorem for lack of space.

Theorem 6 *The Algorithm* **BCNA1** *is an FPTAS for the budget-constrained problem when $\frac{D}{c_{\min}}$ is bounded above by any polynomial in n.*

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.

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.*

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, OPT $\leq$ 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 OPT$' \geq$ OPT$(1-\epsilon_1)$. From this inequality, we have

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

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 consider only the basic network augmentation problem. The heuristic works by computing the SPRT of the given network. It then computes the optimal congestion-free augmentation tree T of this SPRT using the DP presented in Section 3. In other words if the management flows were to use the paths in T then the links are guaranteed to be congestion-free. The heuristic then attempts to "force" the management flows onto the paths in T by setting the costs of the new links to 0 and by setting the costs of the links that are not in T but were in the original SPRT to a large value. Note that as implied by our hardness results this heuristic is not guaranteed to eliminate all the congestion. However our simulation results show that this is an effective heuristic for reducing congestion in service provider networks. The pseudo-code for the heuristic is presented below:

Algorithm.

1. Compute the shortest path rooted tree (SPRT) T (based on the provisioned link weights) rooted at the management gateway router (MGR).
2. Use the tree algorithm (DP in Section 3 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 $T' - T$ and set the weights of these links to 0, so they are likely to be in the new SPRT.

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

$\square$

6 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 good performance in service provider networks. Our work raises several open questions. For the budget constrained problem, it is still open to find an algorithm that 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, however, it is not clear how to do this with hard budget and capacity constraints. We conjecture 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 *ACM SIGCOMM*, 2003.

[2] Y. Bejerano and R. Rastogi. Robust monitoring of link delays and faults in IP networks. In *IEEE INFOCOM*, 2003.

[3] 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.

[4] Y. Breitbart, C. Y. Chan, M. Garofalakis, R. Rastogi, and A. Silberschatz. Efficiently monitoring bandwidth and latency in ip networks. In *IEEE INFOCOM*, 2001.

[5] J. Chen, R. Klienberg, L. Lovasz, R. Rajaraman, R. Sundaram, and A. Vetta. (almost) tight bounds and existence theorems for confluent flows. In *Symposium on Theory of Computing (STOC)*, 2004.

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

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

[8] B. Fortz, J. Rexford, and M. Thorup. Internet traffic engineering by optimizing OSPF weights. In *IEEE INFOCOM*, pages 118–124, 2000.

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

[10] B. Fortz and M. Thorup. Optimizing OSPF/IS-IS weights in a changing world. *IEEE Journal on Selected Areas in Communications*, 20(4):756–767, 2002.

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

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

[13] S Jamin, C. Jin, Y. Jin, D. Raz, Y. Shavitt, and L. Zhang. On the placement of internet instrumentation. In *IEEE INFOCOM*, 2000.

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

[15] 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.

[16] L. Li, M. Thottan, B. Yao, and S. Paul. Distributed network monitoring with bounded link

utilization in ip networks. In *IEEE INFOCOM*, 2003.

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

[18] J. Wang and K. Nahrsdedt. Hop-by-hop routing algorithms for premimum-class traffic in diff-serv networks. In *IEEE Infocom*, 2002.

A Appendix

A.1 Simulations

In this section we present our simulation results on our algorithms for the basic network augmentation problem. Our main goal is to understand the performance of the tree based algorithm when used as a heuristic for alleviating congestion of management traffic in general network topologies (as presented in Section 5).

The metrics that we use to measure the performance of the heuristics is the improvement in the congestion of the network and the cost of the new links needed for the network augmentation. We measure these metrics for different network topologies and as a function of the bandwidths of the links available for the augmentation. For the SPRT of these networks we are interested in the minimum cost of the links needed to alleviate its congestion, as function of the bandwidths of the links available for the augmentation.

The data for our simulations comprises of five independent management domains in a service providers ATM network. We call these domains A, B, C, D, E. Each domain consists of one management gateway (MGR) which forms the root of the SPRT for that domain to which all the management traffic of the domain is destined. Management PVCs are set up between the switches in a domain and the MGR of the domain. Some of the links in these domains have 100% of their bandwidth designated to carry only management traffic and have bandwidth equivalent to a $T1$ ($1.5Mpbs$). Other links which range from $DS1$ to $OC12$ have a fixed proportion of their bandwidth designated for management traffic (approximately $0-5\%$). A link is considered congested for management traffic if the amount of management traffic flowing on the link exceeds the bandwidth designated on the link for management traffic. We define the amount of congestion as the maximum ratio of the management traffic load of a link to the bandwidth of the link designated for management traffic. There are between 50 to 100 switches in each domain each of whose management requirement varies from $50Kbps$ to $2Mbps$. The amount of congestion in the five domains before augmentation is $96, 5, 218, 414, 91$ respectively.

Our first set of results are for the case when all the links used for augmentation have the same bandwidth. These results are shown in Fig. 2 and Fig. 3 as a function of the common bandwidth of the links used for augmentation (on the X axis) in terms of the minimum number of links needed for augmentation of the SPRT to a congestion-free rooted tree. These results for the DP are shown in Fig. 3. Here the common bandwidth of the links used for augmentation is plotted on the X axis and the minimum number of links needed for augmentation (cost) is on the Y axis. As shown in Fig. 3, starting at a link bandwidth of approximately $1000Kbps$, congestion-free rooted tree is possible in some domains. At approximately $2000Kbps$, congestion-free rooted trees are possible in all domains. Note also that except for one domain, where 7 new links are needed, all domains can be made to have a congestion-free rooted tree by adding at most 3 links.

Next the congestion-free rooted tree computed by the DP algorithm for a domain is used to modify the topology of the domain (using the heuristic in Section 5). The congestion of the new rooted SPRT after this augmentation is shown in Fig. 2. Here the congestion value for the links of the augmented network (the new SPRT) is shown on the Y axis which is plotted as a function of the common bandwidth of the links used for augmentation (X axis). Note that the heuristic exhibits 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. However, at higher bandwidths ($7000Kbps$ and above) the DP based heuristic is able to alleviate congestion in all the domains.

The next set of results are for the case when the bandwidth and the cost of the links used for augmentation varies with the node on which they are incident. The bandwidth of these links vary uniformly within a factor 16 of the least bandwidth value, and the link costs vary from 1 to 4, with higher bandwidth links costing more. Here we plot the minimum bandwidth of the links used for augmentation on the X axis. Fig. 5 depicts the minimum total cost of the links (on the Y axis) needed for guaranteeing congestion-free rooted trees for the DP. Fig. 4 shows the performance of the heuristic (defined in Section 5). in terms of the impact on the congestion in the new network. Here the congestion value for the links of

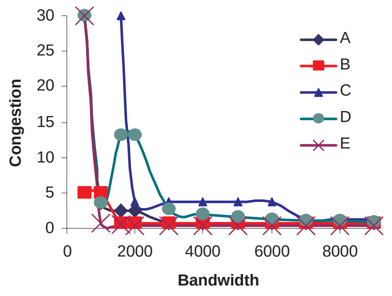

Figure 2: Dynamic Programming (DP) based Heuristic

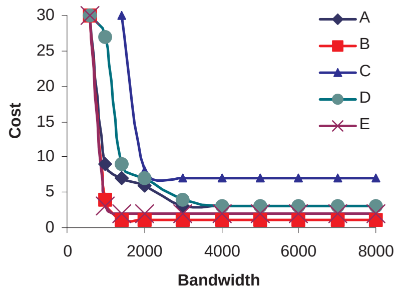

Figure 3: Number of links added as a function of their common bandwidth by the DP

the augmented network (the new SPRT) is shown on the Y axis which is plotted as a function of the common bandwidth of the links used for augmentation (X axis). Note that in this case the 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.

Our results show that in almost all cases the congestion for the management flows can be brought down significantly by using the heuristics defined in Section 5 based on the DP algorithm. In addition only a small number of augmenting links of low bandwidth are needed for this purpose.

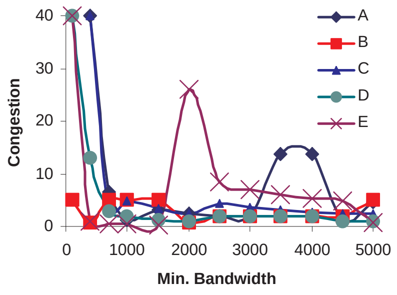

Figure 4: DP based heuristic with non-uniform link bandwidths and costs

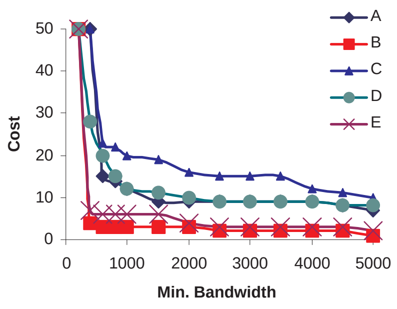

Figure 5: Cost of new links added by DP Algorithm to the SPRT