Algorithms for Provisioning Virtual Private Networks in the Hose Model

Amit Kumar*
Rajeev Rastogi
Avi Silberschatz
Bulent Yener

600 Mountain Avenue
Murray Hill, NJ 07974

Abstract

Virtual Private Networks (VPNs) provide customers with predictable and secure network connections over a shared network. The recently proposed *hose* model for VPNs allows for greater flexibility since it permits traffic to and from a hose endpoint to be arbitrarily distributed to other endpoints. In this paper, we develop novel algorithms for provisioning VPNs in the hose model. We connect VPN endpoints using a *tree* structure and our algorithms attempt to optimize the total bandwidth reserved on edges of the VPN tree. We show that even for the simple scenario in which network links are assumed to have infinite capacity, the general problem of computing the optimal VPN tree is NP-hard. Fortunately, for the special case when the ingress and egress bandwidths for each VPN endpoint are equal, we can devise an algorithm for computing the optimal tree whose time complexity is $O(mn)$, where m and n are the number of links and nodes in the network, respectively. We present a novel integer programming formulation for the general VPN tree computation problem (that is, when ingress and egress bandwidths of VPN endpoints are arbitrary) and develop an algorithm that is based on the primal-dual method. Finally, we extend our proposed algorithms for computing VPN trees to the case when network links have capacity constraints. We show that in the presence of link capacity constraints, computing the optimal VPN tree is NP-hard even when ingress and egress bandwidths of each endpoint are equal. Our experimental results with synthetic network graphs indicate that the VPN trees constructed by our proposed algorithms dramatically reduce bandwidth requirements (in many instances, by more than a factor of 2) compared to scenarios in which *Steiner trees* are employed to connect VPN endpoints.

1 Introduction

Virtual Private Networks (VPNs) are becoming an increasingly important source of revenue for Internet Service Providers (ISPs). Informally, a VPN establishes connectivity between a set of geographically dispersed endpoints over a shared network infrastructure. The goal is to provide VPN endpoints with a service comparable to a private dedicated network established with *leased lines*. Thus, providers of VPN services need to address the QoS and security issues associated with deploying a VPN over a shared IP network. In recent years, substantial progress in the technologies for IP security [KA98, DGG+98] have enabled existing VPN service offerings to provide customers with a level of privacy comparable to that offered by a dedicated line. However, ISPs have been slow to offer customers with guaranteed bandwidth VPN services since IP networks in the past had little support for enforcing QoS in the network. The recent emergence of IP technologies like MPLS and RSVP, however, have made it possible to realize IP-based VPNs that can provide end customers with QoS guarantees. In this paper, we address the problem of provisioning VPN services with QoS guarantees, a problem which has received little attention from the research community.

*Current Address: Department of Computer Science, Cornell University.

1.1 The Hose Model

There are two popular models for providing QoS in the context of VPNs - the "pipe" model and the "hose" model [DGG+98, DR00]. In the pipe model, the VPN customer specifies QoS requirements between every pair of VPN endpoints. Thus, the pipe model requires the customer to know the complete traffic matrix; that is, the load between every pair of endpoints. However, the number of endpoints per VPN is constantly increasing and the communication patterns between endpoints are becoming increasingly complex. As a result, it is almost impossible to predict traffic characteristics between pairs of endpoints required by the pipe model.

The hose model alleviates the above-mentioned shortcomings of the pipe model. In the hose model, the VPN customer specifies QoS requirements per VPN endpoint and not every pair of endpoints. Specifically, associated with each endpoint, is a pair of bandwidths – an *ingress* bandwidth and an *egress* bandwidth. The ingress bandwidth for an endpoint specifies the incoming traffic from all the other VPN endpoints into the endpoint, while the egress bandwidth is the amount of traffic the endpoint can send to the other VPN endpoints. Thus, in the hose model, the VPN service provider supplies the customer with certain guarantees for the traffic that each endpoint sends to and receives from other endpoints of the same VPN. The customer does not have to specify how this traffic is distributed among the other endpoints. As a result, in contrast to the pipe model, the hose model does not require a customer to know its traffic matrix, which in turn, places less burden on a customer that wants to use the VPN service.

In summary, the hose model provides customers with the following advantages over the pipe model [DGG+98]:

1. **Ease of Specification.** Only one ingress and egress bandwidth per hose endpoint needs to be specified, compared to bandwidth for each pipe between pairs of hose endpoints.

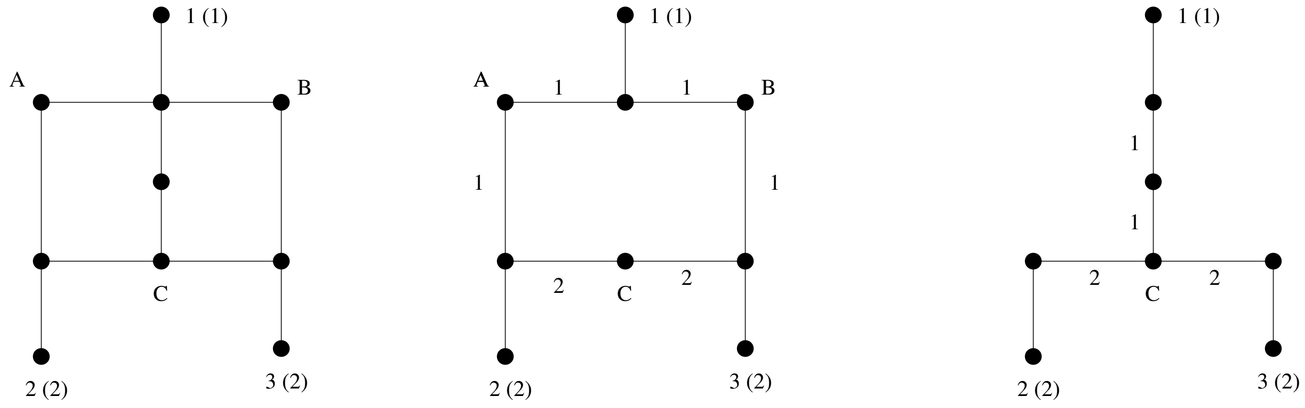
2. **Flexibility.** Traffic to and from a hose endpoint can be distributed arbitrarily over other endpoints as long as the ingress and egress bandwidths of each hose endpoint are not violated.

3. **Multiplexing Gain.** Due to statistical multiplexing gain, hose ingress and egress bandwidths can be less than the aggregate bandwidth required for a set of point to point pipes.

4. **Characterization.** Hose requirements are easier of characterize because the statistical variability in the individual source-destination traffic is smoothed by aggregation into hoses.

The multiplexing gain due to the hose model (point 3 above) can result in significant improvements in the utilization of network resources. For example, consider a VPN used to offer VoIP service to customers located at VPN endpoints in NY, LA and Chicago. Suppose N is the maximum volume of calls out of NY (bounded by the number of customers being served by the VPN endpoint in NY). Further, suppose that there is a large variability in the destinations of the calls with as many as 80% of all calls out of NY being directed to LA at some times and to Chicago at other times. Thus, in the pipe model, two pipes, one from NY to LA and another from NY to Chicago, each with bandwidth to carry $0.8 * N$ calls need to be reserved. As a result, in the pipe model, the total bandwidth reserved for traffic out of NY is $1.6 * N$ units. In contrast, in the hose model, bandwidth is reserved only for the aggregate traffic out of NY, which is N units. Thus, by reserving bandwidth for the aggregate traffic out of VPN endpoints, the hose model improves network bandwidth utilization because of statistical multiplexing effects.

From the above discussion, it follows that the hose model provides VPN customers with a simple mechanism for specifying bandwidth requirements and enables VPN service providers to utilize network bandwidth more efficiently. However, in order to realize these benefits, efficient algorithms must be devised for provisioning hoses. These hose provisioning algorithms need to set up paths between every pair of VPN endpoints such that the aggregate bandwidth reserved on the links traversed by the paths is minimum. A naive algorithm that sets up independent shortest paths between every pair of endpoints, however, could lead to excessive bandwidth being reserved. The reason for this is that the hose model provides the flexibility for traffic from a hose endpoint to be arbitrarily distributed to other endpoints. Consequently, the distribution of traffic between VPN endpoints is non-deterministic and the provisioning algorithms need to reserve sufficient bandwidth to accommodate the worst-case



(a) Graph (b) Independent Shortest Paths (c) Link Sharing Among Paths

Figure 1: Link Sharing Among Paths to Reduce Reserved Bandwidth

traffic distribution among endpoints that meets the ingress and egress bandwidth constraints of hose endpoints. Intuitively, in order to conserve bandwidth and realize the multiplexing benefits of the hose model, paths entering into and originating from each hose endpoint need to share as many links as possible. Thus, sophisticated hose provisioning algorithms need to be developed to ensure that the amount of bandwidth reserved in order to meet the hose traffic requirements is minimum.

Example 1.1: Consider the network graph in Figure 1(a). The 3 VPN hose endpoints 1, 2 and 3 have bandwidth requirements of 1, 2 and 2 units, respectively (each endpoint has equal ingress and egress bandwidths). Figure 1(b) depicts the bandwidth reserved on relevant links of the network when a naive independent shortest paths approach is used to connect VPN endpoints. For instance, the shortest path between 1 and 2 passes through A, while the shortest path between endpoints 2 and 3 passes through C. Also, 1 unit of bandwidth needs to be reserved (in each direction) on the two links incident on A and on the shortest path from 1 to 2 since endpoint 1 can send/receive at most 1 unit of traffic. Similarly, the bandwidth reserved on the two links incident on C is 2 units, the minimum of the bandwidth requirements of endpoints 2 and 3. Thus, the total reserved bandwidth using independent shortest paths is 8 units (only considering one direction for each link).

The reserved bandwidth can be reduced from 8 to 6 by exploiting link sharing among paths connecting the VPN endpoints as illustrated in Figure 1(c). Here, both paths from endpoint 1 to the two other endpoints pass through C, thus allowing the two links connecting 1 to C to be shared between them. (Note that the path from 1 to 2 passing through C is longer than the path from 1 to 2 through A). Further, since endpoint 1 cannot receive or send more than 1 unit of traffic, the bandwidth reserved on each of the two links is 1 unit (in each direction) and is shared between the two paths. Thus, the total bandwidth reserved decreases from 8 to 6 as a result of link sharing between paths. ∎

Note that provisioning pipes between each pair of VPN endpoints in the pipe model is somewhat simpler since the traffic between every pair of endpoints is fixed and is input to the provisioning algorithm. Thus, the VPN provisioning problem simply reduces to that of computing a set of fixed bandwidth paths between VPN endpoint pairs, which is an instance of the well-studied multicommodity flow problem [AMO93, Hoc97]. However, as mentioned earlier, the drawback of the pipe model is the difficulty of capturing and specifying bandwidth requirements between each pair of VPN endpoints. Thus, the hose model trades off provisioning simplicity for ease of specification and multiplexing gains.

1.2 Our Contributions

In this paper, we develop novel algorithms for provisioning VPNs in the hose model. In order to take advantage of the multiplexing gain possible due to hoses, we connect VPN endpoints using a tree structure (instead of independent point-to-point paths between VPN endpoints). A VPN tree has several benefits, which we list below.

1. **Sharing of bandwidth reservation.** A single bandwidth reservation on a link of the tree can be shared by the entire traffic between the two sets of VPN endpoints connected by the link. Thus, the bandwidth reserved on the link only needs to accommodate the aggregate traffic between the two sets of VPN endpoints.

2. **Scalability.** For a large number of VPN endpoints, a tree structure scales better than point-to-point paths between all VPN endpoint pairs.

3. **Simplicity of Routing.** The structural simplicity of trees ensures that if MPLS [DR00] is used for setting up paths between VPN endpoints, then fewer labels are required and label stacks on packets are not as deep.

4. **Ease of Restoration.** Trees also simplify restoration of paths in case of link failures, since all paths traversing a failed link can be restored as a single group, instead of each path being restored separately.

We develop algorithms for computing *optimal* VPN trees; that is, trees for which the amount of total bandwidth reserved on edges of the tree is minimum. Initially, we assume that network links have infinite capacity, and show that even for this simple scenario, the general problem of computing the optimal VPN tree is NP-hard. However, for the special case when the ingress and egress bandwidths for each VPN endpoint are equal, we are able to devise a *breadth first search* algorithm for computing the optimal tree whose time complexity is $O(mn)$, where m and n are the number of links and nodes in the network, respectively. We present a novel integer programming formulation for the general VPN tree computation problem (that is, when ingress and egress bandwidths of VPN endpoints are arbitrary) and develop an algorithm that is based on the primal-dual method [Hoc97]. Finally, we extend our proposed algorithms for computing VPN trees to the case when network links have capacity constraints. We show that in the presence of link capacity constraints, computing the optimal VPN tree is NP-hard even when ingress and egress bandwidths of each endpoint are equal. Further, we also show that computing an approximate solution that is within a constant factor of the optimum is as difficult as computing the optimal VPN tree itself.

In [DGG+98], the authors suggest that a *Steiner tree* can be employed to connect VPN endpoints. However, our experimental results with synthetic network graphs indicate that the VPN trees constructed by our proposed algorithms require dramatically less bandwidth to be reserved (in many instances, by more than a factor of 2) compared to *Steiner trees*. Further, among the three algorithms, the primal-dual algorithm performs the best, reserving less bandwidth than both the breadth first search and Steiner tree algorithms over a wide range of parameter values.

2 System Model and Problem Formulation

We model the network as a graph $G = (V, E)$ where V is a set of nodes and E is a set of *bidirectional* links between the nodes. Each link (i, j) has associated capacities in the two directions – we denote the capacity from node i to j by L_{ij} and the capacity (in the opposite direction) from node j to i by L_{ji}. Also, R_{ij} denotes the residual capacity (that is, the available bandwidth) of link (i, j) for carrying traffic from node i to j.

In the hose model, each VPN specification consists of the following two components: (1) A set of nodes $P \subseteq V$ corresponding to the VPN endpoints, and (2) For each node $i \in P$, the hose ingress and egress bandwidths B_i^{in} and B_i^{out}, respectively. Without loss of generality, we assume that each node $i \in P$ is a leaf, that is, there is a single (bidirectional) link incident on i[1]. As mentioned earlier, we consider tree structures to connect the VPN endpoints in P since trees are scalable, simplify routing and restoration. Further, trees allow the bandwidth

[1] In case there is a VPN endpoint $i \in P$ that is not a leaf, then we simply introduce a new node j in G and a new link (i, j) with very large capacity, and replace node i in P with node j.

Symbol	Description
$G = (V, E)$	Graph with nodes V and *bidirectional* edges E
m	Number of links in graph G ($\|E\|$)
n	Number of nodes in graph G ($\|V\|$)
L_{ij}	Capacity of link (i, j) in the direction from node i to j
R_{ij}	Residual capacity of link (i, j) in the direction from node i to j
i, j, l, u, v, w	Generic symbols for nodes in G
P	Set of VPN endpoints
T	Generic notation for VPN tree
T_v	VPN tree rooted at node v
B_i^{in}	Ingress bandwidth for node i
B_i^{out}	Egress bandwidth for node i
$T_i^{(i,j)}$	Component of tree T containing i when link (i, j) is deleted from T
$P_i^{(i,j)}$	VPN endpoints contained in $T_i^{(i,j)}$
$C_T(i, j)$	Bandwidth reserved on link (i, j) of VPN tree T
C_T	Sum of bandwidths reserved on all links of VPN tree T
$d_T(i, j)$	Distance (in number of links) between nodes i and j in tree T
$d_G(i, j)$	Length of the shortest path (in number of links) between nodes i and j in graph G

Table 1: Notation Used in the Paper

reserved on a link to be shared by the traffic between the two sets of VPN endpoints connected by the link. Thus, the bandwidth reserved on a link must be equal to the maximum possible aggregate traffic between the two sets of endpoints connected by the link.

Before we can compute the exact bandwidth to be reserved on each link of VPN tree T whose leaves are nodes from P, we need to develop some notation. For a link (i, j) in tree T, we denote by $T_i^{(i,j)}/T_j^{(i,j)}$ the connected component of T containing node i/j when link (i, j) is deleted from T. Also, let $P_i^{(i,j)}$ and $P_j^{(i,j)}$ denote the set of VPN endpoints in $T_i^{(i,j)}$ and $T_j^{(i,j)}$, respectively. Table 1 describes the notation used in the remainder of the paper.

Observe that all traffic from one VPN endpoint to another traverses the unique path in the VPN tree T between the two endpoints. Now consider link (i, j) that connects the two sets of VPN endpoints $P_i^{(i,j)}$ and $P_j^{(i,j)}$. The traffic from node i to j cannot exceed $\min\{\sum_{l \in P_i^{(i,j)}} B_l^{out}, \sum_{l \in P_j^{(i,j)}} B_l^{in}\}$, that is, the minimum of the cumulative egress bandwidths of endpoints in $P_i^{(i,j)}$ and the sum of ingress bandwidths of endpoints in $P_j^{(i,j)}$. This is because the only traffic that traverses link (i, j) from i to j is the traffic originating from endpoints in $P_i^{(i,j)}$ and directed towards endpoints in $P_j^{(i,j)}$. The bound on the former is $\sum_{l \in P_i^{(i,j)}} B_l^{out}$, while the latter is bounded by $\sum_{l \in P_j^{(i,j)}} B_l^{in}$. Thus, the bandwidth to be reserved on link (i, j) of T in the direction from i to j is given by $C_T(i, j) = \min\{\sum_{l \in P_i^{(i,j)}} B_l^{out}, \sum_{l \in P_j^{(i,j)}} B_l^{in}\}$. Similarly, the bandwidth that must be reserved on link (j, i) in the direction from j to i can be shown to be $C_T(j, i) = \min\{\sum_{l \in P_i^{(i,j)}} B_l^{in}, \sum_{l \in P_j^{(i,j)}} B_l^{out}\}$. Note that $C_T(i, j)$ may not be equal to $C_T(j, i)$.

Thus, the total bandwidth reserved for tree T is given by $C_T = \sum_{(i,j) \in T} C_T(i, j)$ (note that (i, j) and (j, i) are considered as two distinct links in T). Further, since we are interested in minimizing the reserved bandwidth for tree T, the problem of computing the optimal VPN tree becomes the following:

Problem Statement (Optimal VPN Tree without Link Capacity Constraints): Given a set of VPN endpoints P, and their ingress and egress bandwidths, compute a VPN tree T whose leaves are nodes in P and for which C_T is minimum. ∎

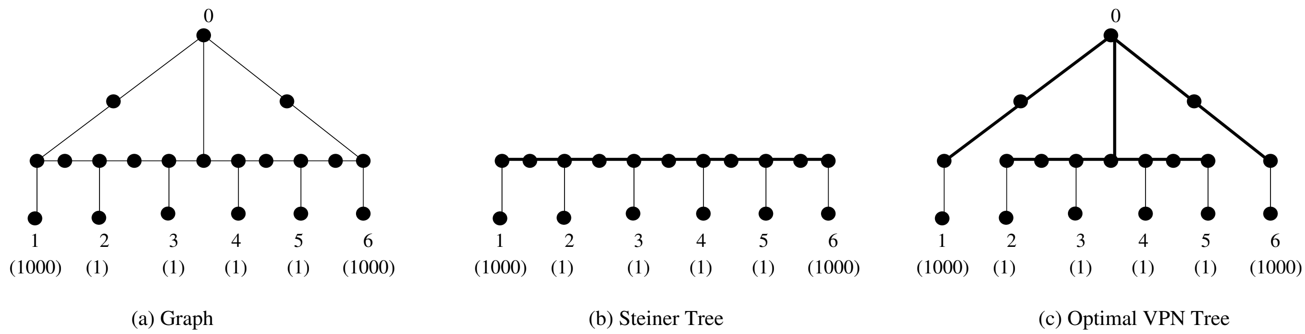


Figure 2: Example of Suboptimal Steiner Tree

In [DGG+98], it has been suggested that a Steiner tree can be used to connect the VPN endpoints. However, even though a Steiner tree has the smallest number of links, it may be suboptimal, as illustrated by the following example.

Example 2.1: Consider the network graph shown in Figure 2(a). Nodes $1, 2, \ldots, 6$ are the VPN endpoints and $B_1^{in} = B_1^{out} = B_6^{in} = B_6^{out} = 1000$ while $B_i^{in} = B_i^{out} = 1$ for $i = 2, \ldots, 5$. Figure 2(b) shows the Steiner tree connecting the VPN endpoints in P and containing 16 edges. The total bandwidth reserved on the edges of the Steiner tree (in a single direction) is given by $1000 * 2 + 1001 * 2 + 1002 * 2 + 1001 * 2 + 1000 * 2 = 10008$ (this excludes the 2006 units that need to be reserved on the links incident on the VPN endpoints in P). The reason for this is that 1000 units need to be reserved on the two links connecting endpoints 1 and 2, 1001 units need to be reserved on the two links connecting endpoints 2 and 3, and so on. However, the optimal VPN tree containing 17 links is shown in Figure 2(c). The cumulative bandwidth reserved on all the links of the optimal tree is $1000*2+1000*2+4+2*2+1*2+1*2 = 4012$ (again, in a single direction per link and excluding the 2006 units that need to be reserved on the links incident on the VPN endpoints in P). This is because the bandwidth reserved on the two links connecting endpoints 1 and 0 is 1000, the bandwidth for the two links between endpoints 6 and 0 is 1000, the bandwidth for the link connecting 0 to endpoints $2, \ldots, 5$ is 4 units, and for the two links connecting endpoints 3 and 4 is 2 units, and so on. ∎

Thus, in the above example, the bandwidth reserved for the Steiner tree is more than twice the optimum bandwidth. Further, note that, in the example, the performance of the Steiner tree can be made arbitrarily worse compared to the optimum (by increasing the number of VPN endpoints between endpoints 1 and 6).

In our earlier formulation of the optimal VPN tree computation problem, we assumed that links do not have capacity constraints. In the following, we present the problem formulation that also takes into account the residual bandwidths of network links.

Problem Statement (Optimal VPN Tree with Link Capacity Constraints): Given a set of VPN endpoints P, and their ingress and egress bandwidths, compute a tree T whose leaves are nodes in P such that $C_T(i, j) \leq R_{ij}$ and C_T is minimum. ∎

In the remainder of the paper, we will refer to $C_T(i, j)$ as the cost of link (i, j) in tree T and C_T as the cost of tree T. In the following two sections (Sections 3 and 4), we first develop algorithms to compute the optimal VPN tree ignoring link capacity constraints. We then show how the algorithms can be extended to handle link capacity constraints in Section 5.

3 Symmetric Ingress and Egress Bandwidths

For symmetric ingress and egress bandwidths, that is, when $B_i^{in} = B_i^{out}$ for each VPN endpoint i, one can devise efficient algorithms for computing the minimum cost VPN tree if links do not have capacity constraints. In this section, we present a polynomial time algorithm for computing the optimal VPN tree for the symmetric bandwidth case under the assumption that the residual capacity of each edge is large. Since the ingress and egress bandwidths are equal, in the following, we simply drop the superscripts in and out, and denote the bandwidth for endpoint i simply by B_i.

Before presenting our algorithm for computing the optimal tree (that is, tree with minimum cost), we first develop some intuition for the cost C_T of a tree T. Recall (from the previous section) that $C_T = \sum_{(i,j) \in T} C_T(i, j)$. For a set of leaves $L \subseteq P$, we define $B(L)$ as $\sum_{l \in L} B_l$. Thus, $C_T(i, j) = \min\{B(P_i^{(i,j)}), B(P_j^{(i,j)})\}$. It is straightforward to observe that $C_T(i, j) = C_T(j, i)$, that is, the bandwidth reserved on link (i, j) in the two directions is equal. Now suppose that for a tree T and a node v in T, we define the quantity $Q(T, v)$ to be $2 * \sum_{l \in P} B_l * d_T(v, l)$, where the sum is over all leaves l, and $d_T(v, l)$ denotes the length of the unique v to l path in T. Then, for any tree T whose leaves are nodes in P, we can show that C_T satisfies the following two properties:

- **Property 1.** There exists a node $w \in T$ such that $C_T = Q(T, w)$.
- **Property 2.** For all nodes $v \in T$, $C_T \leq Q(T, v)$.

In order to show the above two properties for tree T, we construct a directed tree T_{dir} from T by giving a direction to each edge $e = (i, j)$ of T as follows:

- if $B(P_i^{(i,j)}) < B(P_j^{(i,j)})$, then direct the edge towards i.
- if $B(P_j^{(i,j)}) < B(P_i^{(i,j)})$, then direct the edge towards j.
- if $B(P_i^{(i,j)}) = B(P_j^{(i,j)})$, then direct the edge towards the component which contains a particular leaf, say, $\hat{l}$.

Clearly, T_{dir} must contain a node whose indegree is 0 — we denote this node in T_{dir} with no incoming edges by $a(T)$. We show that $a(T)$ is indeed unique and $C_T = Q(T, a(T))$. For this, we prove some properties about T_{dir} in the following lemma.

Lemma 3.1: *Every edge e in T_{dir} is directed away from $a(T)$.* ▮

Proof: Let $e = (i, j)$ be an edge in tree T such that i is closer to $a(T)$ than j in T. We show that e is directed from i to j in T_{dir}. Consider the path in T from $a(T)$ to i. We know that the first edge (u, v) along the path is directed away from $u = a(T)$. So, $B(P_v^{(u,v)}) \leq B(P_u^{(u,v)})$. Since (u, v) is the first edge of the path from $a(T)$ to i, $P_u^{(u,v)} \subseteq P_i^{(i,j)}$ and also, $P_j^{(i,j)} \subseteq P_v^{(u,v)}$. Thus, we get, $B(P_j^{(i,j)}) \leq B(P_v^{(u,v)}) \leq B(P_u^{(u,v)}) \leq B(P_i^{(i,j)})$. If $B(P_j^{(i,j)}) < B(P_i^{(i,j)})$, then edge e is directed from i to j and we are done. The only other possibility is $B(P_j^{(i,j)}) = B(P_i^{(i,j)})$. But then, it must be the case that $P_u^{(u,v)} = P_i^{(i,j)}$, $P_v^{(u,v)} = P_j^{(i,j)}$ and $B(P_u^{(u,v)}) = B(P_v^{(u,v)})$. As a result, it follows that $u = i$ and $v = j$, and since edge (u, v) is directed from u to v, edge (i, j) must also be directed from i to j. ▮

Note that from the above lemma, one can easily show that $a(T)$ is unique since every other node in T_{dir} has an edge directed into it (and consequently, an indegree of 1). In the following lemma, we prove that Property 1 mentioned above holds for $w = a(T)$.

Lemma 3.2: *The cost of tree T, $C_T = Q(T, a(T))$.* ▮

Proof: Let l be a leaf and $e = (i, j)$ be an edge on the path from $a(T)$ to l (node i being closer to $a(T)$) — note that B_l contributes to both $C_T(i, j)$ and $C_T(j, i)$. The reason for this is that $C_T(i, j)$ is $B(P_j^{(i,j)})$ since due to Lemma 3.1, the edge is directed from i to j in T_{dir} and so $B(P_j^{(i,j)}) \leq B(P_i^{(i,j)})$. Thus, since $l \in P_j^{(i,j)}$, B_l contributes to each edge on the path from $a(T)$ to l, and to no other edge. This proves the lemma. ▮

Property 2 is a straightforward corollary of the following lemma (since $C_T = Q(T, a(T))$ due to Lemma 3.2).

Lemma 3.3: *Let v be any node in T. Then, $Q(T, a(T)) \leq Q(T, v)$.* ▮

Proof: We first show the following result: suppose $e = (i, j)$ is an edge that is directed from i to j in T_{dir}. Then, $Q(T, i) \leq Q(T, j)$. We know that $B(P_j^{(i,j)}) \leq B(P_i^{(i,j)})$. Now,

$$\begin{aligned}
Q(T, i) - Q(T, j) &= 2 * \sum_{l \in P_i^{(i,j)}} B_l * (d_T(i, l) - d_T(j, l)) + 2 * \sum_{l \in P_j^{(i,j)}} B_l * (d_T(i, l) - d_T(j, l)) \\
&= -2 * \sum_{l \in P_i^{(i,j)}} B_l + 2 * \sum_{l \in P_j^{(i,j)}} B_l \\
&= 2 * B(P_j^{(i,j)}) - 2 * B(P_i^{(i,j)}) \ \leq \ 0
\end{aligned}$$

```
procedure ComputeTreeSymmetric(G, P)
T_opt := ∅
for each node v in G {
    T_v := v
    openQ := {v}
    while (openQ ≠ ∅){
        dequeue first node u from openQ
        for each edge (u, w) in G such that w is not in T_v {
            add edge (u, w) to T_v
            append node w to end of openQ
    }
    prune leaves of T_v that do not correspond to VPN endpoints (that is, do not belong to P)
    if C_{T_v} < C_{T_opt}
        T_opt := T_v
}
return T_opt
```

Figure 3: Algorithm for Computing Optimal Tree for Symmetric Bandwidths

Now, we can prove the lemma. Consider the path from $a(T)$ to v in T_{dir}. By Lemma 3.1, all edges along the path are directed away from $a(T)$. So, repeated application of the result proved above (for each edge along the path) proves the lemma. ∎

From Properties 1 and 2, it follows that for the optimal tree T_{opt}, $C_{T_{opt}} = Q(T_{opt}, a(T_{opt}))$. Thus, if we could compute, for each node v, the tree T_v such that $Q(T_v, v)$ is minimum, then the optimal tree is simply the tree T_v with the minimum $Q(T_v, v)$. Figure 3 contains the procedure for computing the optimal tree T_{opt}. For each node v in G, Procedure ComputeTreeSymmetric computes a breadth first spanning tree rooted at v — we show in Lemma 3.4 below that this tree T_v minimizes the value of $Q(T_v, v)$ among possible trees rooted at v. The procedure then outputs the tree T_v for which $Q(T_v, v)$ is minimum — let $T_{\hat{v}}$ be this tree.

In the following, we show that $C_{T_{\hat{v}}} = C_{T_{opt}}$ where T_{opt} is the optimal tree. First note the following important fact for the breadth first tree T_v rooted at node v.

Lemma 3.4: *Let T be any tree and v be a node in G. Then, $Q(T_v, v) \le Q(T, v)$.* ∎

Proof: Note that $d_{T_v}(v, l) \le d_T(v, l)$ for each leaf l in T. So, the lemma is true. ∎

In the next lemma, we show the following important relationship between $T_{\hat{v}}$ and T_{opt} that enables us to subsequently show that their costs are equal. Note that due to Lemma 3.2, $C_{T_{opt}} = Q(T_{opt}, a(T_{opt}))$.

Lemma 3.5: $Q(T_{\hat{v}}, \hat{v}) \le Q(T_{opt}, a(T_{opt})) = C_{T_{opt}}$. ∎

Proof: Lemma 3.4 above implies that $Q(T_{a(T_{opt})}, a(T_{opt})) \le Q(T_{opt}, a(T_{opt}))$. Since $Q(T_{\hat{v}}, \hat{v})$ is minimum among all nodes v in the graph, $Q(T_{\hat{v}}, \hat{v}) \le Q(T_{a(T_{opt})}, a(T_{opt}))$ and the lemma follows. ∎

Theorem 3.6: $C_{T_{\hat{v}}} = C_{T_{opt}}$. ∎

Proof: First, due to Lemma 3.2, $C_{T_{\hat{v}}} = Q(T_{\hat{v}}, a(T_{\hat{v}}))$. Further, Lemma 3.3 implies that $Q(T_{\hat{v}}, a(T_{\hat{v}})) \le Q(T_{\hat{v}}, \hat{v})$, which by Lemma 3.5 is at most $C_{T_{opt}}$. Thus, it must be the case that $C_{T_{\hat{v}}}$ is optimum. ∎

It is straightforward to observe that the time complexity of Procedure ComputeTreeSymmetric is $O(mn)$, where $n = |V|$ is the number of nodes and $m = |E|$ is the number of edges in G. This is because the outermost **for** loop iterates over every node in G, and the body of the for loop, in the worst case, considers every edge in G.

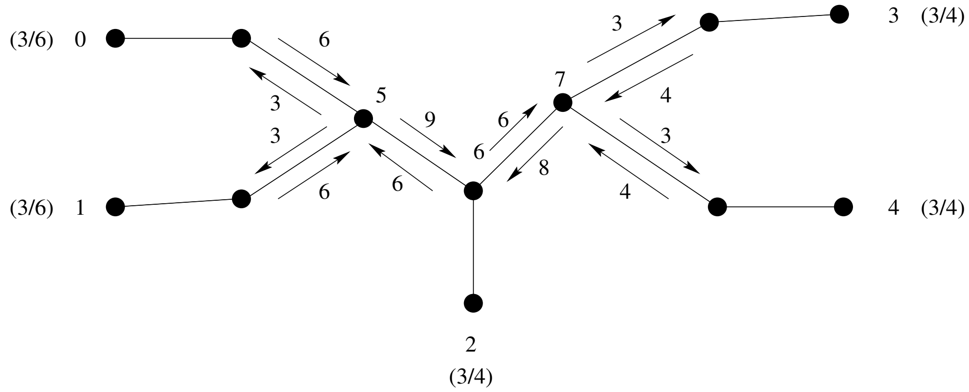


Figure 4: Example of Tree Cost for Asymmetric Bandwidths

4 Asymmetric Ingress and Egress Bandwidths

In this section, we address the case when VPN endpoint bandwidth requirements are asymmetric, that is, for a VPN endpoint j, B_j^{in} and B_j^{out} may be unequal. Asymmetric ingress and egress bandwidths complicate the VPN tree computation problem since for a VPN tree T connecting the VPN endpoints, the bandwidth reserved along edge (i, j) of T may not be identical in the two directions – that is, $C_T(i, j)$ may not be equal to $C_T(j, i)$. This is because for an edge (i, j), $C_T(i, j) = \min\{\sum_{l \in P_i^{(i,j)}} B_l^{out}, \sum_{l \in P_j^{(i,j)}} B_l^{in}\}$ and $C_T(j, i) = \min\{\sum_{l \in P_i^{(i,j)}} B_l^{in}, \sum_{l \in P_j^{(i,j)}} B_l^{out}\}$. Thus, in the asymmetric case, since B_l^{in} and B_l^{out} may not be equal, $C_T(i, j)$ and $C_T(j, i)$ may not be equal. Note that this is different from the symmetric bandwidths case in which the bandwidths reserved in both directions along an edge (i, j) of a tree T were equal.

Example 4.1: Consider the VPN tree shown in Figure 4 connecting the VPN endpoints in $P = \{0, 1, \ldots, 4\}$. The bandwidth requirements for the endpoints are as follows: for endpoints 0 and 1, $B^{in} = 3$ and $B^{out} = 6$, and for endpoints 2, 3 and 4, $B^{in} = 3$ and $B^{out} = 4$. The bandwidths reserved in the two directions for the various edges of the tree are shown adjacent to the edges in the figure. Thus, for instance, for edge $(5, 6)$, $C_T(5, 6) = 9$ (since $\sum_{l \in P_5^{(5,6)}} B_l^{out} = 12$ is greater than $\sum_{l \in P_6^{(5,6)}} B_l^{in} = 9$) and $C_T(6, 5) = 6$ (since $\sum_{l \in P_6^{(5,6)}} B_l^{out} = 12$ is greater than $\sum_{l \in P_5^{(5,6)}} B_l^{in} = 6$). Similarly, one can show that for edge $(6, 7)$, $C_T(6, 7) = 6$ (since $\sum_{l \in P_6^{(6,7)}} B_l^{out} = 16$ is greater than $\sum_{l \in P_7^{(6,7)}} B_l^{in} = 6$) and $C_T(7, 6) = 8$ (since $\sum_{l \in P_7^{(6,7)}} B_l^{out} = 8$ is less than $\sum_{l \in P_6^{(6,7)}} B_l^{in} = 9$). ∎

For the asymmetric bandwidths case, the problem of computing the VPN tree with the minimum cost can be shown to be at least as difficult as that of computing a Steiner tree connecting the VPN endpoints. Since the Steiner tree computation problem for a set of VPN endpoints is NP-hard [Hoc97, GJ79], it follows that the problem of computing the optimal VPN tree is also NP-hard.

Theorem 4.2: *For the asymmetric bandwidths case, the problem of computing the optimal VPN tree connecting the set of VPN endpoints in P is NP-hard.* ∎

4.1 Integer Programming Formulation

In this section, we show that the problem of computing the optimal tree can be formulated as an integer programming problem. For this, we first need to examine the properties of VPN trees connecting endpoints with asymmetric bandwidths.

For an edge (i, j) of VPN tree T, we say that it is *biased* towards i if the following two conditions hold:

1. ($\sum_{l\in P_i^{(i,j)}} B_l^{in} < \sum_{l\in P_j^{(i,j)}} B_l^{out}$) or ($\sum_{l\in P_i^{(i,j)}} B_l^{in} = \sum_{l\in P_j^{(i,j)}} B_l^{out}$ and $P_i^{(i,j)}$ contains a special node, say $\hat{l}$), and
2. ($\sum_{l\in P_i^{(i,j)}} B_l^{out} < \sum_{l\in P_j^{(i,j)}} B_l^{in}$) or ($\sum_{l\in P_i^{(i,j)}} B_l^{out} = \sum_{l\in P_j^{(i,j)}} B_l^{in}$ and $P_i^{(i,j)}$ contains a special node, say $\hat{l}$).

An edge (i, j) is said to be biased if it is biased towards either i or j. An edge that is not biased is said to be *balanced*. Also, we refer to a node of T as a *core* node if a balanced edge is incident on it.

Going back to Example 4.1, edge $(5, 6)$ is a balanced edge (since $\sum_{l\in P_5^{(5,6)}} B_l^{in} = 6$ is less than $\sum_{l\in P_6^{(5,6)}} B_l^{out} = 12$ and $\sum_{l\in P_6^{(5,6)}} B_l^{in} = 9$ is less than $\sum_{l\in P_5^{(5,6)}} B_l^{out} = 12$). Thus, nodes 5 and 6 are core nodes. However, edge $(6, 7)$ is biased towards 7 since $\sum_{l\in P_7^{(6,7)}} B_l^{in} = 6$ is less than $\sum_{l\in P_6^{(6,7)}} B_l^{out} = 16$ and $\sum_{l\in P_7^{(6,7)}} B_l^{out} = 8$ is less than $\sum_{l\in P_6^{(6,7)}} B_l^{in} = 9$.

In the following, we state certain properties of balanced edges. Let $M = \min\{\sum_{l\in P} B_l^{in}, \sum_{l\in P} B_l^{out}\}$.

Lemma 4.3: *The sum of the bandwidths reserved on a balanced edge (i, j) of a VPN tree T in both directions is M (that is, $C_T(i, j) + C_T(j, i) = M$).* ∎

Proof: For a balanced edge (i, j), it must be the case that (1) $\sum_{l\in P_i^{(i,j)}} B_l^{in} \ge \sum_{l\in P_j^{(i,j)}} B_l^{out}$ or $\sum_{l\in P_i^{(i,j)}} B_l^{out} \ge \sum_{l\in P_j^{(i,j)}} B_l^{in}$, and (2) $\sum_{l\in P_i^{(i,j)}} B_l^{in} \le \sum_{l\in P_j^{(i,j)}} B_l^{out}$ or $\sum_{l\in P_i^{(i,j)}} B_l^{out} \le \sum_{l\in P_j^{(i,j)}} B_l^{in}$. We consider the two cases involving $\sum_{l\in P_i^{(i,j)}} B_l^{in} \ge \sum_{l\in P_j^{(i,j)}} B_l^{out}$ (the first term in (1)). The other two cases for $\sum_{l\in P_i^{(i,j)}} B_l^{out} \ge \sum_{l\in P_j^{(i,j)}} B_l^{in}$ (the second term in (1)) can be handled in a similar fashion. Without loss of generality, we assume that $M = \sum_{l\in P} B_l^{in}$.

Case 1. In this case, $\sum_{l\in P_i^{(i,j)}} B_l^{in} = \sum_{l\in P_j^{(i,j)}} B_l^{out}$. Since $M = \sum_{l\in P} B_l^{in}$, it follows that $\sum_{l\in P_i^{(i,j)}} B_l^{out} \ge \sum_{l\in P_j^{(i,j)}} B_l^{in}$. Thus, it follows that $C_T(i, j) = \sum_{l\in P_j^{(i,j)}} B_l^{in}$ and $C_T(i, j) + C_T(j, i) = \sum_{l\in P} B_l^{in} = M$.

Case 2. In this case, $\sum_{l\in P_i^{(i,j)}} B_l^{in} \ge \sum_{l\in P_j^{(i,j)}} B_l^{out}$ and $\sum_{l\in P_i^{(i,j)}} B_l^{out} \le \sum_{l\in P_j^{(i,j)}} B_l^{in}$. Note that it must be the case that $\sum_{l\in P_i^{(i,j)}} B_l^{in} = \sum_{l\in P_j^{(i,j)}} B_l^{out}$ and $\sum_{l\in P_i^{(i,j)}} B_l^{out} = \sum_{l\in P_j^{(i,j)}} B_l^{in}$, since otherwise, $M = \sum_{l\in P} B_l^{in}$ is not possible. Thus, it follows that $C_T(i, j) + C_T(j, i) = \sum_{l\in P} B_l^{in} = M$. ∎

Revisiting Example 4.1, the total bandwidth reserved on balanced edge $(5, 6)$ is $M = \sum_{l\in P} B_l^{in} = 15$.

Lemma 4.4: *The restriction of T to only balanced edges forms a connected component.* ∎

Note that, from the above lemma, it follows that the core nodes are connected by a tree consisting entirely of balanced edges. Thus, if we delete the balanced edges from T, then in each of the resulting connected components, there is a single core node. We refer to the component containing core node v by C_v. Of course, if T contains no balanced edges, then there is only one component. Since all edges in the component are biased, one can show that there exists a unique node v in T such that every edge incident on v is biased away from it. This node v is then considered to be the core node for the component.

Lemma 4.5: *Consider the component C_v corresponding to core node v. Every edge (i, j) in C_v is biased towards j (assuming j is further from node v than i).* ∎

Lemma 4.6: *The cost of component C_v is $\sum_{l\in(C_v\cap P)} d_T(v, l) * (B_l^{in} + B_l^{out})$.* ∎

Thus, a VPN tree T consists of a set of core nodes connected by balanced edges, and connected components C_v for each core node containing only biased edges. Suppose that $core(T)$ denotes the set of core nodes in T and $bal(T)$ is the set of balanced edges in T. Then, from the above lemmas, we can infer that the cost of T, C_T, is equal to $M * |bal(T)| + \sum_{v \in core(T)} \sum_{l \in (C_v \cap P)} d_T(v,l) * (B_l^{in} + B_l^{out})$.

From the above discussion, it follows that each VPN tree, for the most part, can be completely characterized by its set of core nodes. Consider a set of nodes S. We define the cost of the set of nodes S, denoted by C_S, as $M * b + \sum_{l \in P} \min_{v \in S}\{d_G(v,l)\} * (B_l^{in} + B_l^{out})$. Here, b is the number of edges in the Steiner tree connecting the nodes in S. Next, we show how to construct the tree $T(S)$ for a set of nodes S such that $C_{T(S)} \leq C_S$. First, connect the nodes in S by a Steiner tree and add all the Steiner tree edges to $T(S)$. Next, coalesce all the nodes in S into one supernode, and construct a breadth first tree rooted at the supernode and connecting all the VPN endpoints in P (as the leaves). Add the edges of the breadth first tree to $T(S)$.

In the following, we show that the problem of computing the optimal VPN tree is equivalent to that of computing a set of nodes S for whom C_S is minimum. Specifically, we show that if T_{opt} is the optimal tree, then for the minimum C_S, $C_S \leq C_{T_{opt}}$. Thus, once we determine S, we can always construct the tree $T(S)$ whose cost is optimal due to the following lemma.

Lemma 4.7: *Let S be a set of nodes. Then $C_{T(S)} \leq C_S$.* ∎

The following lemma enables us to prove that computing the set of nodes S for which C_S is minimum yields the optimal VPN tree for the asymmetric case.

Lemma 4.8: *For a VPN tree T, $C_{core(T)} \leq C_T$.* ∎

Theorem 4.9: *Let S be a set of nodes for whom C_S is minimum. Then $T(S)$ is the optimal VPN tree.* ∎

Proof: Let T_{opt} be the optimal tree. We know that $C_S \leq C_{core(T_{opt})}$. Due to Lemma 4.7, $C_{T(S)} \leq C_S$, and due to Lemma 4.8, $C_{core(T_{opt})} \leq C_{T_{opt}}$. As a result, it follows that $C_{T(S)} \leq C_{T_{opt}}$. Thus, $T(S)$ is the VPN tree with the minimum cost. ∎

Thus, we have shown that, for the asymmetric case, to compute the optimal VPN tree, we simply need to compute a set of nodes S whose cost C_S is minimum; that is, a set S of nodes for whom the quantity $b * M + \sum_{l \in P} \min_{v \in S}\{d_G(v,l)\} * (B_l^{out} + B_l^{in})$ is minimum, where b is the number of edges in the Steiner tree connecting the nodes in S.

The problem of computing the set of nodes S with the minimum cost can be formulated as an integer program if we know the identity of one of the nodes in S. Let x_{ij}, y_i and z_e be 0-1 variables, where y_i is 1 if node i belongs to S, x_{ij} is 1 if VPN endpoint j is assigned to node i and z_e is 1 if edge e belongs to the Steiner tree connecting the nodes in S. Also, let $\delta(\hat{V})$ denote the set of edges crossing sets $\hat{V}$ and $V - \hat{V}$ in G and $B_j = B_j^{in} + B_j^{out}$. Suppose we know apriori that node $v \in S$. Then, the solution to the following integer program yields the optimal set of nodes S containing v.

$$\text{minimize} \sum_{i \in V, j \in P} d_G(i,j) * B_j * x_{ij} + M * \sum_{e \in E} z_e \tag{1}$$

subject to the following constraints

$$\forall j \in P: \ \sum_{i \in V} x_{ij} \geq 1$$

$$\forall i \in V, j \in P: \ y_i - x_{ij} \geq 0$$

$$\forall \hat{V} \subset V, v \notin \hat{V}, j \in P: \ \sum_{e \in \delta(\hat{V})} z_e - \sum_{i \in \hat{V}} x_{ij} \geq 0$$

$$x_{ij}, y_i, z_e \in \{0,1\}$$

The first two constraints state that each VPN endpoint j must be assigned to at least one node in S. The third constraint ensures that nodes in S are connected by a Steiner tree. It achieves this by requiring that if a VPN endpoint j is assigned to a node i (causing node i to belong to S), then i is connected to v by Steiner tree edges. The objective function that we minimize is the cost of the set of nodes S.

Thus, since we know that S must contain a node from V, we can compute the optimal tree by performing the following steps:

1. For each node $v \in V$, solve the integer program to compute S_v, the optimal set of nodes containing v (S_v consists of those nodes i for which $y_i = 1$ in the integer programming solution).

2. Return the tree $T(S_v)$ whose cost is minimum.

Note that while the minimum cost nodes computation problem has some similarities to the well-known *facility location problem* (FLP) [STA97], there is one significant difference. If we view the nodes in V as facilities, then in our case, the cost of each individual facility is 0. However, the chosen facilities as a whole have an associated cost since they need to be connected by a Steiner tree, the cost of each of whose edge is M. Thus, the cost of each individual facility is replaced by the cost of the Steiner tree connecting all the chosen facilities.

4.2 Rounding Based Approximation Algorithm

Solving integer program (1) to compute S_v is known to be computationally intractable [Hoc97]. In this subsection, we present an approximation algorithm that is based on solving the linear relaxation of the integer program, and then rounding the fractional solution to an integer solution that increases the cost of the fractional solution by a relatively small constant factor. The rounding algorithm consists of the following two phases: first we apply the filtering and rounding technique of Lin & Vitter [LV92] to obtain a new fractional solution, where the new solution has the property that whenever an endpoint j is fractionally assigned to a (partially chosen) node i, the distance $d_G(i,j)$ associated with that assignment is not too big. We then show how a fractional solution with this closeness property can be rounded to a near-optimal integer solution.

The relaxation of the integer program results in the following LP.

$$\text{minimize} \sum_{i \in V, j \in P} d_G(i,j) * B_j * x_{ij} + M * \sum_{e \in E} z_e \tag{2}$$

subject to the following constraints

$$\forall j \in P: \ \sum_{i \in V} x_{ij} \geq 1$$

$$\forall i \in V, j \in P: \ y_i - x_{ij} \geq 0$$

$$\forall \hat{V} \subset V, v \notin \hat{V}, j \in P: \ \sum_{e \in \delta(\hat{V})} z_e - \sum_{i \in \hat{V}} x_{ij} \geq 0$$

$$x_{ij}, y_i, z_e \geq 0$$

It can be shown that using the ellipsoid algorithm [Que93], we can solve the above LP in polynomial time. Let x, y, z be an optimal fractional solution to this LP. We show how to round this to an integer solution.

Let $0 < c < 1$ be a constant (we fix its value later). We use the filtering technique of Lin and Vitter from [LV92] (that was also employed in [STA97]) to derive a new fractional solution as follows. For each VPN endpoint j, we define a quantity α_j as follows: suppose π is a permutation of nodes such that $d_G(\pi(1),j) \leq d_G(\pi(2),j) \leq \cdots \leq d_G(\pi(n),j)$. Define $i^* = \min\{i' : \sum_{i=1}^{i'} x_{\pi(i)j} \geq c\}$. Now, define $\alpha_j = d_G(\pi(i^*),j)$. We will use the following later to prove our approximation result.

$$(1-c) * \alpha_j \ \leq \ \sum_{i \geq i^*} x_{\pi(i)j} * d_G(\pi(i),j) \ \leq \ \sum_{i \in V} d_G(i,j) * x_{ij} \tag{3}$$

```
procedure ComputeTreeRounding(F, α, G, P, v)
1. T_v := ∅
2. activeSet:= P
3. seedSet:= ∅
4. for each VPN endpoint l in P
5.    let F'_l be the set of nodes i for which d_G(i,l) ≤ c'α_l
6. while activeSet ≠ ∅ {
7.    let j be a VPN endpoint in activeSet with the minimum value of α_j
8.    seedSet := seedSet ∪{j}
9.    S_j := {j}
10.   for each endpoint l in activeSet {
11.      if F'_j ∩ F_l ≠ ∅ or F_j ∩ F'_l ≠ ∅ {
12.         delete l from activeSet
13.         add l to S_j
14.      }
15.   }
16. }
17. let G' be the graph obtained from G as a result of coalescing all nodes in F_j into a supernode for each j in seedSet
18. construct a Steiner tree T connecting the supernodes in G' /* v, if not coalesced, is considered a supernode */
19. add edges in T to T_v
20. let T_dir denote the tree T with edges in T directed to form an outgoing arborescence from v in G'
21. for each j in seedSet {
22.    let u ∈ F_j denote the node with an incoming arc in T_dir
23.    for each w ∈ F_j with an outgoing arc in T_dir
24.       add edges in the shortest path from u to w in G, to T_v
25. }
26. delete edges from T_v until it contains no loops
27. return T_v
```

Figure 5: Algorithm for Computing VPN Tree from Fractional LP Solution

We define a new feasible fractional solution $(\bar{x}, \bar{y}, \bar{z})$ for the LP as follows: for each endpoint j, define $c_j = \sum_{i:d_G(i,j)\le\alpha_j} x_{ij}$. Note that $c_j \ge c$. Define $\bar{x}_{ij} = x_{ij}/c_j$, if $d_G(i,j) \le \alpha_j$, and 0 otherwise. For each i, define $\bar{y}_i = \min\{1, y_i/c\}$. Finally, for each edge e, define $\bar{z}_e = \min\{1, z_e/c\}$. It is easy to verify that this new solution is feasible to the LP. We next show how to round the new fractional solution to a near-optimal integer solution such that each endpoint j is assigned to some node i such that $\bar{x}_{ij} > 0$. We denote by $F_j = \{i : \bar{x}_{ij} > 0\}$ the set of eligible nodes for endpoint j.

Procedure ComputeTreeRounding shown in Figure 5 computes a feasible integer solution $(\hat{x}, \hat{y}, \hat{z})$ that is within a constant factor of the fractional solution $(\bar{x}, \bar{y}, \bar{z})$. The procedure begins by clustering the VPN endpoints in Steps 6–16. Each cluster has a VPN endpoint j that is the *seed* of the cluster. The set of seed endpoints are stored in seedSet and S_j is the cluster with endpoint j as the seed. Every endpoint $l \in S_j$ has the following properties: (1) $\alpha_j \le \alpha_l$, (2) for some node $i \in F_l$, $d_G(i,j) \le c'\alpha_j$ or for some node $i \in F_j$, $d_G(i,l) \le c'\alpha_l$ (here $c' > 1$ is a constant that we define later). We will assign each endpoint in S_j to some node in F_j, and the following lemma states that this will not increase the overall cost by much.

Lemma 4.10: *For each $l \in S_j$ and $i \in F_j$, $d_G(i,l) \le (c'+2) * \alpha_l$.* ∎

Proof: Consider an $i \in F_j$. Due to the definition of F_j, it follows that $d_G(i,j) \le \alpha_j$. We consider the following two cases, one of which must hold because $l \in S_j$.

1. For some node $i' \in F_l$, $d_G(i', j) \leq c'\alpha_j$. Since $i' \in F_l$, it must be the case that $d_G(i', l) \leq \alpha_l$. Thus, due to the triangle inequality, $d_G(i, l) \leq d_G(i, j) + d_G(i', j) + d_G(i', l) = (c' + 1)\alpha_j + \alpha_l$. Since $\alpha_j \leq \alpha_l$, the lemma follows.

2. For some node $i' \in F_j$, $d_G(i', l) \leq c'\alpha_l$. Since $i' \in F_j$, it must be the case that $d_G(i', j) \leq \alpha_j$. Thus, due to the triangle inequality, $d_G(i, l) \leq d_G(i, j) + d_G(i', j) + d_G(i', l) = c'\alpha_l + 2 * \alpha_j$. Since $\alpha_j \leq \alpha_l$, the lemma holds.

∎

In order to maintain feasibility of the solution once endpoints in S_j are assigned to some node in F_j, we need to construct a Steiner tree T that connects v to at least one node from F_j for each j belonging to seedSet. To accomplish this, in the graph G, for each endpoint j in seedSet, we *contract* the nodes in each F_j to a new supernode. We then connect the supernodes by a Steiner tree T, thus ensuring that each supernode is connected to v (Step 18). However, note that although T connects the supernodes (and v) in G', it may happen that T does not form a single connected subgraph in G. The reason for this is that edges of T may be incident on different nodes in an F_j. Thus, in order to ensure that T forms a connected subgraph even in G, in Steps 22–24, we select a node u in F_j and connect it to every other node of F_j on which an edge of T is incident.

In the following lemma, we show that the number of edges in T_v is within a constant factor of $\sum_{e \in E} \bar{z}_e$.

Lemma 4.11: *The number of edges in T_v is less than or equal to $\frac{2(c'+1)}{c'-1} * \sum_{e \in E} \bar{z}_e$.* ∎

Proof: We first show that the number of edges in the Steiner tree T that connects the supernodes in $G' = (V', E')$ is at most $2\sum_e \bar{z}_e$. Consider any set $S \subset V'$ containing a proper subset of the set of supernodes. Without loss of generality, we assume that S does not contain v (if S contains v, then we replace S by $V' - S$). Let S contain the supernode corresponding to endpoint j (resulting due to collapsing nodes in F_j). Since $(\bar{x}, \bar{y}, \bar{z})$ is a feasible solution for LP (2), $\sum_{e \in \delta(S)} \bar{z}_e \geq \sum_{i \in S} \bar{x}_{ij} \geq \sum_{i \in F_j} \bar{x}_{ij} = 1$. So, if we consider an instance of the Steiner tree problem with the set of supernodes as the set of nodes to be connected in G', then $\bar{z}_e$ is a feasible fractional solution to this problem. Thus, it is possible to construct a Steiner tree T connecting the supernodes containing at most $2\sum_e \bar{z}_e$ edges [Hoc97].

We next show that for every j in seedSet, connecting node $u \in F_j$ to every other node w in F_j with an outgoing arc in T_{dir} (in Steps 22-24) increases the number of edges in T by a factor of at most $\frac{c'+1}{c'-1}$. First, observe that the length of the shortest path between u and w is at most $2 * \alpha_j$ (since endpoint j is at a distance of at most α_j from both u and w). Also, in T, there must be a path from w to a node i belonging to F_l for some other $l \neq j$ in seedSet. Furthermore, $d_G(i, j) > c'\alpha_j$, since otherwise j and l would belong to the same cluster. Thus, the length of the path from w to i is at least $(c' - 1)\alpha_j$. We charge the cost of $2\alpha_j$ of connecting u to w to this segment of T – it is easy to show that disjoint segments of T will be charged in this manner. So, the number of edges in T increases by a factor of at most $(c' + 1)/(c' - 1)$. This proves the lemma. ∎

We are now in a position to show the near-optimality of the final rounded integer solution $(\hat{x}, \hat{y}, \hat{z})$. In this solution, in addition to setting $\hat{y}_v = 1$, for every j in seedSet, for node $u \in F_j$ that has an incoming arc in T_{dir}, $\hat{y}_u$ is set to 1, otherwise $\hat{y}_u$ is set to 0. Further, for every endpoint $l \in S_j$, l is assigned to node $u \in F_j$ with the incoming arc, that is, $\hat{x}_{ul} = 1$. Finally, for every edge e in T_v, $\hat{z}_e = 1$ and $\hat{z}_e = 0$ for all other edges. The integer solution is clearly feasible since T_v connects every $u \in F_j$ (that has an incoming arc in T_{dir}) to v.

Theorem 4.12: *The cost of integer solution $(\hat{x}, \hat{y}, \hat{z})$ is within a factor of 10 of the cost of the optimal LP solution (x, y, z).* ∎

Proof: The cost of the integer solution $(\hat{x}, \hat{y}, \hat{z})$ is given by $\sum_{i \in V, j \in P} d_G(i, j) * B_j * \hat{x}_{ij} + M * \sum_{e \in E} \hat{z}_e$. Due to Lemma 4.10, $\sum_{i \in V, j \in P} d_G(i, j) * B_j * \hat{x}_{ij} \leq (c' + 2) \sum_{j \in P} B_j * \alpha_j$. Also, due to Lemma 4.11, $M * \sum_{e \in E} \hat{z}_e \leq M * \frac{2(c'+1)}{c'-1} \sum_{e \in E} \bar{z}_e$. Combining this with Equation (3) and since $\bar{z}_e \leq z_e/c$, we get that the cost of $(\hat{x}, \hat{y}, \hat{z})$ is at most $\frac{c'+2}{1-c} \sum_{i \in V, j \in P} d_G(i,j) * B_j * x_{ij} + \frac{2(c'+1)}{c(c'-1)} M * \sum_{e \in E} z_e$. Thus, $(\hat{x}, \hat{y}, \hat{z})$ is within a constant factor of the optimal fractional solution (to the LP). This constant turns out to be $\max\{\frac{c'+2}{1-c}, \frac{2(c'+1)}{c(c'-1)}\}$. Choosing $c = \frac{3}{5}$ and $c' = 2$, we get a value of 10 for the constant. ∎

Time Complexity. The time complexity of Procedure COMPUTETREEROUNDING can be shown to be $O(n^2(\log n + p))$, where $p = |P|$ and $n = |V|$. The first term $n^2 \log n$ is the time complexity of constructing a Steiner tree in Step 18 [Hoc97], while the second term $n^2 p$ is due to the overhead of computing shortest paths for at most p (u, w) node pairs in Steps 21–25.

4.3 Primal-Dual Algorithm

While the rounding based algorithm gives a constant factor performance guarantee on the cost of the computed VPN tree (with respect to the cost of the optimal tree), it requires solving the LP relaxation of the integer program. This LP relaxation has a small number of variables, but an exponential number of constraints. Even though the ellipsoid method can be used to solve the LP in polynomial time [Que93], it may not be computationally efficient and thus impractical.

In this section, we propose an algorithm that employs the primal-dual method in order to find a feasible solution to the integer program (1). The dual for the LP relaxation (2) is as follows.

$$\text{maximize} \sum_{j \in P} \alpha_j \tag{4}$$

$$\forall i \in V, j \in P: \quad \alpha_j - \sum_{\hat{V} \subset V, i \in \hat{V}, v \notin \hat{V}} \gamma_{\hat{V}j} \leq B_j * d_G(i,j)$$

$$\forall e \in E: \quad \sum_{\hat{V} \subset V, e \in \delta(\hat{V}), v \notin \hat{V}} \gamma_{\hat{V}j} \leq M$$

$$\forall j \in P: \quad \alpha_j \geq 0$$

$$\forall j \in P, V \subset \hat{V}, v \notin \hat{V}: \quad \gamma_{\hat{V}j} \geq 0$$

The dual is employed to guide in the selection of the set F_j of potential nodes for each VPN endpoint j. The primal complementary slackness conditions imply the following:

- If endpoint j is assigned to node $i \in V$, then $\alpha_j - \sum_{i \in \hat{V}, v \notin \hat{V}} \gamma_{\hat{V}j} = B_j * d_G(i,j)$.
- If edge e is a Steiner tree edge, then $\sum_{e \in \delta(\hat{V}), v \notin \hat{V}} \gamma_{\hat{V}j} = M$.

Thus, F_j consists of all nodes i for which $\alpha_j - \sum_{i \in \hat{V}, v \notin \hat{V}} \gamma_{\hat{V}j} = B_j * d_G(i,j)$, while edges e for which $\sum_{e \in \delta(\hat{V}), v \notin \hat{V}} \gamma_{\hat{V}j} = M$ constitute potential Steiner tree edges. Once F_j for each endpoint j has been computed, Procedure COMPUTETREEROUNDING (see Figure 5) is used to compute the final set of nodes S to which VPN endpoints are to be assigned, and the Steiner tree connecting the nodes.

The overall procedure for computing the optimal VPN tree is shown in Figure 6. Procedure COMPUTETREEPRIMALDUAL uses the primal-dual method to compute a set of nodes S containing a specific node $v \in V$ and the Steiner tree edges connecting the nodes in S. This is done in the body of the for loop spanning Steps 2–38 for each node $v \in V$ and the tree with the smallest cost is returned. The primal-dual algorithm is an iterative algorithm – during each iteration, α_j for the VPN endpoint j with the smallest α_j is increased and in order to preserve dual feasibility, the $\gamma_{\hat{V}j}$s are appropriately adjusted. When α_j for a VPN endpoint j becomes equal to $d_G(i,j) * B_j$ for a node $i \in V$, node i becomes a potential node for assigning endpoint j and is added to F_j. Further, when for an edge e, the sum of $\gamma_{\hat{V}j}$s (for which $e \in \delta(\hat{V})$) becomes equal to M, the edge is added to the set of potential Steiner tree edges connecting the nodes that are chosen for VPN endpoints. The α_j for a VPN endpoint j is not increased once one of the potential nodes for it (in F_j) becomes connected to v via potential Steiner tree edges.

This is because, as explained below, when α_j is increased, dual feasibility cannot be maintained by increasing $\gamma_{\hat{V}j}$, where $v \notin \hat{V}$.

Data Structures. The algorithm collects the *potential* nodes for a VPN endpoint j in F_j. These are the nodes i for which $\alpha_j \geq B_j * d_G(i,j)$. Also, for each edge e, w_e stores the sum of all the $\gamma_{\hat{V}j}$s that contribute to e – here $\hat{V}$ does not contain v and $e \in \delta(\hat{V})$. Thus, when $w_e = M$, e becomes a potential Steiner tree edge. Also note that for any potential Steiner tree edge e, if $e \in \delta(\hat{V})$, then $\gamma_{\hat{V}j}$ cannot be increased since this would result in a violation of dual feasibility. In the procedure, C_u is used to store the nodes connected to u via potential Steiner tree edges. Finally, S_j is used to store the set of all nodes connected to nodes in F_j via potential Steiner tree edges (thus $F_j \subseteq S_j$). As mentioned earlier, once S_j contains v, then α_j for endpoint j cannot be increased any further.

Algorithm. The complete primal-dual algorithm for computing the VPN tree with low cost is illustrated in Figure 6. For each $v \in V$ (in the outermost for loop), the algorithm first employs the primal-dual method to compute a set of potential nodes F_j for each VPN endpoint j and a set of potential Steiner tree edges that connect each F_j to v (Steps 3–30). It then invokes Procedure COMPUTETREEROUNDING to compute the Steiner tree containing v and connecting the nodes to which VPN endpoints are assigned. This tree is then extended to connect the VPN endpoints in P in Steps 32–35.

The variable activeSet stores the VPN endpoints j for which α_j can still be incremented. The set S_j denotes the smallest set $\hat{V}$ of nodes for which $\gamma_{\hat{V}j}$ needs to be increased when α_j for a VPN endpoint j is increased. The reason for this is that for every $i \in F_j$, $\alpha_j \geq B_j * d_G(i,j)$. Thus, in order to ensure that the dual equation $\alpha_j - \sum_{i \in \hat{V}, v \notin \hat{V}} \gamma_{\hat{V}j} \leq B_j * d_G(i,j)$ stays feasible when α_j is incremented, $\gamma_{\hat{V}j}$ where $i \in \hat{V}$ must also be incremented. Note also that $\gamma_{\hat{V}j}$ can be incremented only if $\delta(\hat{V})$ contains no potential Steiner tree edges. This is because for a potential Steiner tree edge e, $\sum_{e \in \delta(\hat{V}), v \notin \hat{V}} \gamma_{\hat{V}j} = M$. As a result, increasing $\gamma_{\hat{V}j}$ if $e \in \delta(\hat{V})$ could cause the dual equation $\sum_{e \in \delta(\hat{V}), v \notin \hat{V}} \gamma_{\hat{V}j} \leq M$ to be violated. Thus, if $\gamma_{\hat{V}j}$ is the variable that is increased to maintain dual feasibility when α_j for a VPN endpoint j is increased, then $\hat{V}$ must contain all the nodes in C_i for every $i \in F_j$, or alternately $S_j \subseteq \hat{V}$.

From the above discussion, it follows that if S_j for a VPN endpoint j contains v, then α_j cannot be increased any further. This is because there are no $\gamma_{\hat{V}j}$ variables for sets $\hat{V}$ that contain node v. As a result, it is not possible to increase $\gamma_{\hat{V}j}$ to ensure that the dual equation $\alpha_j - \sum_{i \in \hat{V}, v \notin \hat{V}} \gamma_{\hat{V}j} \leq B_j * d_G(i,j)$ stays feasible when α_j is incremented. Thus, activeSet only contains VPN endpoints j for whom S_j does not contain v.

In each iteration of the while loop spanning Steps 7–30, α_j for a single VPN endpoint j belonging to activeSet is incremented by $\min\{\epsilon_1, \epsilon_2\}$, where j, ϵ_1 and ϵ_2 are as defined in Steps 8–12. Note that increasing α_j causes one of the following to happen – (1) k to be added to F_j since $\alpha_j = d_G(j,k)$ (if $\epsilon_1 \leq \epsilon_2$), or (2) edge e to become a potential Steiner tree edge since $M = w_e$ (if $\epsilon_2 < \epsilon_1$). For the latter case, the connected components for nodes connected to u and v need to be adjusted (Steps 23 and 24). In addition, S_js for VPN endpoints j that contain either u or v need to be expanded as described in Steps 26 and 27.

Finally, note that in order to maintain feasibility of equations $\alpha_j - \sum_{i \in \hat{V}, v \notin \hat{V}} \gamma_{\hat{V}j} \leq B_j * d_G(i,j)$ for endpoints i in F_j, $\gamma_{S_j j}$ is increased by ϵ_1/ϵ_2 when α_j is increased by ϵ_1/ϵ_2. This, in turn, contributes ϵ_1/ϵ_2 to $w_{e'}$ for all edges $e' \in \delta(S_j)$ (Steps 15 and 22).

Time Complexity. The time complexity of Procedure COMPUTETREEPRIMALDUAL can be shown to be $O(n(m^2p + mnp + n^2 \log n))$, where $p = |P|$, $m = |E|$ and $n = |V|$. The outermost for loop performs n iterations, one for each node $v \in V$. Further, for each iteration of the outermost loop, the body of the if condition (Steps 14–18) can be executed at most np times, once for each VPN endpoint node pair, while the body of the else condition (Steps 21–28) can be executed at most m times, once for each edge in E. Assuming that unions involving S_j can be performed in $O(n)$ steps and lookups of S_j can be carried out in constant time, the complexity of Steps 14-18 can be shown to be $O(m+n)$, while the complexity of Steps 21–28 can be shown to be $O(mp+np)$. Finally,

```
procedure COMPUTETREEPRIMALDUAL($G$, $P$)
$T := \emptyset$
for each $v \in V$ {
    for each $i \in V$, $C_i := \{i\}$
    for each $e \in E$, $w_e := 0$
    for each $j \in P$, $F_j := \{j\}$, $S_j := \{j\}$, $\alpha_j := 0$
    activeSet := $P$
    while activeSet $\neq \emptyset$ {
        select a VPN endpoint $j$ with minimum $\alpha_j/B_j$ from activeSet
        let $k$ be the node in $V - F_j$ such that $d_G(k, j)$ is minimum
        let $e = (u, w) \in \delta(S_j)$ such that $M - w_e$ is minimum
        let $\epsilon_1 := d_G(k, j) * B_j - \alpha_j$
        let $\epsilon_2 := M - w_e$
        if $\epsilon_1 \leq \epsilon_2$ {
            $\alpha_j := \alpha_j + \epsilon_1$
            for each $e' \in \delta(S_j)$, $w_{e'} := w_{e'} + \epsilon_1$
            $F_j := F_j \cup \{k\}$
            $S_j := S_j \cup C_k$
            if $v \in S_j$, delete $j$ from activeSet
        }
        else{
            $\alpha_j := \alpha_j + \epsilon_2$
            for each $e' \in \delta(S_j)$, $w_{e'} := w_{e'} + \epsilon_2$
            for each $l \in C_u$, $C_l := C_l \cup C_w$
            for each $l \in C_w$, $C_l := C_l \cup C_u$
            for each $l \in P$ {
                if $w \in S_l$, $S_l := S_l \cup C_u$
                if $u \in S_l$, $S_l := S_l \cup C_w$
                if $v \in S_l$, delete $l$ from activeSet
            }
        }
    }
    $T_v$ := computeTreeRounding($F, \alpha/B, G, P, v$)
    let $G'$ be the graph obtained from $G$ as a result of coalescing all nodes in $T_v$ into a supernode $v'$
    construct breadth first tree $T'$ rooted at $v'$ and whose leaves are the VPN endpoints in $P$
    add edges in $T'$ to $T_v$
    delete leaves from $T_v$ that do not correspond to VPN endpoints
    if $C_{T_v} < C_T$, $T := T_v$
}
return $T$
```

Figure 6: Primal-Dual Algorithm for Computing VPN Tree

as shown earlier, the time complexity of Procedure COMPUTETREEROUNDING is $O(n^2 \log n + n^2 p)$. Thus, the Procedure COMPUTETREEPRIMALDUAL has an overall time complexity of $O(n(m^2 p + mnp + n^2 \log n))$.

4.4 Breadth First Search Based Algorithm

The breadth first search algorithm presented in Section 3 can also be used to compute the VPN tree for the asymmetric bandwidth case (see Procedure COMPUTETREESYMMETRIC in Figure 3). However, since the VPN tree computation problem for the asymmetric case is NP-hard, the algorithm may not return the optimal VPN tree. Nevertheless, one can show that cost of the tree computed by the procedure is within a factor of $\frac{\sum_{l \in P} B_l^{in} + B_l^{out}}{M}$ of the cost of the optimal VPN tree.

Theorem 4.13: *The cost of the tree returned by Procedure* COMPUTETREESYMMETRIC *is within a factor of* $\frac{\sum_{l \in P} B_l^{in} + B_l^{out}}{M}$ *of the cost of the optimal VPN tree.* ∎

Proof: Let $B_j = B_j^{in} + B_j^{out}$ for VPN endpoint j. Consider the optimal tree T_{opt} and let $b = |bal(T_{opt})|$ be the number of balanced edges in T_{opt}. Also let $core_j$ denote the core node closest to VPN endpoint j in T_{opt}. Thus, the cost of T_{opt} is at least $\sum_{j \in P} d_{T_{opt}}(core_j, j) * B_j + b * M$.

Now consider the breadth first tree T_v rooted at v, a core node, and computed by Procedure COMPUTETREESYMMETRIC. The quantity $\sum_{j \in P} d_{T_v}(v, j) * B_j$ is an upper bound on the cost of the tree T_v (since the bandwidth reserved on an edge (u, w) (w is further from v than u in T_v) is at most $\sum_{l \in P_w^{(u,w)}} B_l$, each VPN endpoint j can contribute at most B_j to the reserved bandwidth on edges along the path from v to j). Now $d_{T_v}(v, j) \leq b + d_{T_{opt}}(core_j, j)$. As a result, the cost of T_v is no more than $\sum_{j \in P} d_{T_{opt}}(core_j, j) * B_j + \sum_{j \in P} b * B_j$. Thus, the ratio of the costs of trees T_v and T_{opt} is less than or equal to $\frac{b * \sum_{j \in P} B_j}{b * M}$, which proves the lemma. ∎

5 Incorporating Link Capacity Constraints

The algorithms for computing VPN trees in the previous two sections do not take into account link capacity constraints, that is, the algorithms assume that each link has a large available bandwidth. In this section, we consider the problem of provisioning VPN trees when edges of the graph G have associated capacity constraints. Recall that R_{ij} denotes the residual capacity of edge (i, j) - thus, R_{ij} is the available bandwidth on link (i, j). We say that a VPN tree T is *feasible* if for each edge (i, j) in T, $C_T(i, j) \leq R_{ij}$. In this section, we develop algorithms for computing a feasible VPN tree T with the minimum possible cost.

In the presence of edge capacity constraints, the problem of computing the optimal VPN tree is NP-hard even when endpoints have equal ingress and egress bandwidths. Further, one can show that unless $P = NP$, it is impossible to approximate the optimal VPN tree to within a constant factor in polynomial time.

Theorem 5.1: *In the presence of edge capacity constraints, the problem of determining if there exists a feasible VPN tree T is NP-hard.* ∎

Corollary 5.2: *In the presence of edge capacity constraints, the problem of determining if there exists a feasible VPN tree whose cost is within a constant factor c of the optimal cost is NP-hard.*

5.1 Symmetric Ingress and Egress Bandwidths

In Section 3, we showed that in the absence of link capacity constraints and when VPN endpoints had symmetric ingress and egress bandwidth requirements, considering the breadth first tree rooted at each node $v \in V$, yielded the optimal VPN tree (see Procedure COMPUTETREESYMMETRIC in Figure 3). In this subsection, we show how the procedure can be extended to handle link bandwidth constraints.

procedure COMPUTETREESYMMETRICCONST(G, P, v)

$T_v := \emptyset$
$S := P$
make copy G_v of G (including residual capacities of edges in G)
while ($S \neq \emptyset$) {
    $T := v$
    openQ := $\{v\}$
    while (openQ $\neq \emptyset$){
        dequeue first node u from openQ
        for each edge (u, w) in G_v such that w is not in T {
            if w is not in T_v or edge (u, w) is in T_v
                add edge (u, w) to T
                append node w to end of openQ
        }
    }
    prune leaves from T that do not correspond to remaining VPN endpoints (that is, do not belong to S)
    if T does not contain all the VPN endpoints in S
        return $\emptyset$ /* tree rooted at v cannot satisfy capacity constraints */
    compute maximum bandwidth $M_{T,v}(i, j)$ to be reserved on each edge (i, j) of the tree T rooted at v
    violatedSet := $\emptyset$
    while there exists an edge (i, j) in T such that $M_{T,v}(i, j) > R_{ij}$ (in G_v){
        let (i, j) be the violated edge in T furthest from node v (also, let node j be further from v than node i)
        let (u, w) be the most recently added edge to T in $T_j^{(i,j)}$ and such that $M_{T,v}(u, w) > (M_{T,v}(i, j) - R_{ij})$
                (w is further from v than u)
        delete from T edge (u, w) and all nodes and edges in $T_w^{(u,w)}$
        prune leaves from tree T that do not correspond to remaining VPN endpoints (that is, do not belong to S)
        recompute bandwidths $M_{T,v}(x, y)$ to be reserved on each edge (x, y) of the tree T
        add edge (i, j) to violatedSet
    }
    add edges of T to T_v
    for each edge (i, j) in tree T {
        set residual capacity R_{ij} of edge (i, j) in G_v to be $R_{ij} - M_{T,v}(i, j)$
        if edge $(i, j) \in$ violatedSet
            delete nodes in $T_j^{(i,j)}$ (and edges incident on them) from G_v
    }
    delete all VPN endpoints in T from S ($S := S - T$)
}
return T_v

Figure 7: Algorithm for Computing VPN Tree in the presence of Edge Capacity Constraints

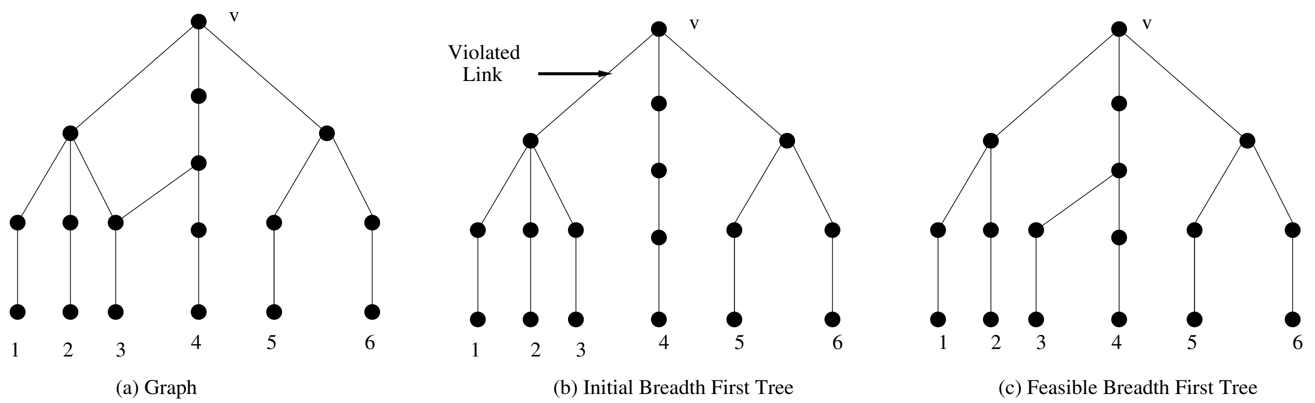


Figure 8: Example of Feasible Tree When Links Have Bandwidth Constraints

Figure 7 depicts the procedure for computing a feasible VPN tree for the case of symmetric endpoint bandwidths. The procedure has polynomial time complexity and as a result, due to Theorem 5.1, may not compute the VPN tree with the optimal cost. It computes a VPN tree T_v with low cost and rooted at node v. Thus, one option is to invoke the procedure for each $v \in V$ and then choose as the VPN tree, the T_v with the smallest cost. A different option is to invoke the procedure for only a small subset of nodes in V – this subset can be the k vertices that result in the k smallest cost trees when link capacity constraints are not considered (Procedure COMPUTETREESYMMETRIC in Figure 3 is used to compute the minimum cost tree rooted at a vertex v).

Before describing the steps of Procedure COMPUTETREESYMMETRICCONST in more detail, we provide a brief overview of the key intuition underlying it. The spirit of the algorithm is similar to the optimal algorithm for the symmetric bandwidth case when we did not take into account link constraints. Similar to Procedure COMPUTETREESYMMETRIC (see Figure 3), the algorithm attempts to construct a breadth first tree rooted at vertex v that is also feasible. If the breadth first tree rooted at v is already feasible, then we are done. However, the problem arises when the initial breadth first tree T_v rooted at v does not satisfy the link capacity constraints. Then, for each violated link (i, j) in T_v, the bandwidth reserved on the edge can be reduced by deleting some VPN endpoints from T_v. The algorithm thus deletes from the subtree of T_v rooted at node j, the minimum number of VPN endpoints so that the bandwidth reserved on link (i, j) is less than the link's residual capacity. Thus, link (i, j) of T_v is no longer violated. Repeating this process for every violated link thus results in a feasible tree T_v – however, T_v may now no longer connect all the VPN endpoints in P. In order to connect the remaining VPN endpoints (that were previously deleted to ensure feasibility), we grow T_v once again in a breadth first fashion. But, this time we do not expand nodes in the subtree of T_v rooted at node j for a previously violated edge (i, j) – the reason for this is that adding new VPN endpoints in this subtree could cause link (i, j) to be violated again. After T_v is expanded to connect all the remaining VPN endpoints, VPN endpoints are again deleted from T_v until it contains no violated links, and the process of growing T_v is repeated.

Example 5.3: Consider the network graph shown in Figure 8(a). Suppose the residual bandwidth of each link of the graph is 2 units, and we are interested in provisioning a VPN tree for the set of VPN endpoints P containing nodes numbered from 1 to 6. Also, suppose that each endpoint has a bandwidth requirement of 1 unit. The breadth first tree T_v rooted at node v is shown in Figure 8(b). Tree T_v, however, is not feasible since the bandwidth to be reserved on the link connecting node v to the subtree containing nodes 1, 2 and 3, is 3 units, which exceeds the available capacity of the link, which is 2 units. (The violated link is marked in Figure 8(b)).

Thus, in order to make tree T_v feasible, VPN endpoint 3 is chosen as the victim and deleted from T_v. Tree T_v is then expanded in a breadth first fashion to also include VPN endpoint 3. However, the portion of T_v rooted at the violated link (subtree containing endpoints 1 and 2) is not expanded. Instead, endpoint 3 is added to T_v along a different link as shown in Figure 8(c) resulting in a final tree that is feasible. ∎

We now describe the steps of Procedure COMPUTETREESYMMETRICCONST in more detail. Inputs to the procedure include the graph G with residual link bandwidths and the set of VPN endpoints P. The procedure returns a feasible VPN tree rooted at v if it can find one – otherwise, it simply returns $\emptyset$. Variable T_v in the procedure, is used to store the feasible VPN tree rooted at v constructed so far, and S is used to store the set of VPN endpoints that still remain to be connected to T_v. In each iteration of the for loop spanning Steps 4–35, T_v is expanded to connect the remaining VPN endpoints in S. This is carried out in Steps 5–14, where a new breadth first tree T rooted at v and whose leaves are endpoints in S, is constructed. The tree T is constructed in a manner that ensures that the restrictions of the two trees T and T_v are identical with respect to common nodes. Thus T, in some respect, corresponds to an expansion of T_v. Note that in Steps 16–17, if T does not contain all the VPN endpoints in S, then it is not possible to expand T_v to connect the remaining VPN endpoints and so a feasible tree rooted at node v cannot be found.

Once the tree T connecting VPN endpoints in S has been computed, we need to delete endpoints from it if the bandwidth to be reserved on some link (i, j) of T exceeds the residual bandwidth R_{ij} of the link. This is achieved in the while loop spanning Steps 20–27. $M_{T,v}(i, j)$ is the maximum bandwidth to be reserved on each link (i, j) of T and is simply equal to $\sum_{l\in(T_j^{(i,j)}\cap P)} B_l$ (the sum of bandwidth requirements of VPN endpoints in $T_j^{(i,j)}$). Thus, an edge (i, j) is violated if $M_{T,v}(i, j) > R_{ij}$. Now, in order to make edge (i, j) feasible, we need to delete VPN endpoints from $T_j^{(i,j)}$, the sum of whose bandwidth requirements is at least the amount of the violation $M_{T,v}(i, j) - R_{ij}$. Also, it would be preferable to choose as the "victims" VPN endpoints that are most distant from v since these require bandwidth to be reserved along more edges of T. Thus, in Steps 23-24, we identify the subtree $T_w^{(u,w)}$ for edge (u, w) in T that is most distant from v and whose deletion from T would cause link (i, j) to be no longer violated. This process of deleting VPN endpoints is carried out until T contains no violated links and the set of violated links is kept track of in the variable violatedSet.

After the while loop spanning Steps 20–27 completes execution, T is a feasible tree connecting some subset of VPN endpoints in S. Thus, since T is consistent with T_v on portions they have in common, edges of T can simply be added to T_v while preserving the property that T_v is a tree. Also, the bandwidth for each link (i, j) in T is reserved by adjusting the residual bandwidth for the link in G_v by $M_{T,v}(i, j)$ (as described in Step 30). Reserving bandwidth for tree T independently during each iteration does not pose a problem since it is relatively straightforward to show that the bandwidth that must be reserved for the union of the trees (generated across the various iterations) is equal to the sum of the bandwidths reserved for each individual tree. More specifically, if $T = T'$ in one iteration and $T = T''$ in another, then for any link (i, j) in $T' \cap T''$, $M_{T'\cup T'',v}(i, j) = M_{T',v}(i, j) + M_{T'',v}(i, j)$. Finally, since we don't want to further expand portions of T_v rooted at violated links (since there is a high probability that these will be violated again), we delete these portions of T_v from G_v (Steps 31–32). This ensures that these subtrees rooted at violated edges will not be part of T when T_v is expanded in the next iteration, and thus no new VPN endpoints will ever be added to these subtrees.

5.2 Asymmetric Ingress and Egress Bandwidths

The procedure for computing the VPN tree when VPN endpoint bandwidths are asymmetric and in the presence of link bandwidth constraints is similar to Procedure COMPUTETREEPRIMALDUAL (see Figure 6), except for the following key differences.

1. As shown earlier in Lemma 4.3 (see Section 4), the balanced edges that connect the core nodes of a VPN tree have the property that the sum of the bandwidths reserved on each edge in the two directions is M. Suppose E_M denotes the set of links in E for whom the residual capacity in each direction is at least M. Then, by requiring that the potential steiner tree edges (that connect each F_j to node v) belong to E_M, we can ensure that there is sufficient capacity on each link to accommodate the M units of bandwidth distributed over the two directions. Thus, in Step 10, potential steiner tree edges are chosen from E_M.

2. In Step 32, in the invocation of Procedure COMPUTETREEROUNDING, E_M is passed as an input parameter instead of E. This ensures that only edges in E_M are used to construct the steiner tree T_v returned by the

procedure.

3. Since links have capacity constraints, instead of constructing a simple breadth first tree rooted at v' in Step 34, Procedure COMPUTETREESYMMETRICCONST is used to compute a feasible breadth first tree rooted at v'.

6 Experimental Study

We conducted an extensive empirical study to measure the performance of our *breadth first search* (BFS) and *primal-dual* algorithms, and compared them with the approach of using a Steiner tree to connect VPN endpoints [DGG+98]. The major findings of our study can be summarized as follows:

- The primal-dual algorithm generates VPN trees with the smallest cost for a wide range of ingress/egress bandwidth ratios. It outperforms both the BFS and the Steiner tree algorithms for medium to large bandwidth ratios.

- For low ingress/egress bandwidth ratios, the BFS and primal-dual algorithms consistently outperform the Steiner tree algorithm. In many cases, they construct VPN trees that reserve half the bandwidth reserved by Steiner trees.

- The BFS algorithm scales well for large networks containing several thousand nodes.

In our implementation of Steiner trees, we used the 2-approximation primal-dual algorithm from [Hoc97].

6.1 Network Generation Models

In our experiments, we used two different network generators, to generate random networks with different characteristics. One generator was based on work by Waxman [Wax88], the other on work by Faloutsos et al. [FFF99]. We generated random and symmetric networks consisting of 50 to 5000 nodes connected by links with large residual capacities. The generation algorithms use the following models.

• **Waxman model [Wax88].** In this model, nodes are placed on a plane, and the probability for two nodes to be connected by a link decreases exponentially with the Euclidean distance between them. In our experiments, we used the Waxman model to generate networks of size less than 1000 nodes. We set the value for the parameter that controls the density of short edges in the network to 0.9 and the value of the parameter for the average node degree to 0.1.

• **Power-Law model [FFF99].** In this model, the node connectivity follows a power-law rule: very few nodes have high connectivity, and the number of nodes with lower connectivity increases exponentially as the connectivity decreases. This model is based on Internet measurements, where a node is an autonomous system (AS). In our experiments, we used the Power-Law model to generate large networks containing 1000 or more nodes.

A subset of the nodes in each network is chosen randomly and uniformly as the VPN endpoints. For the symmetric bandwidth case, each VPN endpoint is assigned bandwidth uniformly chosen from an interval of 2-100 Mbps. Further, to model asymmetric endpoint bandwidths, we introduce a new parameter, the asymmetry ratio r, which is essentially the ingress/egress bandwidth ratio for each VPN endpoint. The same ratio is also maintained for $\sum_l B^l_{in}$ and $\sum_l B^l_{out}$, the sums of ingress and egress bandwidths over all VPN endpoints.

6.2 Experimental Results

We compare the provisioning cost (that is, the total bandwidth reserved on links of the VPN tree) of the algorithms for the symmetric as well as the asymmetric bandwidth models. In the study, we examined the effect of varying the following three parameters on provisioning cost: (i) *network size*, (ii) *number of VPN nodes*, and (ii) *asymmetry ratio*.

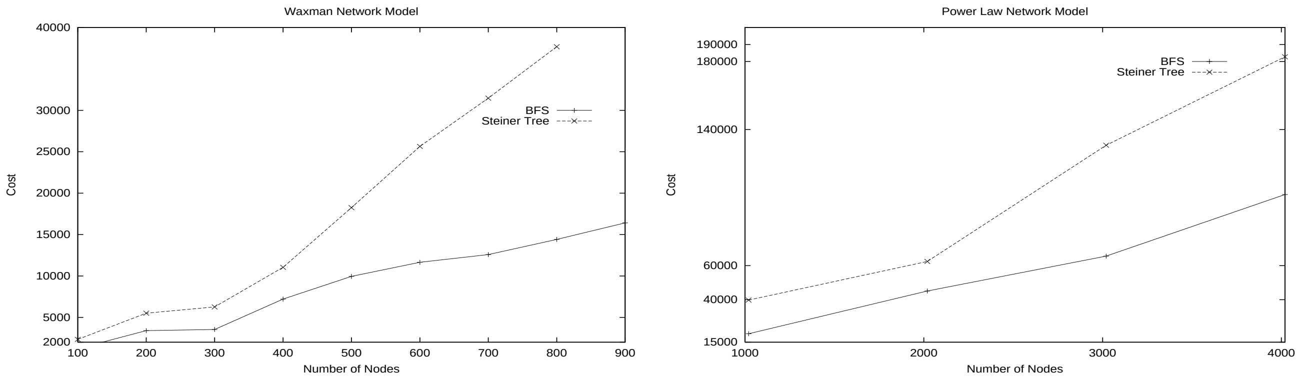


Figure 9: Effect of Number of Network Nodes on Performance of Algorithms.

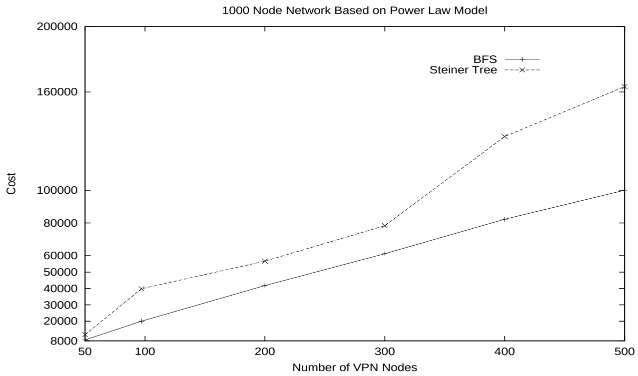


Figure 10: Effect of Number of VPN Nodes.

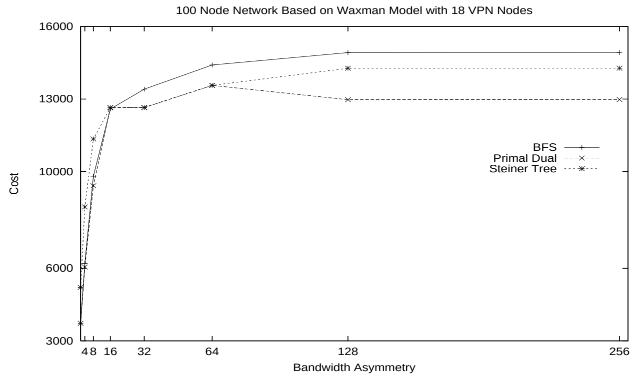


Figure 11: Effect of Asymmetry Ratio.

Network Size. Figure 9 depicts the provisioning cost of the BFS and Steiner tree algorithms as the number of network nodes is increased from 100 to 4000. VPN endpoints are assigned equal ingress/egress bandwidths and the number of VPN endpoints is set to 10% of the network size. Recall that the BFS algorithm is provably optimal for the symmetric case. Further, unlike the Steiner tree algorithm which is oblivious to the bandwidths of endpoints, the BFS algorithm does take into account the bandwidth requirements for VPN endpoints. As a result, it outperforms Steiner tree algorithm by almost a factor of 2 for a wide range of node values.

Number of VPN Nodes. Similar results are obtained for the BFS and Steiner tree approaches for a wide range of VPN node values (see Figure 10). In the experiment, the number of nodes in the network were fixed at 1000 and VPN endpoints were assigned symmetric bandwidths.

Asymmetry Ratio. In Figure 11, we plot the provisioning costs for the three algorithms as the asymmetry ratio is increased from 2 to 256. The network size and number of VPN nodes are fixed at 100 and 18, respectively. Interestingly, the primal-dual algorithm performs the best for the entire range of asymmetry ratios. For small values of the asymmetry ratio (≤ 8), the primal-dual algorithm behaves similar to the BFS algorithm which we showed to be optimal for a ratio of 1. Thus, both algorithms reserve less bandwidth than the Steiner tree algorithm for small ratio values.

As we increase the asymmetry ratio, the size of the steiner tree connecting the core nodes of the VPN tree also increases. Consequently, the cost of the VPN tree computed by the Steiner tree algorithm becomes smaller than the cost due to the BFS algorithm. However, the primal-dual algorithm performs the best since it estimates the cost of the VPN tree most accurately as consisting of a central core steiner tree component with multiple breadth first trees connecting the core to the VPN endpoints.

7 Concluding Remarks

In this paper, we developed novel algorithms for provisioning VPNs in the hose model. We connected VPN endpoints using a *tree* structure and our algorithms attempted to optimize the total bandwidth reserved on edges of the VPN tree. We showed that even for the simple scenario in which network links are assumed to have infinite capacity, the general problem of computing the optimal VPN tree is NP-hard. However, for the special case when the ingress and egress bandwidths for each VPN endpoint are equal, we proposed a *breadth first search* (BFS) algorithm for computing the optimal tree whose time complexity is $O(mn)$, where m and n are the number of links and nodes in the network, respectively. We presented a novel integer programming formulation for the general VPN tree computation problem (that is, when ingress and egress bandwidths of VPN endpoints are arbitrary) and devised an algorithm that is based on the *primal-dual* method. Finally, we extended our proposed algorithms for computing VPN trees when network links have capacity constraints. We showed that in the presence of link capacity constraints, computing the optimal VPN tree is NP-hard even when ingress and egress bandwidths of each endpoint are equal. Further, we also showed that computing an approximate solution that is within a constant factor of the optimum is as difficult as computing the optimal VPN tree itself.

Our experimental results indicate that the primal-dual algorithm performs the best over a wide range of parameter values, reserving less bandwidth than both the BFS and Steiner tree algorithms for large ingress/egress bandwidth ratios. For small bandwidth ratios, the BFS and primal-dual algorithms consistently outperform the Steiner tree algorithm. In many cases, they construct VPN trees that reserve half the bandwidth reserved by Steiner trees.

References

[AMO93] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin. *"Network Flows"*. Prentice Hall, 1993.

[DGG+98] N. G. Duffield, P. Goyal, A. Greenberg, P. Mishra, K. K. Ramakrishnan, and J. E. van der Merwe. A flexible model for resource management in virtual private networks. In *Proceedings ACM SIGCOMM*, 1998.

[DR00] B. Davie and Y. Rekhter. *"MPLS Technology and Applications"*. Morgan Kaufmann Publishers, 2000.

[FFF99] M. Faloutsos, P. Faloutsos, and C. Faloutsos. On power-law relationships of the internet topology. In *Proceedings ACM SIGCOMM*, 1999.

[GJ79] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman and Company, 1979.

[Hoc97] D. S. Hochbaum. *"Approximation Algorithms for NP-Hard Problems"*. PWS Publishing Company, 1997.

[KA98] S. Kent and R. Atkinson. Security architecture for the internet protocol. *RFC 2401*, November 1998.

[LV92] J. H. Lin and J. H. Vitter. ϵ-approximations with minimum packing constraint violation. In *Proceedings ACM Symposium on Theory of Computing*, 1992.

[Que93] M. Queyranne. Structure of a simple scheduling polyhedron. *Mathematical Programming*, 58:163–185, 1993.

[STA97] D. Shmoys, E. Tardos, and K Aardal. Approximation algorithms for facility location problems. In *Proceedings ACM Symposium on Theory of Computing*, 1997.

[Wax88] B. M. Waxman. Routing of multipoint connections. *IEEE Journal on Selected Areas in Communications*, 6(9):1617–1622, December 1988.

– This appendix contains material that can be read at the discretion of the reviewer and has been included only for the purpose of completeness. –

A Proofs for Asymmetric Ingress and Egress Bandwidths

Proof of Theorem 4.2: We reduce the Steiner tree problem to the problem of computing an optimal VPN tree. The Steiner tree problem is as follows: given a graph G and set of endpoints P, compute a tree connecting the endpoints and containing the minimum number of edges. Suppose for each VPN endpoint j, the ingress bandwidth requirement is 1, that is, $B_j^{in} = 1$, and the egress bandwidth requirement is a large constant greater than $|P|$, that is, $B_j^{out} > |P|$. Note that $\sum_{l \in P} B_l^{in} \le |P| < B_j^{out}$, for every VPN endpoint j. Thus, for each edge (i,j), the bandwidth reserved in the two directions is bounded by the ingress bandwidths of VPN endpoints in P_i and P_j – specifically, $C_T(i,j) = \sum_{l \in P_j^{(i,j)}} B_l^{in}$ and $C_T(j,i) = \sum_{l \in P_i^{(i,j)}} B_l^{in}$. Thus, the total bandwidth reserved on edge (i,j) in the two directions is $C_T(i,j) + C_T(j,i) = \sum_{l \in P} B_l^{in} = |P|$. Since $|P|$ is a constant, the cost of a VPN tree T connecting the endpoints in P is simply $|P|$ times the number of edges in T. Thus, the optimal VPN tree is the one with the fewest edges, or alternately, the Steiner tree connecting the VPN endpoints in P. ▮

Proof of Lemma 4.4: We show that if an edge (i,j) is biased towards j, then every other edge (u,w) in $P_j^{(i,j)}$ (such that w is further from j than u) is also biased towards w. Thus, it is not possible for balanced edges to be separated by biased edges – as a result, the set of balanced edges form a connected component.

For an edge (i,j) that is biased towards j, the following is true: (1) $\sum_{l \in P_j^{(i,j)}} B_l^{in} \le \sum_{l \in P_i^{(i,j)}} B_l^{out}$, and (2) $\sum_{l \in P_j^{(i,j)}} B_l^{out} \le \sum_{l \in P_i^{(i,j)}} B_l^{in}$. For edge (u,w) in $P_j^{(i,j)}$, $P_w^{(u,w)} \subset P_j^{(i,j)}$ and $P_i^{(i,j)} \subset P_u^{(u,w)}$. Thus, it follows that (1) $\sum_{l \in P_w^{(u,w)}} B_l^{in} < \sum_{l \in P_j^{(i,j)}} B_l^{in} \le \sum_{l \in P_i^{(i,j)}} B_l^{out} < \sum_{l \in P_u^{(u,w)}} B_l^{out}$, and (2) $\sum_{l \in P_w^{(u,w)}} B_l^{out} < \sum_{l \in P_j^{(i,j)}} B_l^{out} \le \sum_{l \in P_i^{(i,j)}} B_l^{in} < \sum_{l \in P_u^{(u,w)}} B_l^{in}$. Thus, edge (u,w) is biased towards w. ▮

Proof of Lemma 4.5: We show that every biased edge (v,w) incident on v in C_v is biased toward w. Suppose this were not true and edge (v,w) were biased towards v. Then, due to Lemma 4.4, it follows that every other edge incident on v would be biased away from v and thus, v cannot be a core node - which leads to a contradiction. Thus, if every edge (v,w) incident on v in C_v is biased toward w, then, due to Lemma 4.4, it follows that every edge (i,j) in C_v is biased towards j. ▮

Proof of Lemma 4.6: Due to Lemma 4.5, every edge (i,j) in C_v (with j further from node v than i) is biased towards j. As a result, the total bandwidth reserved on edge (i,j) is $\sum_{l \in P_j^{(i,j)}} (B_l^{in} + B_l^{out})$. Thus, VPN site l contributes $(B_l^{in} + B_l^{out})$ to the cost of each edge (i,j) on the path from v to l, and to no other edge. Thus, the cost of the component C_v is given by $\sum_{l \in (C_v \cap P)} d_T(v,l) * (B_l^{in} + B_l^{out})$. ▮

Proof of Lemma 4.7: The tree $T(S)$ consists of two types of edges – (1) Steiner tree edges connecting the nodes in S, and (2) edges in the breadth first tree connecting the nodes in S to VPN endpoints in P. We show that the cost of the Steiner tree edges is at most $M * b$. Also, we show that each VPN endpoint l contributes no more than $\min_{v \in S}\{d_G(v,l)\} * (B_l^{in} + B_l^{out})$ to the edges of the breadth first tree. This proves the lemma.

Let B be the set of Steiner tree edges connecting nodes in S in $T(S)$. Note that the cost of each edge can be at most M. This is true of balanced edges in $T(S)$ (due to Lemma 4.3). We show that this holds even for biased edges. Suppose edge (i,j) is biased towards i. Thus, $\sum_{l \in P_i^{(i,j)}} B_l^{in} \le \sum_{l \in P_j^{(i,j)}} B_l^{out}$ and $\sum_{l \in P_i^{(i,j)}} B_l^{out} \le \sum_{l \in P_j^{(i,j)}} B_l^{in}$. Thus, the total bandwidth reserved on edge (i,j) is $\sum_{l \in P_i^{(i,j)}} B_l^{in} + \sum_{l \in P_i^{(i,j)}} B_l^{out}$. Suppose $M = \sum_{l \in P} B_l^{in}$. If the bandwidth reserved on (i,j) exceeds M, then this would imply that $\sum_{l \in P_i^{(i,j)}} B_l^{out} > \sum_{l \in P_j^{(i,j)}} B_l^{in}$, which would lead to a contradiction (since (i,j) is biased towards i). Thus, since the edges connecting the nodes in S in $T(S)$ form a Steiner tree each of whose cost is at most M, the cost of these edges in $T(S)$ is less than $M * b$,

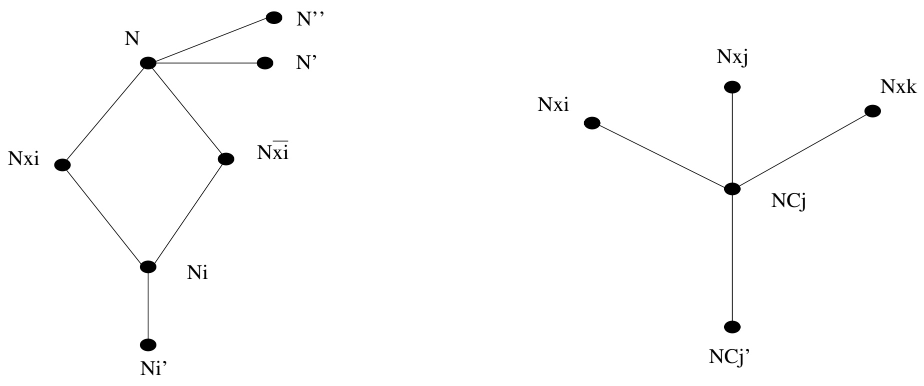


Figure 12: Edges in Graph for Literals and Clauses

where b is the number of edges in the Steiner tree connecting the nodes in S.

Further, consider a component C_v of $T(S)$ containing node $v \in S$ after edges in B are deleted from $T(S)$. Each edge (i, j) (j being further from v than i) has a cost that can be at most $\sum_{l \in P_j^{(i,j)}} (B_l^{in} + B_l^{out})$. Thus, each VPN endpoint l in C_v only contributes to edges in C_v along the path from v to l. As a consequence, the contribution of each VPN endpoint l to edges of the breadth first tree is no more than $d_{T(S)}(v, l) * (B_l^{in} + B_l^{out})$. Further, since $T(S)$ is a breadth first tree rooted at the supernode resulting from collapsing nodes in S into one, it follows that $d_{T(S)}(v, l) \leq \min_{u \in S}\{d_G(u, l)\}$. Thus, the sum of the costs of edges in the breadth first tree due to all the endpoints is less than or equal to $\sum_{l \in P} \min_{v \in S}\{d_G(v, l)\} * (B_l^{in} + B_l^{out})$. Thus, the lemma follows – that is, $C_{T(S)} \leq C_S$. ∎

Proof of Lemma 4.8: The cost of VPN tree T consists of two components – (1) the cost of the balanced edges, and (2) the cost of the components C_v corresponding to each core node $v \in core(T)$. First, observe that the cost of the Steiner tree connecting nodes in $core(T)$ is less than or equal to $M * |bal(T)|$, the sum of the costs of the balanced edges in T (due to Lemma 4.4, the set of balanced edges in T form a connected component and so form a tree that connects the nodes in $core(T)$). Also, for each VPN endpoint l, if C_v is the component of T containing l for $v \in core(T)$ (after balanced edges are deleted from T), then $\min_{u \in core(T)}\{d_G(u, l)\} * (B_l^{in} + B_l^{out})$ is less than or equal to $d_T(v, l) * (B_l^{in} + B_l^{out})$. Thus, the lemma is proved. ∎

B Proofs for Networks with Link Capacity Constraints

Proof of Theorem 5.1: We show a reduction from 3-SAT. In 3-SAT, we are given a boolean expression $C = C_1 \wedge \cdots \wedge C_p$ with each conjunct C_i containing 3 literals from among $x_1, \ldots, x_q$. The problem is to determine if there exists an assignment of truth values to the literals such that C is satisfiable.

We construct our graph G (with residual capacity R_{ij} for edge (i, j)) and the set of VPN endpoints P as follows. For each literal x_i, graph G contains three nodes – N_i, N_{x_i} and $N_{\bar{x_i}}$. For each conjunct C_j, G contains node N_{C_j}. In addition, G contains one special node N. The edges in graph G are as follows. For each literal x_i, G contains four edges (N, N_{x_i}), $(N, N_{\bar{x_i}})$, (N_{x_i}, N_i) and $(N_{\bar{x_i}}, N_i)$ (see Figure 12). In addition, for each conjunct C_j, G contains edge (N_{C_j}, N_{x_i}) if C_j contains literal x_i; otherwise, if C_j contains literal $\bar{x_i}$, then G contains edge $(N_{C_j}, N_{\bar{x_i}})$. Thus, corresponding to each conjunct, G contains three edges, one edge for each literal contained in C_j. Finally, P contains one VPN endpoint N_i' connected to N_i for each literal x_i, one VPN endpoint N_{C_j}' connected to each node N_{C_j} and two VPN endpoints N' and N'' connected to node N. The residual capacity of all edges in G is assumed to be a fairly large constant B ($B > p$), and the bandwidth requirements for N_i', N_{C_j}', N' and N'' are B, 1, B and B, respectively (endpoints are assumed to have equal ingress and egress bandwidths).

We now show that there exists a feasible VPN tree connecting the VPN endpoints if and only if C is satisfiable.

Suppose C is satisfiable. We show that there exists a feasible VPN tree. Since C is satisfiable, there exists an assignment of truth values to each literal x_i such that each conjunct C_j is true. We construct the feasible VPN tree T as follows. First, note that T must contain the edges (N_i, N_i'), (N_{C_j}, N_{C_j}'), (N, N') and (N, N'') since these are the only edges incident on the VPN endpoints. Further, for a literal x_i, if x_i is true in the assignment, then edges $(N, N_{\bar{x}_i})$ and $(N_{\bar{x}_i}, N_i)$ are added to T; otherwise, if x_i is false in the assignment, then edges (N, N_{x_i}) and (N_{x_i}, N_i) are added to T. Note that the bandwidth to be reserved on each of these edges is at most B, the bandwidth requirement for VPN endpoint N_i'. Next, for each conjunct C_j, since it is true, there is some literal x_i or $\bar{x}_i$ that is assigned a true value. Consider one such literal in C_j that is assigned a true value. If C_j contains x_i which is true, then edges (N, N_{x_i}) and (N_{x_i}, N_{C_j}) are added to T. On the other hand, if C_j contains $\bar{x}_i$ which is true, then edges $(N, N_{\bar{x}_i})$ and $(N_{\bar{x}_i}, N_{C_j})$ are added to T. For both cases, one can show that the reserved bandwidth does not exceed B. For example, the first case (when x_i is true), the only edges in T incident on N_{x_i} are (N_{x_i}, N_{C_j}) for conjuncts C_j containing x_i. Since the bandwidth requirement for each VPN endpoint N_{C_j}' is 1 and the number of conjuncts p is less than B, the bandwidth to be reserved on edge (N, N_{x_i}) is at most B. Thus, the constructed VPN tree T is feasible and connects the VPN endpoints in P.

We next show that if there exists a feasible VPN tree connecting the VPN endpoints in P, then C is satisfiable. Suppose there exists a feasible VPN tree T. We begin by showing that for every VPN endpoint N_i', either edges $(N, N_{\bar{x}_i})$ and $(N_{\bar{x}_i}, N_i)$ are in T or edges (N, N_{x_i}) and (N_{x_i}, N_i) are in T. To show this, first observe that since both nodes N and N_i are connected to VPN endpoints, it must be the case that node N_i is connected to node N in T. Suppose N_i is connected to N via the sequence of edges (N_i, N_{x_i}), (N_{x_i}, N_{C_j}), (N_{C_j}, N_{x_l}), $\ldots$ to node N. Then, since N_i' and N_{C_j}' (connected to nodes N_i and N_{C_j}) are both VPN endpoints with bandwidth requirements B and 1, respectively, and N' and N'' are VPN endpoints connected to N with a total bandwidth requirement of $2B$, the bandwidth to be reserved on edge (N_{C_j}, N_{x_l}) must be at least $B + 1$, which would make T infeasible. Thus, for every VPN endpoint N_i', either edges $(N, N_{\bar{x}_i})$ and $(N_{\bar{x}_i}, N_i)$ are in T or edges (N, N_{x_i}) and (N_{x_i}, N_i) belong to T.

We assign x_i a value true if edges $(N, N_{\bar{x}_i})$ and $(N_{\bar{x}_i}, N_i)$ are in T; else if edges (N, N_{x_i}) and (N_{x_i}, N_i) are in T, we assign $\bar{x}_i$ a value of true. Next, we show that if x_i is assigned true, then edge $(N_{C_j}, N_{\bar{x}_i})$ cannot belong to T. The reason for this is that then T would contain edges $(N, N_{\bar{x}_i})$, $(N_{\bar{x}_i}, N_i)$ and $(N_{C_j}, N_{\bar{x}_i})$ causing the bandwidth to be reserved on edge $(N, N_{\bar{x}_i})$ to exceed B. Similarly, we can show that if $\bar{x}_i$ is assigned true, then edge (N_{C_j}, N_{x_i}) cannot belong to T. Since T is a VPN tree, node N_{C_j} must be connected to node N through an edge (N_{C_j}, N_{x_i}) or $(N_{C_j}, N_{\bar{x}_i})$ for a literal x_i or $\bar{x}_i$ contained in C_j. If (N_{C_j}, N_{x_i}) is in T, then x_i must be true and C_j is true. On the other hand, if $(N_{C_j}, N_{\bar{x}_i})$ is in T, then $\bar{x}_i$ must be true and C_j is true. Thus, C is satisfiable. ∎

Proof of Corollary 5.2: We show a reduction from 3-SAT to the problem of determining if there exists a feasible VPN tree whose cost is within a factor c of the optimal cost. The graph G that we construct for the boolean expression C contains all the nodes and edges as described in the proof of Theorem 5.1 above. In addition, we add a separate path containing $2qc + \frac{2pc}{B}$ edges between each node N_i and N. The residual capacity of each edge along each of the paths is B. Essentially, if any of these long paths is chosen in VPN tree T to connect N_i and N, then the cost of T exceeds $4qBc + 4pc$, since $2B$ is the bandwidth that needs to be reserved (B in each direction) on each edge along the path. In the proof of Theorem 5.1, we showed that in a feasible VPN tree that does not contain any of the long paths, each VPN endpoint N_i' or N_{C_j}' is connected to N via paths consisting of two edges (if we were to include the edge incident on the endpoint itself, then each path would contain three edges). Thus, the cost of a feasible VPN tree that excludes all long paths is at most $4qB + 4p$ (excluding the cost of edges incident on VPN endpoints). As a result, one can show (in a manner similar to the proof of Theorem 5.1) that C is satisfiable if and only if there is a feasible VPN tree T whose cost is equal to $4qB + 4p$ (that is, T does not contain any of the long paths). Since a long path adds $4qBc + 4pc$ to the cost of a VPN tree, one can show that C is satisfiable if and only if there is a feasible VPN tree T whose cost is less than or equal to $4qBc + 4pc$. Thus, it follows that the problem of determining if there exists a feasible VPN tree whose cost is within a constant factor c of the optimal cost is NP-hard. ∎